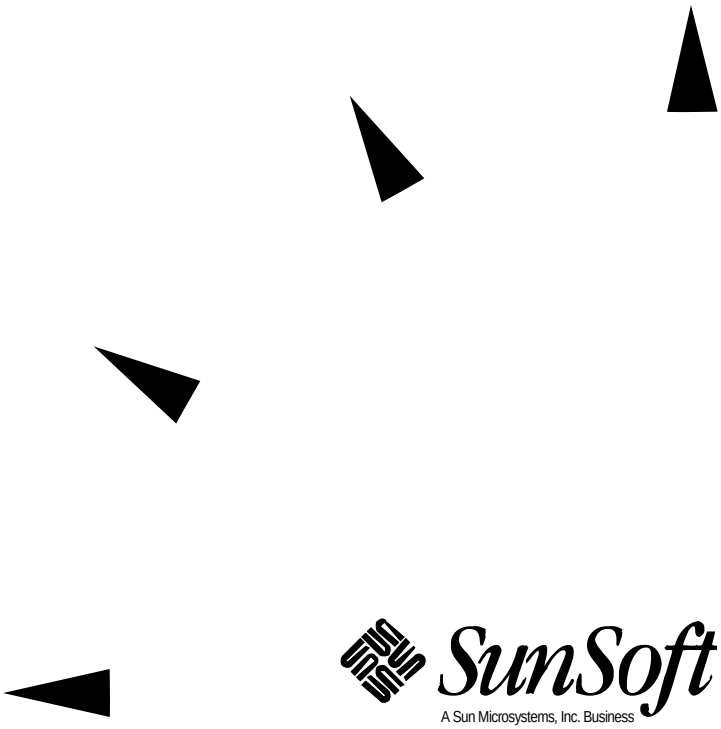
OpenStep Programming Reference



2550 Garcia Avenue
Mountain View, CA 94043
U.S.A.

Part No 802-2112-10
Revision A, September 1996

SunSoft
A Sun Microsystems, Inc. Business

Contents

Introduction

This book describes the Application Programming Interface (API) for the OpenStep™ development environment. OpenStep is an operating system independent, object-oriented application layer, based on NeXT's advanced object technology. See the "Further Reading" section at the end of this introduction for a description of the other books in the OpenStep documentation set.

The OpenStep software is divided into the following kits:

- The Application Kit contains the software for writing applications that use windows, draw on the screen, and respond to user actions on the keyboard and mouse.
- The Foundation Kit provides the fundamental building blocks that applications use to manage data and resources. It defines facilities for handling multibyte character sets, object persistency and distribution, and provides an interface to common operating system facilities.
- Display Postscript Kit provides OpenStep with its device-independent imaging model.
- The Sound Kit provides software for capturing, manipulating, reading, and writing sounds.

Each kit is a combination of Objective C classes and protocols, along with C functions, types, and constants. Please note that many of the types used for method argument and return values in this book are defined in the Objective C language. These include:

- `BOOL`
- `Class`
- `id`
- `IMP`
- `nil`
- `Protocol`
- `SEL`

In addition, the type codes used to encode method argument and return types for archiving and other purposes are also defined in the Objective C language.

Book Organization

This book contains sections for each kit. Each section contains chapters describing the kit's classes, protocols, functions, and types and constants. The following outline shows the book organization:

- Application Kit
  - Classes
  - Protocols
  - Functions
  - Types and Constants
- Foundation Kit
  - Classes
  - Protocols
  - Functions
  - Types and Constants
- Display PostScript System
  - Classes
  - Protocols
  - Display PostScript Operators
  - Client Library Functions
  - Single-Operator Functions
  - Types and Constants
- Sound Kit
  - Classes
  - Types and Constants

Chapter Organization

The following sections desribe each chapters organization, and briefly discusses:

- Inheritance Hierarchies
- Delegates
- Formal and Informal Protocols

Classes

Each class chapter contains a kit's class descriptions listed alphabetically. Each class description starts with a table listing the class's inheritance hierarchy, protocol conformance, and header file containing the class interface. For example:

Inherits From:	NSCell : NSObject
Conforms To:	NSCoding, NSCopying (NSCell), NSObject (NSObject)
Declared In:	AppKit / NSActionCell.h

The first line in this example specifies a class's inheritance hierarchy, in this case `NSActionCell`'s inheritance hierarchy. It specifies that `NSActionCell` inherits from `NSCell`, and that `NSCell` inherits from `NSObject`. `NSCell` is called `NSActionCell`'s *superclass*. `NSObject` is the root class of almost all OpenStep inheritance hierarchies.

The second line in the previous example specifies the formal protocols that the class conforms to. These include both protocols the class adopts and those it inherits from other adopting classes. If inherited, the name of the adopting class in given in parentheses.

The third line in the previous example specifies the header file that declares the class interface.

After a general description of the class, the class's methods are listed in functional groups. For example, here is NSScroller's list:

Activity	Class Method
Laying out the NSScroller	+ scrollerWidth – arrowsPosition – checkSpaceForParts – rectForPart: – setArrowsPosition: – usableParts
Setting the NSScroller's values	– knobProportion – setFloatValue:knobProportion:
Displaying	– drawArrow:highlight: – drawKnob – drawParts – highlight:
Handling events	– hitPart – testPart: – trackKnob: – trackScrollButtons:

This list is followed by class methods in alphabetical order, then instance methods in alphabetical order. Each method's prototype is given, followed by a brief description.

Some classes have separate sections with titles such as "Methods Implemented by the Superview", "Methods Implemented by ", "Methods Implemented by the Owner." These are informal protocols. They document methods that can or must be implemented to receive messages on behalf of instances of the class.

Methods Implemented by the Delegate

If the class describes a delegate, the delegate methods are listed last. These are not methods defined in the class; they are methods that you can define to respond to messages sent from instances of the class. If you define a delegate method, the delegate will receive automatic messages to perform that delegate method at the appropriate time. For example, if you define a `windowDidBecomeKey:` method for an `NSWindow` object's delegate, the delegate will receive `windowDidBecomeKey:` messages whenever the

`NSWindow` object becomes the key window. Messages are sent to an object's delegate only if you define a method that can respond to the message within the delegate.

In essence, this section documents an informal protocol. But because these methods are so closely tied to the behavior of a particular class, they're documented with the class rather than in the Protocols chapters.

Protocols

The protocol chapters describe OpenStep's formal and informal protocols. Formal protocols are declared using the `@protocol` compiler directive. They can be formally adopted and implemented by a class and tested by sending an object a conformsToProtocol: message.

Some formal protocols are adopted and implemented by OpenStep classes. However, many formal protocols are declared by a kit, but not implemented by it. These formal protocols list methods that you can implement to respond to kit-generated messages.

A few formal protocols are implemented by a kit, but not by a class that's part of the OpenStep API. Rather, the protocol is implemented by an anonymous object that the kit supplies. The protocol lets you know what messages you can send to the object.

Like formal protocols, informal protocols declare a list of methods that others are invited to implement. If an informal protocol is closely associated with one particular class, for example, the list of methods implemented by the delegate, it's documented in the class description.

Note – Informal protocols associated with more than one class, or not associated with any particular class, are documented in the Protocols chapters.

Protocol information is organized into many of the same sections as described previously for a class specification. But protocols are not classes and therefore differ somewhat in the kind of information provided.

Each formal protocol description starts with a table listing the classes that adopt the protocol, and the header file containing the protocol description. For example:

Adopted By:	NSText
Declared In:	AppKit/NSSpellProtocol.h

Many protocols declare methods that applications must implement and so are not adopted by any OpenStep classes. Some protocols are implemented by anonymous objects (instances of an unknown class); the protocol is the only information available about what messages the object can respond to. Protocols that have an implementation available through an anonymous object generally don't have to be reimplemented by other classes.

An informal protocol cannot be formally adopted by a class and it cannot formally incorporate another protocol. So its description begins with information about the category where it's declared:

Category Of:	NSObject
Declared In:	AppKit/NSDragging.h

Informal protocols are typically declared as categories of the `NSObject` class. This gives them the widest possible scope. If the protocol includes enough methods to warrant it, they're divided by type and presented just as the methods of a class are.

Functions

Within the function chapters related functions are grouped together under a heading that describes that groups similarities. Here is a partial list of these headings from the Application Kit:

- Rectangle Drawing Functions
- Color Functions
- Text Functions
- Array Allocation Functions
- Imaging Functions

Each function, its arguments, and its return value are briefly described in an accompanying comment.

Types and Constants

Within these chapters related defined types, enumeration constants, symbolic constants, structures, and global variables are grouped together under a heading that describes where the type or constant is used. Here is a partial list of these headings from the Application Kit:

- Application
- Box
- Buttons
- Cells and Button Cells
- Color

A short description accompanies each group.

Further Reading

In addition to this document, the OpenStep documentation set consists of:

- *QuickStart to Using the OpenStep Desktop*—for beginning end-users. A minimal set of instructions to get you started running OpenStep.
- *Using the OpenStep Desktop*—the complete end-user guide.
- *User Interface Guidelines*—for application developers; identifies the objects supplied in the Application Kit, describes their appearances and behaviors, and the kinds of application-specific behaviors that developers must implement. Includes detailed discussions of the mouse and keyboard operations performed by users to operate the interface. Provides detailed guidelines for such things as choosing keyboard shortcut characters. Also describes the behaviors that should be implemented for custom objects.
- *OpenStep Development Tools*—describes the essential tools for developing an OpenStep application: the Project Builder, Interface Builder, Header Viewer, Icon Builder, Edit applications, and the distributed Debugger. The manual also included chapters on the Objective C language and the `NSObject` class.

Part 1 — Application Kit

Classes 1

The Application Kit classes are the core of OpenStep. They describe OpenStep's appearance and behavior. The following sections give an overview of the Application Kit classes.

Encapsulating an Application

The central class of the Application Kit is `NSApplication`. Every application that uses the Application Kit is a single `NSApplication` object, known to your program as `NSApp`. Your `NSApplication` object

- Keeps track of the application's windows and menus
- Controls the main event loop
- Opens Interface Builder files (with support from the `NSAwakening` protocol)
- Maintains information about printing, languages, screens, color support, and so on

General Drawing and Event Handling

The `NSWindow` and `NSView` classes are the centerpieces of drawing. More specifically, `NSWindow` objects represent rectangular areas on the screen in which the user works. To the extent that everything the user does is directed to an `NSWindow`, an application's set of `NSWindow`s *is* the application. `NSView`s are areas within `NSWindow`s that perform your application's drawing.

NSPanel is a subclass of NSWindow that you use to display transient, global, or pressing information. For example, you would use a Panel, rather than an instance of NSWindow, to display error messages, or to query the user for a response to remarkable or unusual circumstances.

The NSResponder class defines the *responder chain*, an ordered list of objects that respond to user events. When the user clicks the mouse or presses a key, an event is generated and passed up the responder chain in search of an object that can respond to it.

Menus and Cursors

The NSMenu, NSMenuCell, and NSCursor classes define the look and behavior of the menus and cursors that your application displays to the user.

Grouping and Scrolling Views

The NSBox, NSSplitView, and NSScrollView classes provide graphic widgets to some other NSView or collection of NSViews. An NSBox groups some number of other NSViews, and lets you draw a border around the entire group. NSSplitView lets you stack NSViews vertically, apportioning to each NSView some amount of a common territory; a sliding control bar lets the user redistribute the territory among NSViews. NSScrollView, and its helper NSClipView, provide a scrolling mechanism as well as the graphic objects that let the user initiate and control a scroll.

Controlling an Application

The NSControl and NSCell classes, and their subclasses, define an easily recognized set of buttons, sliders, and browsers that the user can manipulate graphically to control some aspect of your application. Just what a particular control affects is up to you: When a control is "touched," it sends a specific message to a specific object. This is the *targeted/action paradigm*; for each NSControl, you define both the target (an object) and the action (the message that's sent to that object).

An NSCell completes the implementation of an NSControl. In general, for each NSControl there is a corresponding NSCell; thus a button comprises a NSButton and an NSButtonCell, a slider is an NSSlider and an NSSliderCell, and so on.

Text and Fonts

Most applications display text in some form. The `NSCStringText` and `NSTextField` classes make this presentation as straightforward and simple as possible. The size of the `NSCStringText` class is daunting at first, but for simple text presentation only a handful of methods are actually needed (or you can use the streamlined `NSTextField` class). More complicated text-based applications, such as word processors, can take advantage of the `NSCStringText` class' more sophisticated features, such as rulers and break tables.

The `NSFont` and `NSFontManager` classes encapsulate and manage different font families, sizes, and variations. The `NSFont` class defines a single object for each distinct font. For efficiency, these objects, which can be large, are shared by all the objects in your application. The `NSFontPanel` class defines the font-specification panel that's presented to the user.

Graphics and Color

The `NSImage`, `NSImageRep`, and the other image representation classes encapsulate graphic data, allowing you to easily and efficiently access images stored in files on the disk. The presentation of an image is greatly influenced by the hardware that it's displayed on. For example, a particular image may look good on a color monitor, but may be too "rich" for monochrome. Through the image classes, you can group representations of the same image, where each representation fits a specific type of display device—the decision of which representation to use can be left to the `NSImage` class itself.

Colors are represented by the `NSColor` class. Applications incorporate and support colors by using the `NSColorPanel`, `NSColorList`, `NSColorPicker`, and `NSColorWell` classes. These classes let the user to select and apply colors. The `NSColorPicking` protocol lets you extend the standard color panel.

The four standard color formats—RGB, CMYK, HSB, and grayscale—are recognized by the color classes. You can also tell the classes to recognize custom representations.

1

Printing and Faxing

The NSPrinter, NSPageLayout, and NSPrintInfo classes work together to provide the means for printing and faxing the information that your application displays in its NSWindows and NSViews. For more control, the NSWindow and NSView classes define methods that can fine-tune the printing and faxing mechanism.

Accessing the File System

The Application Kit does not provide a class that defines objects to correspond to files on the disk. However, the NSOpenPanel and NSSavePanel provide a convenient and familiar user interface to the file system.

Sharing Data with Other Applications

The NSPasteboard class defines a repository for data that's copied from your application, making this data available to any application that cares to use it. This is the familiar cut-copy-paste mechanism. The NSServicesRequest protocol uses the NSPasteboard to communicate data that's passed between applications by a registered service.

An intimate link between applications can be created through the NSDataLink, NSDataLinkManager, NSDataLinkPanel, and NSSelection classes. Through these classes, multiple applications can share the same data. A change to the data in one application is seen immediately in all others that display that data.

Spell-Checking

The NSSpellServer class lets you define a spell-checking facility and provide it as a service to other applications. To connect your application to a spelling checker, you use the NSSpellChecker class. The NSIgnoreMisspelledWords, and NSChangeSpelling protocols support the spell-checking mechanism.

Application Kit Class Hierarchy

The Application Kit contains over 60 classes which inherit directly or indirectly from `NSObject`, the root class defined in the Foundation Kit. The following diagram shows the Application Kit's class inheritance relationships.

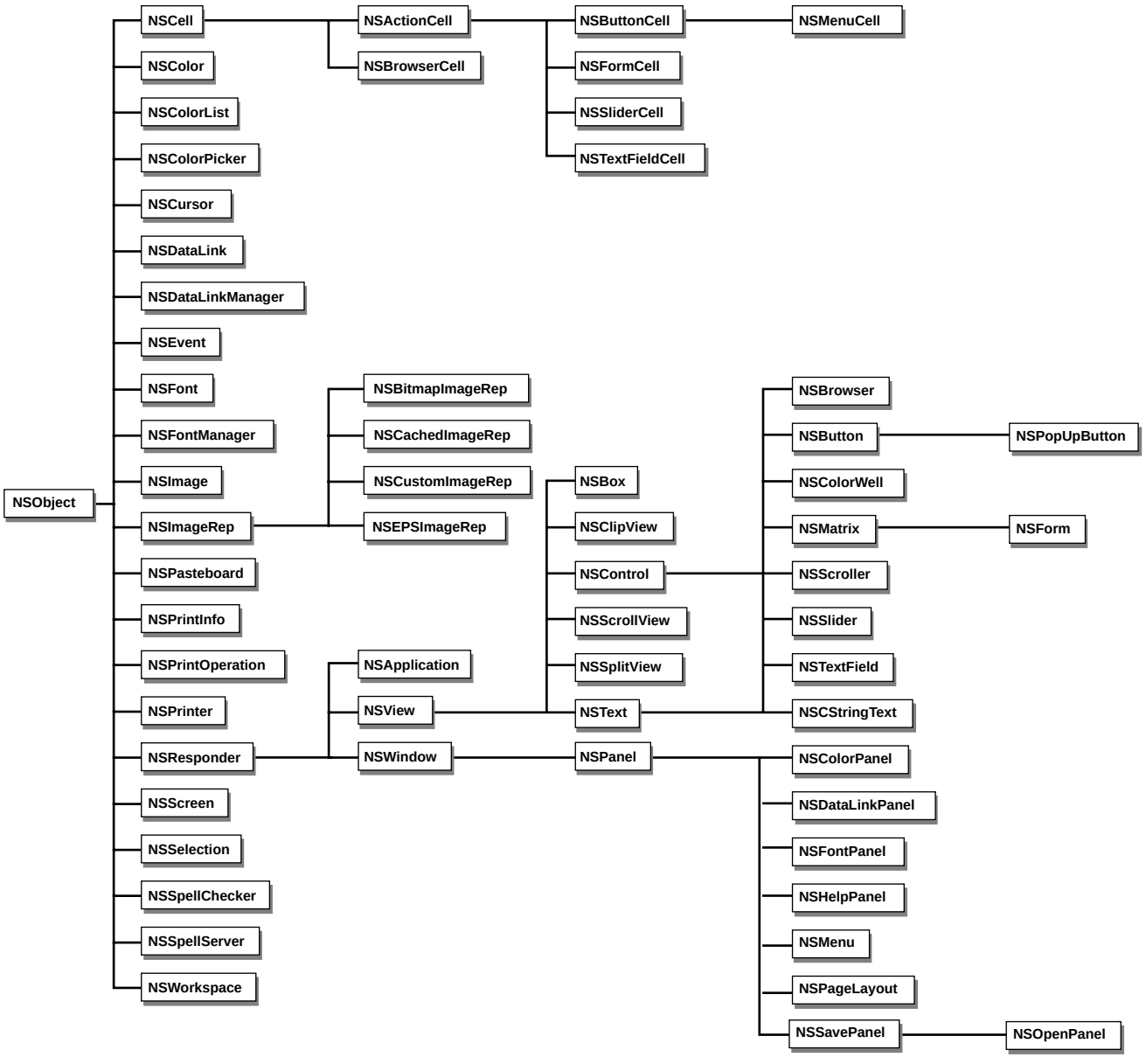


Figure 1-1 Application Kit Classes

NSActionCell

Inherits From:	NSCell : NSObject
Conforms To:	NSCoding, NSCopying (NSCell), NSObject (NSObject)
Declared In:	AppKit/NSActionCell.h

Class Description

An `NSActionCell` defines an active area inside a control (an instance of `NSControl` or one of its subclasses). As an `NSControl`'s active area, an `NSActionCell` does three things: it usually performs display of text or an icon (the subclass `NSSliderCell` is an exception); it provides the `NSControl` with a target and an action; and it handles mouse (cursor) tracking by properly highlighting its area and sending action messages to its target based on cursor movement. The only way to specify the `NSControl` for a particular `NSActionCell` is to send the `NSActionCell` a `drawWithFrame:inView:` message, passing the `NSControl` as the argument for the `inView:` keyword of the method.

`NSActionCell` implements the target object and action method as defined by its superclass, `NSCell`. As a user manipulates an `NSControl`, `NSActionCell`'s `trackMouse:inRect:ofView:untilMouseUp:` method (inherited from `NSCell`) updates its appearance and sends the action message to the target object with the `NSControl` object as the only argument.

Usually, the responsibility for an `NSControl`'s appearance and behavior is completely given over to a corresponding `NSActionCell`. (`NSMatrix`, and its subclass `NSForm`, are `NSControl`s that don't follow this rule.)

A single `NSControl` may have more than one `NSActionCell`. To help identify it in this case, every `NSActionCell` has an integer tag. Note, however, that no checking is done by the `NSActionCell` object itself to ensure that the tag is unique. See the `NSMatrix` class for an example of a subclass of `NSControl` that contains multiple `NSActionCell`s.

Many of the methods that define the contents and look of an `NSActionCell`, such as `setFont:` and `setBordered:`, are reimplementations of methods inherited from `NSCell`. They are subclassed to ensure that the `NSActionCell` is redisplayed if it's currently in an `NSControl`.

Method Types

Activity	Class Method
Configuring an NSActionCell	– setAlignment: – setBezeled: – setBordered: – setEnabled: – setFloatingPointFormat:left:right: – setFont: – setImage:
Manipulating NSActionCellValues	– doubleValue – floatValue – intValue – setStringValue: – stringValue
Displaying	– drawWithFrame:inView: – controlView
Target and action	– action – setAction: – setTarget: – target
Assigning a tag	– setTag: – tag

Instance Methods

action

```
- (SEL)action
```

Returns the action cell's action method. Keep in mind that the argument of an action method sent by an action cell is its associated `NSControl` (the object returned by `controlView`). See also `setAction:`, `target`, `controlView`.

controlView

```
- (NSView *)controlView
```

Returns the view (normally an `NSControl`) in which the action cell was last drawn. In general, your code should use the object returned by this method only to redisplay indirectly the action cell. For example, the subclasses of `NSActionCell` defined by the Application Kit invoke this method in order to

send the NSControl a message such as updateCellInside:. The NSControl in which an action cell is drawn is set automatically by the drawWithFrame:inView: method. You can't explicitly set the NSControl. See also drawWithFrame:inView:.

doubleValue

```
- (double)doubleValue
```

Returns the action cell's contents as a double-precision floating point number. If the action cell is being edited when this message is received, editing is validated first. See also setDoubleValue: (NSCell), floatValue, intValue, stringValue, validateEditing (NSControl).

drawWithFrame:inView:

```
- (void)drawWithFrame:(NSRect)cellFrame
    inView:(NSView *)controlView
```

Draws the action cell in the rectangle cellFrame of controlView (which should normally be an NSControl). This sets the action cell's control to controlView and performs the drawing if and only if controlView is an NSControl object (an instance of NSControl or a subclass thereof). Focus must be locked on the NSControl before invoking this method. The NSControl automatically performs this locking. See also drawWithFrame:inView: (NSCell).

floatValue

```
- (float)floatValue
```

Returns the action cell's contents as a single-precision floating point number. If the action cell is being edited when this message is received, editing is validated first. See also setFloatValue: (NSCell), doubleValue, intValue, stringValue, validateEditing (NSControl).

intValue

```
- (int)intValue
```

Returns the action cell's contents as an integer. If the action cell is being edited when this message is received, editing is validated first. See also `setIntValue:` (`NSCell`), `doubleValue`, `floatValue`, `stringValue`, `validateEditing` (`NSControl`).

setAction:

```
- (void)setAction:(SEL)aSelector
```

Sets the action cell's action method to `aSelector`. The argument of an action method sent by an action cell is its associated `NSControl` (the object returned by `controlView`). See also `action`, `setTarget:`, `controlView`, `sendAction:to:` (`NSControl`).

setAlignment:

```
- (void)setAlignment:(NSTextAlignment)mode
```

If the action cell is a text cell (type `NSTextCellType`), this method sets its text alignment mode to `mode`, which should be one of the following enumeration constants:

- `NSLeftTextAlignment`
- `NSRightTextAlignment`
- `NSCenterTextAlignment`
- `NSJustifiedTextAlignment`
- `NSNaturalTextAlignment`

If it's currently in an `NSControl` view, the action cell is redisplayed or marked as needing redisplay. See also `alignment` (`NSCell`).

setBezeled:

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

Adds or removes the action cell's bezel, according to the value of `flag`. Adding a bezel removes the action cell's border, if any. If it's currently in an `NSControl` view, the action cell is redisplayed or marked as needing redisplay. See also `isBezeled` (`NSCell`), `setBordered:`.

setBordered:

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

Adds or removes the action cell's border, according to the value of `flag`. The border is black and has a width of 1.0. Adding a border removes the action cell's bezel, if any. If it's currently in an `NSControl` view, the action cell is redisplayed or marked as needing redisplay. See also `isBordered` (`NSCell`), `setBezeled:`.

setEnabled:

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

Enables or disables the action cell's ability to receive mouse and keyboard events, according to the value of `flag`. If it's currently in an `NSControl` view, the action cell is redisplayed or marked as needing redisplay. See also `isEnabled` (`NSCell`).

setFloatingPointFormat:left:right:

```
- (void)setFloatingPointFormat:(BOOL)autoRange
    left:(unsigned int)leftDigits right:(unsigned int)rightDigits
```

Sets the action cell's floating point format as described in the `NSCell` class specification for the `setFloatingPointFormat:left:right:` method. If it's currently in an `NSControl` view, the action cell is redisplayed or marked as needing redisplay. See also `setFloatingPointFormat:left:right:` (`NSCell`).

setFont:

```
- (void)setFont:(NSFont *)fontObject
```

Sets the action cell's font to `fontObject`. If the action cell is a text cell (type `NSTextCellType`), this method sets its font to `fontObject`. In addition, if it's currently in an `NSControl` view, the action cell is redisplayed or marked as needing redisplay. See also `font` (`NSCell`).

setImage:

```
- (void)setImage:(NSImage *)image
```

Sets the action cell's icon to `image` and sets its `NSCell` type to `NSImageCellType`. If it's currently in an `NSControl` view, the action cell is redisplayed or marked as needing redisplay. See also `setImage:` (`NSCell`).

setStringValue:

```
- (void)setStringValue:(NSString *)aString
```

Sets the action cell's contents to a copy of `aString`. If it's currently in an `NSControl` view, the action cell is redisplayed or marked as needing redisplay. See also `setStringValue:` (`NSCell`), `stringValue`, `doubleValue`, `floatValue`, `intValue`.

setTag:

```
- (void)setTag:(int)anInt
```

Sets the action cell's tag to `anInt`. The tag can be used to identify the action cell in an `NSControl` that contains multiple `NSCell`s (an `NSMatrix`, for example). See also `tag`, `setTag:` (`NSControl`).

setTarget:

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

Sets the action cell's target object to `anObject`. This is the object that is sent the action cell's action method. See also `target`, `setAction:`.

stringValue

```
- (NSString *)stringValue
```

Returns the action cell's contents as an `NSString` object. If the action cell is being edited when this message is received, editing is validated first. See also `setStringValue:`, `stringValue` (`NSCell`), `validateEditing` (`NSControl`), `doubleValue`, `floatValue`, `intValue`.

tag

```
- (int)tag
```

Returns the action cell's tag. The tag can be used to identify the action cell in an `NSControl` that contains multiple `NSCell`s (an `NSMatrix`, for example). See also `setTag:`, `tag` (`NSControl`).

`target`

```
- (id)target
```

Returns the action cell's target object (the object that receives the action cell's action method). See also `setTarget:`, `action`.

NSApplication

Inherits From:	NSResponder : NSObject
Conforms To:	NSCoding (NSResponder), NSObject (NSObject)
Declared In:	AppKit/NSApplication.h AppKit/NSColorPanel.h AppKit/NSDataLinkPanel.h AppKit/NSHelpPanel.h AppKit/NSPageLayout.h

Class Description

The `NSApplication` class provides the central framework for your application's execution. Every application must have exactly one instance of `NSApplication` (or of a custom subclass of `NSApplication`). Your program's `main()` function should create this instance by calling the `sharedApplication` class method. (Alternatively, you could use `alloc` and `init`, making sure they're called only once.) After creating the `NSApplication`, the `main()` function should load your application's main nib file, and then start the event loop by sending the `NSApplication` a `run` message. Here's an example of a typical OpenStep `main()` function in its entirety:

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

Creating the NSApplication object connects the program to the window system and the Display PostScript server, and initializes its PostScript environment. The NSApplication object maintains a list of all the NSWindows that the application uses, so it can retrieve any of the application's NSViews.

The NSApplication object's main task is to receive events from the window system and distribute them to the proper NSResponders. The NSApplication translates an event into an NSEvent object, then forwards the NSEvent to the affected NSWindow object. A key-down event that occurs while the Command key is pressed results in a performKeyEquivalent: message, and every NSWindow has an opportunity to respond to it. Other keyboard and mouse events are sent to the NSWindow associated with the event; the NSWindow then distributes these NSEvents to the objects in its view hierarchy.

In general, it's neater and cleaner to separate the code that embodies your program's functionality into a number of custom objects. Usually those custom objects are subclasses of NSObject. Methods defined in your custom objects can be invoked from a small dispatcher object without being closely tied to the NSApplication object. It's rarely necessary to create a custom subclass of NSApplication. You will need to do so only if you need to provide your own special response to messages that are routinely sent to the NSApplication object. To use a custom subclass of NSApplication, simply substitute it for NSApplication in the main() function above.

When you create an instance of NSApplication (or of a custom subclass of NSApplication), it gets stored as the global variable NSApp. Although this global variable isn't used in the example main() function above, you might find it convenient to refer to NSApp within the source code for your application's custom objects. Note that you can also retrieve the NSApplication object by invoking sharedApplication.

The NSApplication class sets up autorelease pools during initialization and during the event loop—that is, within its init (or sharedApplication) and run methods. Similarly, the methods that the Application Kit adds to NSBundle employ autorelease pools during the loading of nib files. The autorelease pools aren't accessible outside the scope of the respective NSApplication and NSBundle methods. This isn't usually a problem, because a typical OpenStep application instantiates its objects by loading nib files (and by having the objects from the nib file create other objects during initialization and during the event loop). However, if you do need to use OpenStep classes within the main() function itself (other than to invoke the methods just mentioned), you should instantiate an autorelease pool before

using the classes, and then release the pool once you're done. For more information, see the description of the NSAutoreleasePool class in the Foundation Kit section of this book.

The Delegate and Observers

The NSApplication object can be assigned a delegate that responds on behalf of the NSApplication to certain messages addressed to the NSApplication object. Some of these messages, such as application:openFile:withType:, ask the delegate to open a file. Another message, applicationShouldTerminate:, lets the delegate determine whether the application should be allowed to quit.

An NSApplication can also have *observers*. Observers receive notifications of changes in the NSApplication, but they don't have the unique responsibility that a delegate has. Any instance of a class that implements an observer method can register to receive the corresponding notification. For example, if a class implements applicationDidFinishLaunching: and registers to receive the corresponding notification, instances of this class are given an opportunity to react after the NSApplication has been initialized. The observer methods are listed later in this class specification. For information about how to register to receive notifications, see the class specification for the Foundation Kit's NSNotificationCenter class.

There can be only one delegate, but there can be many observers. The delegate itself can be an observer—in fact, in many applications the delegate might be the only observer. Most observers need to explicitly register with an NSNotificationCenter before they can receive a particular notification message, but the delegate only needs to implement the method. By simply implementing an observer method, the NSApplication's delegate is automatically registered to receive the corresponding notification.

Method Types

Activity	Class Method
Creating and initializing the NSApplication	+ sharedApplication – finishLaunching
Changing the active application	– activateIgnoringOtherApps: – deactivate – isActive
Running the event loop	– abortModal – beginModalSessionForWindow: – endModalSession: – isRunning – run – runModalForWindow: – runModalSession: – sendEvent: – stop: – stopModal – stopModalWithCode:
Getting, removing, and posting events	– currentEvent – postEvent:atStart: – discardEventsMatchingMask:beforeEvent: – nextEventMatchingMask:untilDate: inMode:dequeue:
Sending action messages	– sendAction:to:from: – targetForAction: – tryToPerform:with:
Setting the application's icon	– setApplicationIconImage: – applicationIconImage
Hiding all windows	– hide: – isHidden – unhide: – unhideWithoutActivation
Managing windows	– keyWindow – mainWindow – makeWindowsPerform:inOrder: – miniaturizeAll: – preventWindowOrdering – setWindowsNeedUpdate: – updateWindows – windows – windowWithWindowNumber:

Activity	Class Method
Showing standard panels	– orderFrontColorPanel: – orderFrontDataLinkPanel: – orderFrontHelpPanel: – runPageLayout:
Getting the main menu	– mainMenu – setMainMenu:
Managing the Windows menu	– addWindowsItem:title:filename: – arrangeInFront: – changeWindowsItem:title:filename: – removeWindowsItem: – setWindowsMenu: – updateWindowsItem: – windowsMenu
Managing the Services menu	– registerServicesMenuSendTypes:returnTypes: – servicesMenu – setServicesMenu: – servicesProvider – setServicesProvider: – validRequestorForSendType:returnType:
Getting the Display PostScript context	– context
Reporting an exception	– reportException:
Terminating the application	– terminate:
Assigning a delegate	– delegate – setDelegate:
Methods Implemented by the Delegate	– applicationDidBecomeActive: – applicationDidFinishLaunching: – applicationDidHide: – applicationDidResignActive: – applicationDidUnhide: – applicationDidUpdate: – application:openFile: – application:openFileWithoutUI: – application:openTempFile: – applicationOpenUntitledFile: – applicationShouldTerminate: – applicationWillBecomeActive: – applicationWillFinishLaunching: – applicationWillHide: – applicationWillResignActive: – applicationWillTerminate: – applicationWillUnhide: – applicationWillUpdate:

Class Methods

sharedApplication

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

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

Instance Methods

abortModal

```
- (void)abortModal
```

Aborts the modal event loop by raising the exception `NSAbortModalException`, which is caught by `runModalForWindow:`, the method that started the modal loop. Since this method raises an exception, it never returns; `runModalForWindow:`, when stopped with this method, returns the enumeration constant `NSRunAbortedResponse`. Note that you can't use this method to abort modal sessions where you control the modal loop and periodically invoke `runModalSession:`. See also `runModalSession:`, `endModalSession:`, `stopModal`, `stopModalWithCode:`.

activateIgnoringOtherApps:

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

Makes the receiving application the active application. If `flag` is `NO`, the application is activated only if no other application is currently active. Normally, this method is invoked with `flag` set to `NO`. When the Workspace Manager launches an application, it deactivates itself, so `activateIgnoringOtherApps:NO` 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. If `flag` is `YES`, the application will always activate. Regardless of the setting of `flag`, there may be a time lag before the application activates; you should not assume that the application will be active immediately after sending this message.

Note – You can make one of your NSWindows the key window without changing the active application; when you send a makeKeyWindow message to an NSWindow object, you simply ensure that the NSWindow object will be the key window when the application is active.

You should 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. See also deactivate.

addWindowsItem:title:filename:

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

Adds an item to the Windows menu corresponding to the window object aWindow. If isFilename is NO, aString appears literally in the menu. If isFilename is YES, aString is assumed to be a converted name with the name of the file preceding the path (the way NSWindow's setTitleWithRepresentedFilename: method shows a title). If an item for aWindow already exists in the Windows menu, this method has no effect. You rarely invoke this method because an item is placed in the Windows menu for you whenever a window object's title is set. See also changeWindowsItem:title:filename:, setTitleWithRepresentedFilename: (NSWindow).

applicationIconImage

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

Returns the NSImage used for the application's icon. See also setApplicationIconImage:.

arrangeInFront:

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

Arranges all of the windows listed in the Windows menu in front of all other windows. Windows associated with the application but not listed in the Windows menu are not ordered to the front. See also `removeWindowsItem:`, `orderFront:` (`NSWindow`).

beginModalSessionForWindow:

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

Prepares the application for a modal session with `theWindow`. In other words, this method prepares the application so that mouse events get to it only if they occur in `theWindow`. `theWindow` is made the key window and ordered to the front. The return value is a structure that stores information used by the system during a modal session. This structure is allocated by the method and is meant to be used to refer to the session. The application should not access any of the fields of this structure.

The method `beginModalSessionForWindow:` should be balanced by `endModalSession:`. 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. See also `runModalSession:`, `endModalSession:`.

changeWindowsItem:title:filename:

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

Changes the item for `aWindow` in the Windows menu to `aString`. If `aWindow` doesn't have an item in the Windows menu, this method adds the item. If `isFilename` is `NO`, `aString` appears literally in the menu. If `isFilename` is `YES`, `aString` is assumed to be a converted name with the file's name preceding the path (the way `NSWindow`'s `setTitleWithRepresentedFilename:` places a title). See also `addWindowsItem:title:filename:`, `setTitleWithRepresentedFilename:` (`NSWindow`).

context

```
- (NSDPSContext *)context
```

Returns the NSApplication's Display PostScript context. See also NSDPSContext.

currentEvent

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

Returns a pointer to the last event the NSApplication object retrieved from the event queue. A pointer to the current event is also passed with every event message.

deactivate

```
- (void)deactivate
```

Deactivates the application if it's active. Normally, you shouldn't invoke this method; the Application Kit is responsible for proper deactivation. See also activateIgnoringOtherApps:.

delegate

```
- (id)delegate
```

Returns the NSApplication's delegate. See also setDelegate:.

discardEventsMatchingMask:beforeEvent:

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

Removes from the event queue all events matching mask that were generated before lastEvent. If lastEvent is nil, all events matching mask are removed from the queue.

endModalSession:

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

Finishes and cleans up after a modal session. The argument session should be taken from a previous invocation of beginModalSession:for:. See also runModalSession:, beginModalSessionForWindow:.

finishLaunching

```
- (void)finishLaunching
```

Activates the application, opens any files specified by the "NSOpen" user default, and unhighlights the application's icon in the Workspace Manager. This method is invoked by `run` before it starts the event loop. When this method begins, it posts the notification `NSApplicationWillFinishLaunchingNotification` with the receiving object to the default notification center. When it successfully completes, it posts the notification `NSApplicationDidFinishLaunchingNotification`. If you override `finishLaunching`, the subclass method should invoke the superclass method.

hide:

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

Collapses the application's graphics—including all its windows, menus, and panels—into a single small window. The `hide:` message is usually sent using the Hide command in the application's main Menu. When this method begins, it posts the notification `NSApplicationWillHideNotification` with the receiving object to the default notification center. When it completes successfully, it posts the notification `NSApplicationDidHideNotification`. See also `unhide:`.

isActive

```
- (BOOL)isActive
```

Returns `YES` if the application is currently active, and `NO` if it isn't. See also `activateIgnoringOtherApps:`.

isHidden

```
- (BOOL)isHidden
```

Returns `YES` if the application is currently hidden, and `NO` if it isn't.

isRunning

```
- (BOOL)isRunning
```

Returns YES if the application is running, and NO if the stop: method has ended the main event loop. See also run, stop:, terminate:.

keyWindow

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

Returns the key NSWindow, that is, the NSWindow that receives keyboard events. If there is no key NSWindow, or if the key NSWindow belongs to another application, this method returns nil. See also mainWindow, isKeyWindow (NSWindow).

mainMenu

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

Returns the id of the application's main menu. See also NSMenu.

mainWindow

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

Returns the application's main window. See also NSWindow.

makeWindowsPerform:inOrder:

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

Sends the application object's NSWindows a message to perform the aSelector method. The message is sent to each NSWindow in turn until one of them returns YES; the method then returns a pointer to that window. If no NSWindow returns YES, the method returns nil. If flag is YES, the application object's NSWindows receive the aSelector messages in the front-to-back order in which they appear in the Window Server's window list. If flag is NO, the NSWindows receive the messages in the order they appear in the application object's window list. This order generally reflects the order in which the NSWindows were created. The method designated by aSelector can't take any arguments.

miniaturizeAll:

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

Miniaturizes all the receiver's application windows.

nextEventMatchingMask:untilDate: inMode:dequeue:

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

Returns the next event matching `mask`, or `nil` if no such event is found before the `expiration` date. If `flag` is `YES`, the event is removed from the queue. The `mode` argument names an `NSRunLoop` mode that determines what other ports are listened to and what timers may fire while the application is waiting for the event.

orderFrontColorPanel:

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

Brings up the color panel.

orderFrontDataLinkPanel:

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

Shows the shared instance of the data link panel, creating it first if necessary. Note that this method is not part of the OpenStep specification.

orderFrontHelpPanel:

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

Shows the application's help panel or the default help panel. Note that this method is not part of the OpenStep specification.

postEvent:atStart:

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

Adds `event` to the front of the application's event queue if `flag` is `YES`, or to the back of the queue otherwise.

preventWindowOrdering

```
- (void)preventWindowOrdering
```

Suppresses the usual window ordering in handling the most recent mouse-down event. Most applications will not need to use this method since the Application Kit support for dragging will call this method when dragging is initiated.

registerServicesMenuSendTypes:returnTypes:

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

Registers pasteboard types that the application can send and receive in response to service requests. If the application has a Services menu, a menu item is added for each service provider that can accept one of the specified send types or return one of the specified return types. This method should typically be invoked at application startup time or when an object that can use services is created. It can be invoked more than once; its purpose is to ensure that there is a menu item for every service that the application may use. The individual items will be dynamically enabled and disabled by the event handling mechanism to indicate which services are currently appropriate. An application (or object instance that can cut or paste) should register every possible type that it can send and receive. See also `validRequestorForSendType:returnType:` (`NSResponder`), `readSelectionFromPasteboard:` (`NSCStringText`), `writeSelectionToPasteboard:types:` (`NSCStringText`), `NSPasteboard`.

removeWindowsItem:

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

Removes the item for `aWindow` in the Windows menu. Note that this method doesn't prevent the item from being automatically added again, so you must use `NSWindow`'s `setExcludedFromWindowsMenu:` method if you want the item to remain excluded from the Windows menu. See also `changeWindowsItem:title:filename:`, `setExcludedFromWindowsMenu:` (`NSWindow`).

reportException:

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

Logs the given exception by calling `NSLog()`(Foundation Kit Functions).

run

```
- (void)run
```

Initiates the application object's main event loop. The loop continues until a `stop:` or `terminate:` message is received. Each iteration through the loop, the next available event from the Window Server is stored, and is then dispatched by sending the event to the application object using `sendEvent:`. A `run` message should be sent as the last statement from `main()`, after the application's objects have been initialized. This method returns if it is terminated by `stop:`, but never returns if it is terminated by `terminate:`. See also `runModalForWindow:`, `sendEvent:`, `stop:`, `terminate:`.

runModalForWindow:

```
- (int)runModalForWindow:(NSWindow *)theWindow
```

Establishes a modal event loop for `theWindow`. Until the loop is broken by a `stopModal`, `stopModalWithCode:`, or `abortModal` message, the application won't respond to any mouse, keyboard, or window-close events unless they're associated with `theWindow`. If `stopModalWithCode:` is used to stop the modal event loop, this method returns the argument passed to `stopModalWithCode:`. If `stopModal` is used, it returns the constant `NSRunStoppedResponse`. If `abortModal` is used, it returns the constant `NSRunAbortedResponse`. This method is functionally similar to the following code:

```
NSWindow *theWindow;
NSModalSession session;

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

See also stopModal, stopModalWithCode:, abortModal, runModalSession:.

runModalSession:

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

Runs a modal session represented by session, as defined in a previous invocation of beginModalSessionForWindow:. A loop using this method is similar to a modal event loop run with runModalForWindow: except that the application can continue processing between method invocations. When you invoke this method, events for the NSWindow of this session are dispatched as normal. This method returns when there are no more events. You must invoke this method frequently enough that the window remains responsive to events.

If the modal session was not stopped, this method returns NSRunContinuesResponse. If stopModal was invoked as the result of event processing, NSRunStoppedResponse is returned. If stopModalWithCode: was invoked, this method returns the value passed to stopModalWithCode:. The NSAbortModalException exception raised by abortModal isn't caught. See also beginModalSessionForWindow:, endModalSession:, stopModal, stopModalWithCode:.

runPageLayout:

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

Brings up the application object's Page Layout panel, which allows the user to select the page size and orientation.

sendAction:to:from:

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

Sends an action message to the object aTarget. If aTarget is nil, the application object looks for an object that can respond to the message—that is, for an object that implements a method matching aSelector. 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 NSResponder chain. If none of the objects in the key window's responder chain can handle the message, the application object attempts to send the message to the key NSWindow'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, the NSApplication object 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. This method returns YES if the action is applied; otherwise it returns NO.

sendEvent:

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

Sends an event to the application object. You rarely send sendEvent: messages directly although you might want to override this method to perform some action on every event. The sendEvent: messages are sent from the main event loop (the run method). This method dispatches events to the appropriate responders: the application object handles application events; the NSWindow indicated in the event record handles window related events; and mouse and key events are forwarded to the appropriate NSWindow for further dispatching.

When sending the activate application event, this method posts the notifications NSApplicationWillBecomeActive and NSApplicationDidBecomeActive with the receiving object to the default notification center. When sending the deactivate application event, it posts the NSApplicationWillResignActiveNotification and NSApplicationDidResignActiveNotification notifications with the receiving object to the default notification center.

servicesMenu

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

Returns the application object's Services menu. Returns nil if no Services menu has been created. See also setServicesMenu:.

servicesProvider

```
- (id)servicesProvider
```

Returns the application's services provider application. The services provider application responds to remote messages sent from the Services menus of other applications. The services provider application should contain methods that a service-providing application uses to give services to other applications. See also `setServicesProvider:`, `NSRegisterServicesProvider()`.

setApplicationIconImage:

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

Sets the application's icon to `anImage`. See also `applicationIconImage`.

setDelegate:

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

Makes `anObject` the application's delegate. The notification messages that a delegate can expect to receive are listed under "Methods Implemented by the Delegate" on page -34. The delegate doesn't need to implement all the methods. See also `delegate`.

setMainMenu:

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

Makes `aMenu` the application's main menu. See also `mainMenu`.

setServicesMenu:

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

Makes `aMenu` the application object's Services menu. See also `servicesMenu`.

setServicesProvider:

```
- (id)setServicesProvider:(id)provider
```

Registers the service provider application that will respond to remote messages. Applications registered with this method should create an `NSApplication` object. See also `servicesProvider`, `NSRegisterServicesProvider()`.

setWindowsMenu:

```
- (void)setWindowsMenu:(id)aMenu
```

Makes `aMenu` the application object's Windows menu. See also `windowsMenu`.

setWindowsNeedUpdate:

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

Sets whether the application's windows need updating when the application has finished processing the current event. This method is especially useful for making sure menus are updated to reflect changes not initiated by user actions.

stop:

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

Stops the main event loop. This method will break the flow of control out of the `run` method, thereby returning to the `main()` function. A subsequent `run` message will restart the loop. If this method is applied during a modal event loop, it will break that loop but not the main event loop. See also `terminate:`, `run`, `runModalSession:`.

stopModal

```
- (void)stopModal
```

Stops a modal event loop. This method should always be paired with a previous `runModalForWindow:` or `beginModalSessionForWindow:` message. When `runModalForWindow:` is stopped with this method, it returns `NSRunStoppedResponse`. This method will stop the loop only if it's executed by code responding to an event. See also `stopModalWithCode:`, `runModalSession:`, `abortModal`.

stopModalWithCode:

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

This method is similar to `stopModal` except that the argument `returnCode` allows you to specify the value that `runModalForWindow:` will return. See also `stopModal`, `abortModal`.

targetForAction:

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

Returns the object that receives the action message `aSelector`.

terminate:

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

Frees the application object and exits the application. This is the default action method for the application's Quit menu item. Each use of `terminate:` invokes `applicationShouldTerminate:` to notify the delegate that the application is about to terminate. If `applicationShouldTerminate:` returns `NO`, control is returned to the main event loop, and the application isn't terminated. Otherwise, this method frees the application object and terminates the application.

Note – You should not put final cleanup code in your application's `main()` function; it will never be executed.

See also `stop:`, `applicationShouldTerminate:` (delegate method).

tryToPerform:with:

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

Aids in dispatching action messages. The application object tries to perform the method `aSelector` using its inherited `NSResponder` method `tryToPerform:with:`. If the application object doesn't perform `aSelector`, the object's delegate is given the opportunity to perform it using its inherited `NSObject` method `performSelector:object:afterDelay:`. If either the application object or the application object's delegate accept `aSelector`, this method returns `YES`; otherwise it returns `NO`. See also `tryToPerform:with:` (`NSResponder`), `instancesRespondToSelector:` (`NSObject`), `performSelector:object:afterDelay:` (`NSObject`).

unhide:

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

Restores a hidden application to its former state (all of the windows, menus, and panels visible), and makes it the active application. This method is usually invoked as the result of double-clicking the icon for the hidden application. See also `hide:`, `unhideWithoutActivation`, `activateIgnoringOtherApps:`.

unhideWithoutActivation

```
- (void)unhideWithoutActivation
```

Unhides the application but doesn't make it the active application. You might want to invoke `activateIgnoringOtherApps:NO` after invoking this method to make the receiving application active if there is no active application. When this method begins, it posts the notification `NSApplicationWillUnhideNotification` with the receiving object to the default notification center. When it completes successfully, it posts the notification `NSApplicationDidUnhideNotification`. See also `hide:`, `activateIgnoringOtherApps:`.

updateWindows

```
- (void)updateWindows
```

Sends an `update` message to on-screen `NSWindow`s. When this method begins, it sends the notification `NSApplicationWillUpdateNotification` with the receiving object to the default notification center. When it successfully completes, it sends the notification `NSApplicationDidUpdateNotification`. If the delegate implements `applicationWillUpdate:`, that message is sent to the delegate before the windows are updated. Similarly, if the delegate implements `applicationDidUpdate:`, that message is sent to the delegate after the windows are updated. See also `applicationWillUpdate:` (delegate method), `applicationDidUpdate:` (delegate method).

updateWindowsItem:

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

Updates the item for aWindow in the Windows menu to reflect the edited status of aWindow. You rarely need to invoke this method because it is invoked automatically when the edited status of an NSWindow is set. See also changeWindowsItem:title:filename:, setDocumentEdited: (NSWindow).

validRequestorForSendType:returnType:

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

Indicates whether the application object can send and receive the specified types. This message is passed on to the application object's delegate if the delegate can respond (and isn't an NSResponder with its own next responder). If the delegate can't respond or returns nil, this method returns nil, indicating that no object was found that could supply sendType data for a remote message from the Services menu and accept back returnType data. If such an object was found, it is returned. Messages to perform this method are initiated by the Services menu. See also validRequestorForSendType:returnType: (NSResponder), registerServicesMenuSendTypes:returnTypes:, writeSelectionToPasteboard:types: (NSCStringText), readSelectionFromPasteboard: (NSCStringText).

windows

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

Returns a pointer to the NSArray object used to keep track of all the application object's NSWindows, including menus, panels, and the like. In the current implementation, this array also contains global (shared) NSWindows.

windowsMenu

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

Returns the application object's Windows menu. Returns nil if no Windows menu has been created.

windowWithWindowNumber:

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

Returns the NSWindow object corresponding to windowNum.

Methods Implemented by the Delegate

applicationDidBecomeActive:

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

Sent by the default notification center to the delegate; aNotification is always NSApplicationDidBecomeActiveNotification. If the delegate implements this method, it's automatically registered to receive the notification. See also applicationDidFinishLaunching: (delegate method).

applicationDidFinishLaunching:

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

Sent by the default notification center to the delegate; aNotification is always NSApplicationDidFinishLaunchingNotification. If the delegate implements this method, it's automatically registered to receive the notification. See also applicationDidBecomeActive: (delegate method).

applicationDidHide:

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

Sent by the default notification center to the delegate; aNotification is always NSApplicationDidHideNotification. If the delegate implements this method, it's automatically registered to receive the notification. See also hide:, applicationDidUnhide: (delegate method).

applicationDidResignActive:

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

Sent by the default notification center to the delegate immediately after the application is deactivated; `aNotification` is always `NSApplicationDidResignActiveNotification`. If the delegate implements this method, it's automatically registered to receive the notification.

applicationDidUnhide:

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

Sent by the default notification center to the delegate immediately after the application is unhidden; `aNotification` is always `NSApplicationDidUnhideNotification`. If the delegate implements this method, it's automatically registered to receive the notification. See also `hide:`, `applicationDidHide:` (delegate method).

applicationDidUpdate:

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

Sent by the default notification center to the delegate immediately after the application object updates its `NSWindows`.; `aNotification` is always `NSApplicationDidUpdateNotification`. If the delegate implements this method, it's automatically registered to receive the notification. See also `updateWindows`, `updateWindowsItem:`, `applicationWillUpdate:` (delegate method).

application:openFile:

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

Sent directly by `application` to the delegate. This method is like `application:openFileWithoutUI:`, but brings up the user interface of the file's application. The method returns `YES` if it is able to open the file, and returns `NO` otherwise. See also `application:openFileWithoutUI:` (delegate method), `application:openTempFile:` (delegate method).

application:openFileWithoutUI:

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

Sent directly by `sender` to the delegate. Opens the specified file to run without a user interface. Work with the file will be under programmatic control of `sender`, rather than under keyboard control of the user. Returns `YES` or `NO` to indicate whether the file was successfully opened. See also `application:openFile:` (delegate method).

application:openTempFile:

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

Sent directly by `application` to the delegate. This method is like `application:openFile:`, except that 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. This method returns `YES` if it is able to open the file, and `NO` otherwise. See also `application:openFile:` (delegate method).

applicationOpenUntitledFile:

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

Sent directly by `application` to the delegate. This method is like `application:openFile:`, but it opens a new, untitled document.

applicationShouldTerminate:

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

Sent directly by `sender` to the delegate. Returns `YES` if the application should terminate.

applicationWillBecomeActive:

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

Sent by the default notification center to the delegate; `aNotification` is always `NSApplicationWillBecomeActiveNotification`. If the delegate implements this method, it's automatically registered to receive this notification.

applicationWillFinishLaunching:

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

Sent by the default notification center to the delegate; `aNotification` is always `NSApplicationWillFinishLaunchingNotification`. If the delegate implements this method, it's automatically registered to receive this notification.

applicationWillHide:

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

Sent by the default notification center to the delegate; `aNotification` is always `NSApplicationWillHideNotification`. If the delegate implements this method, it's automatically registered to receive this notification.

applicationWillResignActive:

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

Sent by the default notification center to the delegate to indicate that the application is about to give up its active status; `aNotification` is always `NSApplicationWillResignActiveNotification`. If the delegate implements this method, it's automatically registered to receive this notification.

applicationWillTerminate:

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

Sent by the default notification center to the delegate to indicate that the application is about to terminate; `aNotification` is always `NSApplicationWillTerminateNotification`. If the delegate implements this method, it's automatically registered to receive the notification.

applicationWillUnhide:

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

Sent by the default notification center to the delegate to indicate that the application is about to unhide any hidden windows; `aNotification` is always `NSApplicationWillUnhideNotification`. If the delegate implements this method, it's automatically registered to receive the notification.

applicationWillUpdate:

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

Sent by the default notification center to the delegate immediately before the application object updates its `NSWindows`; `aNotification` is always `NSApplicationWillUpdateNotification`. If the delegate implements this method, it's automatically registered to receive this notification. See also `updateWindows`, `updateWindowsItem:`, `applicationDidUpdate:` (delegate method).

NSBitmapImageRep

Inherits From:	NSImageRep : NSObject
Conforms To:	NSCoding, NSCopying (NSImageRep), NSObject (NSObject)
Declared In:	AppKit/NSBitmapImageRep.h

Class Description

An `NSBitmapImageRep` is an object that can render an image from bitmap data. The data can be in Tag Image File Format (TIFF), or it can be raw image data. If it's raw data, the object must be informed about the structure of the image when it's first initialized—its size, the number of color components, the number of bits per sample, and so on. If it's TIFF data, the object can get this information from the various TIFF fields included with the data.

Although `NSBitmapImageRep`s are often used indirectly, through instances of the `NSImage` class, they can also be used directly—for example, to manipulate the bits of an image as you might need to do in a paint program.

Setting Up an NSBitmapImageRep

A new `NSBitmapImageRep` is passed bitmap data for an image when it's first initialized. An `NSBitmapImageRep` can also be created from bitmap data that's read from a specified rectangle of a focused `NSView`.

Although the `NSBitmapImageRep` class inherits `NSImageRep` methods that set image attributes, these methods shouldn't be used. Instead, you should either allow the object to find out about the image from the TIFF fields or use methods defined in this class to supply this information when the object is initialized.

TIFF Compression

TIFF data can be read and rendered after it has been compressed using any one of the four schemes briefly described below:

Table 1-1 TIFF Compression Schemes

Scheme	What It Does
LZW	Compresses and decompresses without information loss, achieving compression ratios up to 5:1. It may be somewhat slower to compress and decompress than the PackBits scheme.
PackBits	Compresses and decompresses without information loss, but may not achieve the same compression ratios as LZW.
JPEG	Compresses and decompresses with some information loss, but can achieve compression ratios anywhere from 10:1 to 100:1. The ratio is determined by a user-settable factor ranging from 1.0 to 255.0, with higher factors yielding greater compression. More information is lost with greater compression, but 15:1 compression is safe for publication quality. Some images can be compressed even more. JPEG compression can be used only for images that specify at least 4 bits per sample.
CCITTFAX	Compresses and decompresses 1 bit grayscale images using international fax compression standards CCITT3 and CCITT4.

An `NSBitmapImageRep` can also produce compressed TIFF data for its image using any of these schemes.

Method Types

Activity	Class Method
Allocating and initializing a new NSBitmapImageRep object	+ imageRepWithData: + imageRepsWithData: – initWithData: – initWithFocusedViewRect: – initWithBitmapDataPlanes:pixelsWide: pixelsHigh:bitsPerSample:samplesPerPixel: hasAlpha:isPlanar:colorSpaceName:bytesPerRow: bitsPerPixel:
Getting information about the image	– bitmapData – getBitmapDataPlanes:
Producing a TIFF representation of the image	+ TIFFRepresentationOfImageRepsInArray: + TIFFRepresentationOfImageRepsInArray: usingCompression:factor: – TIFFRepresentation – TIFFRepresentationUsingCompression: factor:
Setting and checking compression types	+ getTIFFCompressionTypes:count: + localizedNameForTIFFCompressionType: – canBeCompressedUsing: – getCompression:factor: – setCompression:factor:

Class Methods

getTIFFCompressionTypes:count:

```
+ (void)getTIFFCompressionTypes:(const NSTIFFCompression **)list
    count:(int *)numTypes
```

Returns in `list`, by reference, an array of enumeration constants representing all available compression types that can be used when writing a TIFF image. The number of elements in `list` is represented by `numTypes`. `list` belongs to the `NSBitmapImageRep` class; it shouldn't be freed or altered. The following compression types are supported:

Table 1-2 TIFF Data Compression Schemes

Constant	Value	Usage
NSTIFFCompressionNone	1	
NSTIFFCompressionCCITTFAX3	3	1 bps images
NSTIFFCompressionCCITTFAX4	4	1 bps images
NSTIFFCompressionLZW	5	
NSTIFFCompressionJPEG	6	
NSTIFFCompressionNEXT	32766	Input only
NSTIFFCompressionPackBits	32773	
NSTIFFCompressionOldJPEG	32865	Input only

Note that not all compression types can be used for all images: `NSTIFFCompressionNEXT` can be used only to retrieve image data. Because future releases of OpenStep may include other compression types, always use this method to get the available compression types—for example, when you implement a user interface for selecting compression types. See also `localizedNameForTIFFCompressionType:`, `canBeCompressedUsing:`.

imageRepWithData:

```
+ (id)imageRepWithData:(NSData *)tiffData
```

Creates and returns an initialized `NSBitmapImageRep` corresponding to the first image in `tiffData`.

imageRepsWithData:

```
+ (NSArray *)imageRepsWithData:(NSData *)tiffData
```

Creates and returns initialized `NSBitmapImageRep` objects for all the images in `tiffData`.

localizedNameForTIFFCompressionType:

```
+ (NSString *)localizedNameForTIFFCompressionType:
    (NSTIFFCompression)compression
```

Returns a string containing the localized name for the compression type represented by `compression` or, returns `NULL` if `compression` is unrecognized. The possible compression types are listed in the `getTIFFCompressionTypes:count:` class method description. When implementing a user interface for selecting TIFF compression types, use the `getTIFFCompressionTypes:count:` method to get the list of supported compression types; then use this method to get the localized names for each compression type. The returned string belongs to the `NSBitmapImageRep` class; don't attempt to alter or free it. See also `getTIFFCompressionTypes:count:`.

TIFFRepresentationOfImageRepsInArray:

```
+ (NSData *)TIFFRepresentationOfImageRepsInArray:(NSArray *)anArray
```

Returns a TIFF representation of the images in the specified `NSArray`, using the compression that's returned by `getCompression:factor:` (if applicable).

TIFFRepresentationOfImageRepsInArray: usingCompression:factor:

```
+ (NSData *)TIFFRepresentationOfImageRepsInArray:(NSArray *)anArray
    usingCompression:(NSTIFFCompression)compressionType
    factor:(float)factor
```

Returns a TIFF representation of the images in the specified `NSArray`, which are compressed using `compressionType` and `factor`. If the specified compression isn't applicable, no compression is used.

Instance Methods

bitmapData

```
- (unsigned char *)bitmapData
```

Returns a pointer to the bitmap data. If the data is planar, returns a pointer to the first plane.

bitsPerPixel

```
- (int)bitsPerPixel
```

Returns the number of bits allocated for each pixel in each plane of data. This is normally equal to the number of bits per sample or, if the data is in meshed configuration, the number of bits per sample times the number of samples per pixel. It can be explicitly set to another value (in the `initWithBitmapDataPlanes:...` method) in case extra memory is allocated for each pixel. This may be the case, for example, if pixel data is aligned on byte boundaries.

bytesPerPlane

```
- (int)bytesPerPlane
```

Returns the number of bytes in each plane or channel of data. This will be figured from the number of bytes per row and the height of the image. See also `bytesPerRow`.

bytesPerRow

```
- (int)bytesPerRow
```

Returns the minimum number of bytes required to specify a scan line (a single row of pixels spanning the width of the image) in each data plane. If not explicitly set to another value (in the `initWithBitmapDataPlanes:...` method), this will be figured from the width of the image, the number of bits per sample, and, if the data is in a meshed configuration, the number of samples per pixel. It can be set to another value to indicate that each row of data is aligned on word or other boundaries.

canBeCompressedUsing:

```
- (BOOL)canBeCompressedUsing:(NSTIFFCompression)compression
```

This method tests whether the receiver can be compressed by `compression` type. For a list of the possible compression types, see Table 1-2 on page 41. This method returns YES if the receiver's data matches `compression`; for example,

if compression is NSTIFFCompressionCCITTFAX3, then the data must be one bit-per-sample and one sample-per-pixel. This method returns NO if the data doesn't match compression or if compression is unsupported. See also getTIFFCompressionTypes:count:.

getBitmapDataPlanes:

```
- (void)getBitmapDataPlanes:(unsigned char **)data
```

Provides pointers to each plane of bitmap data. data should be an array of five character pointers. If the bitmap data is in planar configuration, each pointer will be initialized to point to one of the data planes. If there are less than five planes, the remaining pointers will be set to NULL. If the bitmap data is in meshed configuration, only the first pointer will be initialized; the others will be NULL. Color components in planar configuration are arranged in the expected order—for example, red before green before blue for RGB color. All color planes precede the coverage plane. See also isPlanar.

getCompression:factor:

```
- (void)getCompression:(NSTIFFCompression *)compression
    factor:(float *)factor
```

Returns by reference the receiver's compression type and compression factor. Use this method to get information on the compression type for the source image data. compression represents the compression type used on the data, and corresponds to one of the values returned by the class method getTIFFCompressionTypes:count:. factor is usually a value between 0.0 and 255.0, with 0.0 representing no compression. See also getTIFFCompressionTypes:count:, setCompression:factor:.

initWithBitmapDataPlanes:pixelsWide: pixelsHigh:bitsPerSample:samplesPerPixel: hasAlpha:isPlanar:colorSpaceName:bytesPerRow: bitsPerPixel:

```
- (id)initWithBitmapDataPlanes:(unsigned char **)planes
    pixelsWide:(int)width pixelsHigh:(int)height
    bitsPerSample:(int)bps samplesPerPixel:(int)spp
```

```
hasAlpha:(BOOL)alpha isPlanar:(BOOL)config
colorSpaceName:(NSString *)colorSpaceName
bytesPerRow:(int)rowBytes bitsPerPixel:(int)pixelBits
```

Initializes the receiver, a newly allocated NSBitmapImageRep object, so that it can render the image specified in `planes` and described by the other arguments. If the object can't be initialized, this method frees it and returns `nil`. Otherwise, it returns the object (`self`).

`planes` is an array of character pointers, each of which points to a buffer containing raw image data. If the data is in planar configuration, each buffer holds one component—one plane—of the data. Color planes are arranged in the standard order—for example, red before green before blue for RGB color. All color planes precede the coverage plane. If the data is in meshed configuration (`config` is `NO`), only the first buffer is read.

If `planes` is `NULL` or if it's an array of `NULL` pointers, this method allocates enough memory to hold the image described by the other arguments. You can then obtain pointers to this memory (with the `getBitmapDataPlanes:` method) and fill in the image data. In this case, the allocated memory will belong to the object and will be freed when it's freed.

If `planes` is not `NULL` and the array contains at least one data pointer, the object will only reference the image data; it won't copy it. The buffers won't be freed when the object is freed.

Each of the other arguments (besides `planes`) informs the `NSBitmapImageRep` object about the image. They're explained below:

- `width` and `height` specify the size of the image in pixels. The size in each direction must be greater than 0.
- `bps` (bits per sample) is the number of bits used to specify one pixel in a single component of the data. All components are assumed to have the same bits per sample.
- `spp` (samples per pixel) is the number of data components. It includes both color components and the coverage component (alpha), if present. Meaningful values range from 1 through 5. An image with cyan, magenta, yellow, and black (CMYK) color components plus a coverage component would have an `spp` of 5; a gray-scale image that lacks a coverage component would have an `spp` of 1.

- `alpha` should be `YES` if one of the components counted in the number of samples per pixel (`spp`) is a coverage component, and `NO` if there is no coverage component.
- `config` should be `YES` if the data components are laid out in a series of separate "planes" or channels ("planar configuration"), and `NO` if component values are interwoven in a single channel ("meshed configuration"). For example, in meshed configuration, the red, green, blue, and coverage values for the first pixel of an image would precede the red, green, blue, and coverage values for the second pixel, and so on. In planar configuration, red values for all the pixels in the image would precede all green values, which would precede all blue values, which would precede all coverage values.
- `space` indicates how data values are to be interpreted. It should be one of the following `NSString`s:

Table 1-3 Color Space Names

NSString
NSCalibratedWhiteColorSpace
NSCalibratedBlackColorSpace
NSCalibratedRGBColorSpace
NSDeviceWhiteColorSpace
NSDeviceBlackColorSpace
NSDeviceRGBColorSpace
NSDeviceCMYKColorSpace
NSNamedColorSpace
NSCustomColorSpace

- `rowBytes` is the number of bytes that are allocated for each scan line in each plane of data. A scan line is a single row of pixels spanning the width of the image. Normally, `rowBytes` can be figured from the width of the image, the number of bits per pixel in each sample (`bps`), and, if the data is in a meshed configuration, the number of samples per pixel (`spp`). However, if the data for each row is aligned on word or other boundaries, it may have been necessary to allocate more memory for each row than there is data to

fill it. rowBytes lets the object know whether that's the case. If rowBytes is 0, the NSBitmapImageRep assumes that there's no empty space at the end of a row.

- pixelBits informs the NSBitmapImageRep how many bits are actually allocated per pixel in each plane of data. If the data is in planar configuration, this normally equals bps (bits per sample). If the data is in meshed configuration, it normally equals bps times spp (samples per pixel). However, it's possible for a pixel specification to be followed by some meaningless bits (empty space), as may happen, for example, if pixel data is aligned on byte boundaries. Currently, an NSBitmapImageRep cannot render an image if this is the case. If pixelBits is 0, the object will interpret the number of bits per pixel to be the expected value, without any meaningless bits.

This method is the designated initializer for NSBitmapImageReps that handle raw image data.

initWithData:

```
- (id)initWithData:(NSData *)tiffData
```

Initializes a newly allocated NSBitmapImageRep from the first TIFF header and image data found in tiffData.

initWithFocusedViewRect:

```
- (id)initWithFocusedViewRect:(NSRect)rect
```

Initializes the new object using data read from the image contained in the rectangle rect.

isPlanar

```
- (BOOL)isPlanar
```

Returns YES if image data is segregated into a separate plane for each color and coverage component (planar configuration), and NO if the data is integrated into a single plane (meshed configuration). See also samplesPerPixel.

numberOfPlanes

```
- (int)numberOfPlanes
```

Returns the number of separate planes that image data is organized into. This will be the number of samples per pixel if the data has a separate plane for each component (`isPlanar` returns `YES`) and 1 if the data is meshed (`isPlanar` returns `NO`). See also `isPlanar`, `samplesPerPixel`, `hasAlpha` (`NSImageRep`).

samplesPerPixel

```
- (int)samplesPerPixel
```

Returns the number of components in the data. It includes both color components and the coverage component, if present. See also `hasAlpha` (`NSImageRep`).

setCompression:factor:

```
- (void)setCompression:(NSTIFFCompression)compression
    factor:(float)factor
```

Sets the receiver's compression type and compression factor. `compression` is one of the supported compression types listed in the `getTiffCompressionTypes:count:` class method description. `factor` is a compression factor, usually between 0.0 (no compression) and 255.0 (maximum compression). When an `NSBitmapImageRep` is created, the instance stores the compression type and factor for the source data. If you subsequently request a TIFF representation of the image using `TIFFRepresentation`, this method tries to use the stored compression type and factor. Use `setCompression:factor:` to change the compression type and factor. See also `getTIFFCompressionTypes:count:`, `getCompression:factor:`, `TIFFRepresentation`.

TIFFRepresentation

```
- (NSData *)TIFFRepresentation
```

Returns a TIFF representation of the image, using the compression type and factor returned by `getCompression:factor:` (if applicable).

TIFFRepresentationUsingCompression: factor:

```
- (NSData *)TIFFRepresentationUsingCompression:
    (NSTIFFCompression)compressionType factor:(float)factor
```

Returns a compressed TIFF representation of the image, having the specified compression type and compression factor. If the specified compression isn't applicable, no compression is used. Raises `NSTIFFException` if an attempt is made to create a TIFF representation using OpenStep custom color space bitmaps.

NSBox

Inherits From:	NSView : NSResponder : NSObject
Conforms To:	NSCoding (NSResponder), NSObject (NSObject)
Declared IN:	AppKit / NSBox.h

Class Description

An `NSBox` object is a simple `NSView` that can do two things: It can draw a border around itself and it can title itself. You can use an `NSBox` to group, visually, some number of other `NSView`s. These other `NSView`s are added to the `NSBox` through the typical subview-adding methods, such as `addSubview:` and `replaceSubview:with:`.

An `NSBox` contains a *content area*, a rectangle set within the `NSBox`'s frame in which the `NSBox`'s subviews are displayed. The size and location of the content area depends on the `NSBox`'s border type, title location, the size of the font used to draw the title, and an additional measure that you can set through the `setContentViewMargins:` method. When you create an `NSBox`, an instance of `NSView` is created and added (as a subview of the `NSBox` object) to fill the `NSBox`'s content area. If you replace this *content view* with an `NSView` of your own, your `NSView` will be resized to fit the content area. Similarly, as you resize an `NSBox` its content view is automatically resized to fill the content area.

The NSViews that you add as subviews to an NSBox are actually added to the NSBox's content view—NSView's subview-adding methods are redefined by NSBox to ensure that a subview is correctly placed in the view hierarchy. However, you should note that the subviews method *isn't* redefined: it returns an NSArray containing a single object, the NSBox's content view.

Method Types

Activity	Class Mehthod
Getting and modifying the border and title	– borderRect – borderType – setBorderType: – setTitle: – setTitleFont: – setTitlePosition: – title – titleCell – titleFont – titlePosition – titleRect
Setting and placing the content view	– contentView – contentViewMargins – setContentView: – setContentViewMargins::
Resizing the box	– setFrameFromContentFrame:: – sizeToFit

Instance Methods

borderRect

```
– (NSRect)borderRect
```

Returns the rectangle in which the border is drawn.

borderType

```
– (NSBorderType)borderType
```

Returns the box's border type, one of NSNoBorder, NSLineBorder, NSBezelBorder, or NSGrooveBorder. By default, a box's border type is NSGrooveBorder. See also setBorderType:.

contentView

```
- (id)contentView
```

Returns the box's content view. The content view is created automatically when the box is created, and resized as the box is resized (you should never send frame-altering messages directly to a box's content view). You can replace it with an NSView of your own through the setContentView: method. See also setContentView:.

contentViewMargins

```
- (NSSize)contentViewMargins
```

Returns the distances between the border and the content view. See also setContentViewMargins:.

setBorderType:

```
- (void)setBorderType:(NSBorderType)aType
```

Sets the box's border type to aType, which must be NSNoBorder, NSLineBorder, NSBezelBorder, or NSGrooveBorder. By default, a box's border type is NSGrooveBorder. If the size of the new border is different from that of the old border, the content view is resized to absorb the difference. The box isn't redisplayed. See also borderType.

setContentView:

```
- (void)setContentView:(NSView *)aView
```

Replaces the box's content view with aView, resizing the view to fit within the box's current content area. See also contentView.

setContentViewMargins:

```
- (void)setContentViewMargins:(NSSize)offsetSize
```

Sets the horizontal and vertical distance between the border of the box and its content view. Both distances are recorded in `offsetSize`. The horizontal value is applied (reckoned in the box's coordinate system) fully and equally to the left and right sides of the box. The vertical value is similarly applied to the top and bottom. Unlike changing a box's other attributes, such as its title position or border type, changing the offsets *doesn't* automatically resize the content view. In general, you should send a `sizeToFit` message to the box after changing the size of its offsets. This causes the content view to remain unchanged while the box is wrapped around it.

setFrameFromContentFrame:

```
- (void)setFrameFromContentFrame:(NSRect)contentFrame
```

Resizes the box to accommodate `contentFrame`. See also `setContentViewMargins:`.

setTitle:

```
- (void)setTitle:(NSString *)aString
```

Sets the box's title to `aString`. By default, a box's title is "Title". After invoking this method you should send a `sizeToFit` message to the box to ensure that it's wide enough to accommodate the length of the title. See also `title`, `titleFont`.

setTitleFont:

```
- (void)setTitleFont:(NSFont *)fontObj
```

Sets the font of the title to `fontObj`.

setTitlePosition:

```
- (void)setTitlePosition:(NSTitlePosition)aPosition
```

Sets the title position to `aPosition`, which can be one of the values listed in the following table. The default position is `NSAtTop`.

Table 1-4 Title Positions for an `NSBox`

Constant	Meaning
`NSNoTitle`	The box has no title.
`NSAboveTop`	Title positioned above the box's top border.
`NSAtTop`	Title positioned within the box's top border.
`NSBelowTop`	Title positioned below the box's top border.
`NSAboveBottom`	Title positioned above the box's bottom border.
`NSAtBottom`	Title positioned within the box's bottom border.
`NSBelowBottom`	Title positioned below the box's bottom border.

If the new title position changes the size of the box's border area, the content view is resized to absorb the difference. The box isn't redisplayed. See also `titlePosition`.

sizeToFit

```
- (void)sizeToFit
```

Resizes and moves the box's content view so that it just encloses its subviews. The box itself is then moved and resized to wrap around the content view. The box's width is constrained so that its title will be fully displayed.

You should invoke this method after:

- Adding a subview (to the content view)
- Altering the size or location of such a subview
- Setting the box's offsets
- Setting the box's title

The mechanism by which the content view is moved and resized depends on whether the object responds to its own `sizeToFit` message. If it does respond, then that message is sent, and the content view is expected to be so modified. If the content view doesn't respond, the box moves and resizes the content view itself.

title

```
- (NSString *)title
```

Returns the box's title. By default, a box's title is "Title". See also `setTitle:`.

titleCell

```
- (id)titleCell
```

Returns the `NSCell` used to draw the title. See also `NSCell`.

titleFont

```
- (NSFont *)titleFont
```

Returns the `NSFont` used to draw the title. See also `NSFont`.

titlePosition

```
- (NSTitlePosition)titlePosition
```

Returns a constant representing the title position. See the description of `setTitlePosition:` for a list of the title position constants.

titleRect

```
- (NSRect)titleRect
```

Returns the rectangle in which the title is drawn. See also `NSRect`.

NSBrowser

Inherits From:	NSControl : NSView : NSResponder : NSObject
Conforms To:	NSCoding (NSResponder), NSObject (NSObject)
Declared In:	AppKit/NSBrowser.h

Class Description

`NSBrowser` provides a user interface for displaying and selecting items from a list, or from hierarchically organized lists such as directory paths. Hierarchical list levels are displayed in columns, which are numbered from left to right, beginning with 0. Each column consists of an `NSScrollView` containing an `NSMatrix` filled with `NSBrowserCell`s. `NSBrowser` relies on a delegate to provide the data in its `NSBrowserCell`s. See the `NSBrowserCell` class description for more on its implementation.

Browser Selection

An entry in an `NSBrowser`'s column can be either a branch node (such as a directory) or a leaf node (such as a file). When the user selects a single branch node entry in a column, the `NSBrowser` sends itself the `addColumn` message, which messages its delegate to load the next column. The user's selection can be represented as a character string; if the selection is hierarchical (for example, a filename within a directory), each component of the path to the selected node is separated by "`/`". To use some other character as the delimiter, invoke `setPathSeparator:`.

An `NSBrowser` can be set to allow selection of multiple entries in a column, or to limit selection to a single entry. When set for multiple selection, it can also be set to limit multiple selection to leaf nodes only, or to allow selection of both types of nodes together.

As a subclass of `NSControl`, `NSBrowser` has a target object and action message. Each time the user selects one or more entries in a column, the action message is sent to the target. `NSBrowser` also adds an action to be sent when the user double-clicks on an entry, which allows the user to select items without any action being taken, and then double-click to invoke some useful action such as opening a file.

User Interface Features

The user interface features of an NSBrowser can be changed in a number of ways. The NSBrowser may or may not have a horizontal scroller. (The NSBrowser's columns, by contrast, always have vertical scrollers—although a scroller's buttons and knob might be invisible if the column doesn't contain many entries.) You generally shouldn't create an NSBrowser without a horizontal scroller; if you do, you must make sure the bounds rectangle of the NSBrowser is wide enough that all the columns can be displayed. An NSBrowser's columns may be bordered and titled, bordered and untitled, or unbordered and untitled. A column's title may be taken from the selected entry in the column to its left, or may be provided explicitly by the NSBrowser or its delegate.

NSBrowser's Delegate

NSBrowser requires a delegate to provide it with data to display. The delegate is responsible for providing the data and for setting each item as a branch or leaf node, enabled or disabled. It can also receive notification of events like scrolling and requests for validation of columns that may have changed. Note that NSBrowser does not raise an exception if the delegate does not implement the data source methods.

You can implement one of two delegate types: active or passive. An active delegate creates a column's rows (that is, the NSBrowserCells) itself, while a passive one leaves that job to the NSBrowser. Normally, passive delegates are preferable, because they're easier to implement. An active delegate must implement browser:createRowsForColumn:inMatrix: to create the rows of the specified column. A passive delegate, on the other hand, must implement browser:numberOfRowsInColumn: to let the NSBrowser know how many rows to create. These two methods are mutually exclusive; you can implement one or the other, but not both. The NSBrowser ascertains what type of delegate it has by which method the delegate responds to.

Both types of delegate implement browser:willDisplayCell:atRow:column: to set up state (such as the cell's string value and whether the cell is a leaf or a branch) before an individual cell is displayed. This delegate method doesn't need to invoke NSBrowserCell's setLoaded: method, because the NSBrowser can determine that state by itself. An active delegate can instead set the state of all

the cells at the time the cells are created, in which case it doesn't need to implement `browser:willDisplayCell:atRow:column:`. However, a passive delegate must always implement this method.

Methods Types

Activity	Class Method
Setting the delegate	– delegate – setDelegate:
Target and action	– doubleAction – sendAction – setDoubleAction
Setting component classes	+ cellClass – cellPrototype – matrixClass – setCellClass: – setCellPrototype: – setMatrixClass:
Setting NSBrowser behavior	– reusesColumns – setReusesColumns: – setTakesTitleFromPreviousColumn: – takesTitleFromPreviousColumn
Allowing different types of selection	– allowsBranchSelection – allowsEmptySelection – allowsMultipleSelection – setAllowsBranchSelection: – setAllowsEmptySelection: – setAllowsMultipleSelection:
Setting arrow key behavior	– acceptsArrowKeys – sendsActionOnArrowKeys – setAcceptsArrowKeys: – setSendsActionOnArrowKeys:
Showing a horizontal scroller	– hasHorizontalScroller – setHasHorizontalScroller:
Setting the NSBrowser's appearance	– maxVisibleColumns – minColumnWidth – separatesColumns – setMaxVisibleColumns: – setMinColumnWidth: – setSeparatesColumns:

Activity	Class Method
Manipulating columns	– addColumn – columnOfMatrix: – displayAllColumns – displayColumn: – firstVisibleColumn – isLoaded – lastColumn – lastVisibleColumn – loadColumnZero – numberOfVisibleColumns – reloadColumn: – selectAll: – selectedColumn – selectedRowInColumn: – selectRow:inColumn: – setLastColumn: – validateVisibleColumns
Manipulating column titles	– drawTitleOfColumn:inRect: – isTitled – setTitled: – setTitle:ofColumn: – titleFrameOfColumn: – titleHeight – titleOfColumn:
Scrolling an NSBrowser	– scrollColumnsLeftBy: – scrollColumnsRightBy: – scrollColumnToVisible: – scrollViaScroller: – updateScroller
Event handling	– doClick: – doDoubleClick:
Getting matrices and cells	– loadedCellAtRow:column: – matrixInColumn: – selectedCell – selectedCellInColumn: – selectedCells
Getting frame columns	– frameOfColumn: – frameOfInsideOfColumn:

Activity	Class Method
Manipulating paths	– path – pathSeparator – pathToColumn: – setPath: – setPathSeparator:
Arranging an NSBrowser's components	– tile
Methods Implemented by the Delegate	– browser:createRowsForColumn:inMatrix: – browser:isColumnValid: – browser:numberOfRowsInColumn: – browser:selectCellWithString:inColumn: – browser:selectRow:inColumn: – browser:titleOfColumn: – browser:willDisplayCell:atRow:column: – browserDidScroll:

Class Methods

cellClass

```
+ (Class)cellClass
```

Returns the `NSBrowserCell` class regardless of whether a `setCellClass:` message has been sent to a particular instance.

Instance Methods

acceptsArrowKeys

```
- (BOOL)acceptsArrowKeys
```

Returns `YES` if the arrow keys are enabled. If these keys are enabled, then the keyboard arrow keys move the selection whenever the `NSBrowser` or one of its subviews is the first responder. A return value of `NO` indicates that the arrow keys are disabled.

addColumn

```
- (void)addColumn
```

Adds a column to the right of the last column in the NSBrowser and, if necessary, scrolls the NSBrowser so that the new column is visible. Your code should never invoke this method; it's invoked as needed by doClick: and keyDown: (NSResponder) when the user selects a single branch node entry in the NSBrowser, and by setPath: when it matches a path substring with a branch node entry. Override this method if you need the NSBrowser to do any additional updating when a column is added, but be sure to send this message to super. See also loadColumnZero, reloadColumn:, setPath:.

allowsBranchSelection

```
- (BOOL)allowsBranchSelection
```

Returns YES if the user can select branch items when multiple selection is enabled; otherwise returns NO.

allowsEmptySelection

```
- (BOOL)allowsEmptySelection
```

Returns YES if nothing can be selected.

allowsMultipleSelection

```
- (BOOL)allowsMultipleSelection
```

Returns YES if the user can select multiple items.

cellPrototype

```
- (id)cellPrototype
```

Returns the browser's prototype NSCell. This cell is copied to create new cells in the columns of the browser. See also setCellPrototype:.

columnOfMatrix:

```
- (int)columnOfMatrix:(NSMatrix *)matrix
```

Returns the index of the column containing matrix; the leftmost (root) column is 0. Returns –1 if no column contains matrix. See also matrixInColumn:.

delegate

```
- (id)delegate
```

Returns the browser's delegate. The browser delegate provides the browser data, and responds to certain notification messages. See also `setDelegate:`.

displayAllColumns

```
- (void)displayAllColumns
```

Redisplays all currently visible browser columns. This method is useful for redisplaying the browser after manipulating it with display disabled in the window, for instance if `NSCell`s in some of the columns are deleted.

displayColumn:

```
- (void)displayColumn:(int)column
```

Validates and displays column number `column`. `column` must already be loaded. This method is useful for updating the browser after manipulating `column` with display disabled in the window. See also `displayAllColumns`.

doClick:

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

Your code should never invoke this method. This is the action message sent to the browser by a column's `NSMatrix` when a mouse-down event occurs in a column. It sets the browser's last column to that of the `NSMatrix` where the click occurred, and removes any columns to the right that were previously loaded in the browser. If a single branch node entry is selected by the event, this method sends `addColumn` to `self` to display the corresponding data in the column to the right. It also sends the browser's action message to its target. Override this method to add specific behavior for mouse clicks. See also `action` (`NSControl`), `target` (`NSControl`), `doDoubleClick:`.

doDoubleClick:

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

Your code should never invoke this method. This is the action message sent to the browser by a column's NSMatrix when a double-click occurs in a column. This method simply sends the double-click action message to the target; if no double-click action message is set, it sends the regular (single-click) action. You may want to override this method to add specific behavior for double-click events. See also doubleAction, target (NSControl), doClick:.

doubleAction

```
- (SEL)doubleAction
```

Returns the action sent by the browser to its target when the user double-clicks an entry. If no double-click action message has been set, this method returns the regular (single-click) action. See also setDoubleAction, action (NSControl), target (NSControl), doDoubleClick:.

drawTitleOfColumn:inRect:

```
- (void)drawTitle:(NSString *)title inRect:(NSRect)aRect
    ofColumn:(int)column
```

Your code should never invoke this method. It's invoked whenever the browser needs to draw a column title. You may override it if you want your own column titles drawn.

drawTitleOfColumn:inRect:

firstVisibleColumn

```
- (int)firstVisibleColumn
```

Returns the index of the first visible column.

frameOfColumn:

```
- (NSRect)frameOfColumn:(int)column
```

Returns the rectangle containing the column at index column.

frameOfInsideOfColumn:

```
- (NSRect)frameOfInsideOfColumn:(int)column
```

Returns the rectangle containing the column at index `column`, not including borders.

hasHorizontalScroller

```
- (BOOL)hasHorizontalScroller
```

Returns `YES` if the browser has horizontal `NSScroller`.

isLoaded

```
- (BOOL)isLoaded
```

Returns `YES` if any of the browser's columns are loaded. See also `loadColumnZero`, `setPath:`.

isTitled

```
- (BOOL)isTitled
```

Returns `YES` if the browser's columns are displayed with titles above them and `NO` otherwise. See also `setTitled:`.

lastColumn

```
- (int)lastColumn
```

Returns the index of the last loaded column in the browser. See also `lastVisibleColumn`.

lastVisibleColumn

```
- (int)lastVisibleColumn
```

Returns the index of the rightmost visible column. This may be less than the value returned by `lastColumn` if the browser has been scrolled left. See also `firstVisibleColumn`, `lastColumn`.

loadColumnZero

```
- (void)loadColumnZero
```

Loads and displays data in column 0 of the browser, unloading any columns to the right that were previously loaded. Invoke this method to force the browser to be loaded, for example, after initializing the browser, when changing delegates, or when changing the data set managed by the delegate. You may want to override this method if you subclass `NSBrowser`. See also `setPath:`, `reloadColumn:`.

`loadedCellAtRow:column:`

```
- (id)loadedCellAtRow:(int)row column:(int)column
```

Returns the cell at `row` in `column`, if that column is currently in the browser. This method creates and loads the cell if necessary. This method is the safest way to get a particular cell in a column, since lazy delegates don't load every cell in a matrix and very lazy delegates don't even create all cells until they're displayed. This method is preferred to the matrix method `cellAtRow:column:`. If the specified `column` isn't in the browser, or if `row` doesn't exist in `column`, this method returns `nil`.

`matrixClass`

```
- (Class)matrixClass
```

Returns the `NSMatrix` class used in the browser's columns.

`matrixInColumn:`

```
- (NSMatrix *)matrixInColumn:(int)column
```

Returns the `NSMatrix` found in column number `column`. Returns `nil` if column number `column` isn't loaded in the browser.

`maxVisibleColumns`

```
- (int)maxVisibleColumns
```

Returns the maximum number of visible columns allowed. No matter how many loaded columns the browser contains, or how large the browser is made (for example, by resizing its window), it will never display more than this number of columns. If the number of loaded columns can exceed the value

returned by this method, the browser must display left and right scroll buttons. See also `setMaxVisibleColumns:`, `numberOfVisibleColumns`, `setHorizontalScroller:` (`NSScrollView`).

minColumnWidth

```
- (float)minColumnWidth
```

Returns the minimum width of a column in PostScript points. No column will be smaller than the returned value unless the browser itself is smaller than that. The default setting is 100 points. See also `setMinColumnWidth:`.

numberOfVisibleColumns

```
- (int)numberOfVisibleColumns
```

Returns the number of browser columns that can be visible at the same time (that is, the current width, in columns, of the browser). This may be less than the value returned by `maxVisibleColumns` if the window containing the browser has been resized. See also `setMaxVisibleColumns:`, `maxVisibleColumns`.

path

```
- (NSString *)path
```

Returns the browser's current path.

pathSeparator

```
- (NSString *)pathSeparator
```

Returns the path separator. The default is "`/`".

pathToColumn:

```
- (NSString *)pathToColumn:(int)column
```

Returns a string representing the path from the first column to the column at index `column`.

reloadColumn:

```
- (void)reloadColumn:(int)column
```

Reloads `column` if it is loaded and sets it as the last column. `column` is reloaded by sending a message to the delegate to update the `NSCell`s in `column`'s `NSMatrix`, then reselecting the previously selected `NSCell` if it's still in the matrix. You should never send this message for a column that hasn't been loaded (you can check for this with the `lastColumn` method).

reusesColumns

```
- (BOOL)reusesColumns
```

Returns `YES` if the `NSMatrix` objects aren't freed when their columns are unloaded. See also `setReusesColumns:`.

scrollColumnsLeftBy:

```
- (void)scrollColumnsLeftBy:(int)shiftAmount
```

Scrolls the columns in the browser left by `shiftAmount` columns, making higher numbered columns visible. If `shiftAmount` exceeds the number of loaded columns to the right of the first visible column, then the columns scroll left to make the last loaded column visible. See also `scrollColumnsRightBy:`.

scrollColumnsRightBy:

```
- (void)scrollColumnsRightBy:(int)shiftAmount
```

Scrolls the columns in the browser right by `shiftAmount` columns, making lower numbered columns visible. If `shiftAmount` exceeds the number of columns to the left of the first visible column, then the columns scroll right until column 0 is visible. See also `scrollColumnsLeftBy:`.

scrollColumnToVisible:

```
- (void)scrollColumnToVisible:(int)column
```

Scrolls the browser to make the column numbered `column` visible. If there is no column numbered `column` in the browser, this method scrolls to the right as far as possible.

scrollViaScroller:

```
- (void)scrollViaScroller:(NSScroller *)sender
```

Scrolls the browser's columns left or right based on the position of the `NSScroller` sending the message. This message is sent automatically, so your code shouldn't send this message. You may want to override it to provide different behavior.

selectAll:

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

Selects all `NSCell`s in the last column of the browser. See also `setAllowsMultipleSelection:`.

selectedCell

```
- (id)selectedCell
```

Returns the last (rightmost and lowest) selected `NSCell`.

selectedCellInColumn:

```
- (id)selectedCellInColumn:(int)column
```

Returns the last (lowest) `NSCell` that's selected in `column`.

selectedCells

```
- (NSArray *)selectedCells
```

Returns all the rightmost selected `NSCell`s.

selectedColumn

```
- (int)selectedColumn
```

Returns the column number of the rightmost column containing a selected `NSCell`. This won't be the last column if the selected cell isn't a leaf. Returns –1 if no column in the browser contains a selected cell. See also `lastColumn`.

selectedRowInColumn:

```
- (int)selectedRowInColumn:(int)column
```

Returns the selected row number within the given column. Returns -1 if no row is selected. See also `selectRow:inColumn:`, `selectedCellInColumn:`.

selectRow:inColumn:

```
- (void)selectRow:(int)row inColumn:(int)column
```

Selects row number `row` in column number `column`. See also `selectedRowInColumn:`, `selectedColumn`.

sendAction

```
– (BOOL)sendAction
```

Sends the action message to the target. Returns `YES` upon success and `NO` if no responder for the message could be found. See also `sendAction:to:` (`NSControl`).

sendsActionOnArrowKeys

```
– (BOOL)sendsActionOnArrowKeys
```

Returns `NO` if pressing an arrow key only scrolls the browser and `YES` if it also sends the action message specified by `setAction:` (`NSControl`).

separatesColumns

```
– (BOOL)separatesColumns
```

Returns `YES` if the browser's columns are separated by bezeled borders, and returns `NO` otherwise. When titles are set to display (by `setTitled:`), columns are automatically separated by such borders. See also `setTitled:`.

setAcceptsArrowKeys:

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

Enables or disables the arrow keys. See also `acceptsArrowKeys`.

setAllowsBranchSelection:

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

Determiness whether the user can select multiple branch and leaf node entries. If `flag` is `YES` and multiple selection is enabled by `setAllowsMultipleSelection:`, then multiple branch and leaf node entries can be selected. By default, a user can choose only multiple leaf node entries when multiple entry selection is enabled. See also `allowsBranchSelection`, `setAllowsMultipleSelection:`.

setAllowsEmptySelection:

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

If `flag` is `YES`, the browser can display without any `NSCell`s selected; if `flag` is `NO`, then there must always be at least one cell selected. By default, the setting is `NO`, and the browser selects the first item in the first column. See also `allowsEmptySelection`, `setAllowsMultipleSelection:`.

setAllowsMultipleSelection:

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

Sets whether the user can select multiple items in a column. If `flag` is `YES`, the user can choose any number of leaf entries in a column (or leaf and branch entries in a column if enabled by `setAllowsBranchSelection:`). By default, the user can choose just one entry in a column at a time. See also `allowsMultipleSelection`, `setAllowsBranchSelection:`.

setCellClass:

```
- (void)setCellClass:(Class)classId
```

Sets the NSCell class used when adding cells to an NSMatrix in a column of the browser. classId must be the value returned when sending the class message to NSBrowserCell (or subclass). Since a browser always has its matrices copy prototype cells, this method simply makes a prototype, sends it an init message, and records that prototype. You shouldn't use NSControl's class method setCellClass: with an NSBrowser. See also setCellPrototype:.

setCellPrototype:

```
- (void)setCellPrototype:(NSCell *)aCell
```

Sets aCell as the NSCell prototype copied when adding cells to the matrices in the columns of the browser. aCell must be an instance of NSBrowserCell or its subclass. Each NSMatrix gets its own copy of aCell to use as a prototype, and will free that copy when the matrix is freed. Don't use NSControl's class method setCellClass: with an NSBrowser. See also cellPrototype, setCellClass:.

setDelegate:

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

Sets the browser's delegate to anObject. See also delegate.

setDoubleAction

```
- (void)setDoubleAction:(SEL)aSelector
```

Sets the double-click action of the browser. aSelector is the selector for the action message sent to the target when a double-click occurs in one of the columns of the browser. See also doubleAction, setAction: (NSControl), setTarget: (NSControl), doDoubleClick:.

setHasHorizontalScroller:

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

If `flag` is `YES`, this method makes the browser use a horizontal `NSScroller`. Generally, you should allow your browser to scroll horizontally unless your data is nonhierarchical, and thus limited to a single column, or restricted so that the browser will always display enough columns for all data. See also `hasHorizontalScroller`.

setLastColumn:

```
- (void)setLastColumn:(int)column
```

Makes column number `column` the last column loaded and displayed by the browser. Removes any columns to the right of `column` from the browser, and scrolls columns in the browser to make the new last column visible if it wasn't previously. If column number `column` isn't already loaded, this method does nothing. See also `lastColumn`.

setMatrixClass:

```
- (void)setMatrixClass:(Class)classId
```

Sets the `NSMatrix` class used when adding new columns to the browser. `classId` must be the value returned by sending the `class` message to `NSMatrix` (or subclass); otherwise this method retains the previous setting for the browser's `NSMatrix` class. `NSBrowser` initializes the matrix of a new column with the `initWithFrame:mode:prototype:numberOfRows:numberOfColumns:` (`NSMatrix`) method.

setMaxVisibleColumns:

```
- (void)setMaxVisibleColumns:(int)columnCount
```

Sets the maximum number of columns that may be displayed by the browser. To set the number of columns displayed in a new browser, first send it a `setMinColumnWidth:` message with a small argument (1, for example) to ensure that the desired number of columns will fit in the browser's frame. Then invoke this method to set the number of columns you want your browser to display. The minimum column width may then be reestablished to its desired value. See also `maxVisibleColumns`, `setMinColumnWidth:`.

setMinColumnWidth:

```
- (void)setMinColumnWidth:(float)columnWidth
```

Sets the minimum width for each column to `columnWidth`. If the new minimum width is different from the previous one, this method also redisplays the browser with columns set to the new width. `columnWidth` is measured in PostScript points. The default setting is 100. See also `minColumnWidth`.

setPath:

```
- (BOOL)setPath:(NSString *)path
```

Parses `aPath`—a string consisting of one or more substrings separated by the path separator—and selects column entries in the browser that match the substrings. If the first character in `aPath` is the path separator, this method begins searching for matches in column 0; otherwise, it begins searching in the last column loaded. If no column is loaded, this method loads column 0 and begins the search there. While parsing the current substring, it tries to locate a matching entry in the search column. If it finds an exact match, this method selects that entry and moves to the next column (loading the column if necessary) to search for the next substring.

If this method finds a valid path (one in which each substring is matched by an entry in the corresponding column), it returns `YES`. If it doesn't find an exact match on a substring, it stops parsing `aPath` and returns `NO`; however, column entries that it has already selected remain selected. Your code should never try to set a path or select items by sending `NSCell` selection messages to the `NSMatrix`es in the browser's columns. This procedure bypasses every mechanism that allows the browser to update its display and load columns and cells properly. See also `pathToColumn:`, `pathSeparator`, `setPathSeparator:`, `browser:selectCellWithString:inColumn:`.

setPathSeparator:

```
- (void)setPathSeparator:(NSString *)aString
```

Sets the character used as the path separator; the default is the slash character ("/"). See also `pathToColumn:`, `setPath:`.

setReusesColumns:

- (void)setReusesColumns:(BOOL)flag

Sets whether the browser saves a column's NSMatrix and NSClipView or NSScrollView when the column is unloaded, and whether it then reuses these subviews when the column is reloaded. If flag is YES, the browser reuses columns for somewhat faster display of columns as they are reloaded. If flag is NO, the browser frees columns as they're unloaded, reducing average memory use. See also reusesColumns.

setSendsActionOnArrowKeys:

- (void)setSendsActionOnArrowKeys:(BOOL)flag

Determines whether pressing an arrow key will cause the action message to be sent (in addition to causing scrolling). If flag is YES, then when an arrow key is pressed, the browser's action message is sent as though the user had clicked on the new selection; if flag is NO, then arrow keys only move the selection (if they are enabled). See also setAcceptsArrowKeys:.

setSeparatesColumns:

- (void)setSeparatesColumns:(BOOL)flag

If flag is YES, sets the browser so that columns have bezeled borders separating them; if flag is NO, the borders are removed. When titles are set to display by setTitled:, columns are automatically separated. Redraws the browser. See also separatesColumns.

setTakesTitleFromPreviousColumn:

- (void)setTakesTitleFromPreviousColumn:(BOOL)flag

Sets whether the title of a column is set to the string value of the selected NSCell in the previous column. If flag is YES, then each browser column takes its title from the string value in the selected NSCell in the column to its left, leaving column 0 untitled; use setTitle:ofColumn: to give column 0 a title. This method affects the receiver only when it is titled, that is, when isTitled returns YES. By default, the browser is set to get column titles from the previous column. Send this message with NO as the argument if your

delegate implements the `browser:titleOfColumn:` method, or if you use the `setTitle:ofColumn:` method to set all column titles. See also `isTitled`, `setTitled:`, `setTitle:ofColumn:`, `browser:titleOfColumn:`.

setTitle:ofColumn:

```
- (void)setTitle:(NSString *)aString ofColumn:(int)column
```

Sets the title `column` in the browser to `aString`. If `column` isn't loaded, this method does nothing. See also `setTakesTitleFromPreviousColumn:`, `setTitled:`, `browser:titleOfColumn:`.

setTitled:

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

If `flag` is `YES`, columns display titles and are separated by bezeled borders. Otherwise no titles are displayed. See also `setTakesTitleFromPreviousColumn:`, `setTitle:ofColumn:`, `browser:titleOfColumn:`.

takesTitleFromPreviousColumn

```
- (BOOL)takesTitleFromPreviousColumn
```

Returns `YES` if the title of a column is set to the string value of the selected `NSCell` in the previous column.

tile

```
- (void)tile
```

Arranges the various subviews of `NSBrowser`—scrollers, columns, titles, and so on—without redrawing. Your code shouldn't send this message. It is invoked any time the appearance of the browser changes, for example, when scroll buttons or scroll bars are set, a column is added, and so on. Override this method if your code changes the appearance of the browser (for example, if you draw your own titles above columns).

titleFrameOfColumn:

```
- (NSRect)titleFrameOfColumn:(int)column
```

Returns the bounds of the title frame for the column at index `column`.

titleHeight

```
- (float)titleHeight
```

Returns the height of titles drawn above the columns of the browser. Override this method if you display your own titles above the browser's columns.

titleOfColumn:

```
- (NSString *)titleOfColumn:(int)column
```

Returns the title displayed for the column at index `column`.

updateScroller

```
- (void)updateScroller
```

Updates the horizontal scroller to reflect the position of the visible columns of the browser.

validateVisibleColumns

```
- (void)validateVisibleColumns
```

Validates the columns visible in the browser by invoking the delegate method `browser:isColumnValid:` for all visible columns. Use this method to confirm that the entries displayed in each visible column are valid before redrawing. See also `browser:isColumnValid:`.

Methods Implemented by the Delegate

browser:createRowsForColumn:inMatrix:

```
- (void)browser:(NSBrowser *)sender createRowsForColumn:(int)column
    inMatrix:(NSMatrix *)matrix
```

Creates a row in `matrix` for each row of data to be displayed in `column` of the browser. Either this method or `browser:numberOfRowsInColumn:` must be implemented, but not both.

browser:isColumnValid:

```
- (BOOL)browser:(NSBrowser *)sender isColumnValid:(int)column
```

This method is invoked by NSBrowser's validateVisibleColumns method to determine whether the contents currently loaded in column number column need to be updated. This is useful for data sets that may change over time, such as files in a file system, or lists from a shared set of data that others can change. This method returns YES if the contents are valid and returns NO otherwise. See also browser:selectCellWithString:inColumn:.

browser:numberOfRowsInColumn:

```
- (int)browser:(NSBrowser *)sender numberOfRowsInColumn:(int)column
```

Implemented by very lazy delegates, this method is invoked by the browser to ask the delegate for the number of rows in column number column. This method allows the browser to resize its scroll bar for a column without loading all the cells in that column. Returns the number of rows in column. If you implement this method, don't implement the delegate method browser:createRowsForColumn:inMatrix:.

browser:selectCellWithString:inColumn:

```
- (BOOL)browser:(NSBrowser *)sender
    selectCellWithString:(NSString *)title
    inColumn:(int)column
```

Asks NSBrowser's delegate to validate and select an entry in column number column. This method should load the NSCell with the title title if necessary, send it setLoaded: (NSBrowserCell) and setLeaf: (NSBrowserCell) messages as needed to indicate its state, and send the column's NSMatrix a selectCellAtRow:Column: message to select that cell. If there is no cell with the title title, the selection should be cleared by sending selectCellAtRow:Column: to the NSMatrix with –1 and –1 as the arguments. This method returns YES if the method successfully selects the NSCell with the title title in column and NO otherwise.

If the delegate doesn't implement this method, the browser searches for entries by scanning through the entire list of cells in the column. This will always work properly for browsers that browse static data. However, if the data can change while the browser is in use, for example if a new file is created or

deleted, this method lets the delegate find that new data and add it to the column; if the delegate finds that the data no longer exists, or that its status has changed, it should mark it as disabled or remove the cell with matrix's `removeRow:` method. Do not forget to free the cell. See also `browser:isColumnValid:`, `matrixInColumn:`, `selectCellAtRow:column:` (`NSMatrix`), `removeRow:` (`NSMatrix`).

browser:selectRow:inColumn:

```
- (BOOL)browser:(NSBrowser *)sender
    selectRow:(int)row inColumn:(int)column
```

Asks `NSBrowser`'s delegate to select `row` within `column`. This method returns `YES` if `row` is selected, and returns `NO` otherwise. See also `browser:isColumnValid:`, `matrixInColumn:`, `selectCellAtRow:Column:` (`NSMatrix`), `removeRowAt:` (`NSMatrix`).

browser:titleOfColumn:

```
– (NSString *)browser:(NSBrowser *)sender titleOfColumn:(int)column
```

Invoked by `NSBrowser` to get the title for `column` from the delegate. This method is invoked if the delegate implements it, but only when the browser is titled and has received a `setTakesTitleFromPreviousColumn:` message with `NO` as the argument. By default, the browser makes each column title the string value of the selected cell in the previous column. Returns the `NSString` representing the title belonging above column number `column`. See also `setTakesTitleFromPreviousColumn:`, `setTitle:ofColumn:`, `setTitled:`.

browser:willDisplayCell:atRow:column:

```
– (void)browser:(NSBrowser *)sender willDisplayCell:(id)cell
    atRow:(int)row column:(int)column
```

Notifies the delegate when the browser will display the specified cell. The delegate should set any state necessary for correct display of the cell. Implemented by lazy and very lazy delegates, this method loads the entry in the provided `NSBrowserCell` `cell` for the specified row and column in the browser. The browser will resize the cell to fit in the matrix—you can't control the size of an `NSBrowserCell`. A lazy delegate should send a `setLoaded:` message to `cell` at load time; it can send `setLeaf:`, `setStringValue:`, and

setEnabled: messages at load time or later. A very lazy delegate should send setLoaded:, setLeaf:, and setStringValue: messages to cell at load time, and setEnabled: when needed. See also browser:numberOfRowsInColumn:, setEnabled: (NSCell), setLeaf: (NSBrowserCell), setLoaded: (NSBrowserCell), setStringValue: (NSCell).

browserDidScroll:

```
- (void)browserDidScroll:(NSBrowser *)sender
```

Notifies the delegate when the browser has finished scrolling horizontally. This can be useful for aligning other user interface items with the columns of the browser (for example, an icon path or a series of pop-up lists). See also browserWillScroll:.

browserWillScroll:

```
- (void)browserWillScroll:(NSBrowser *)sender
```

This method notifies the delegate when the browser is about to scroll horizontally. This notification can be useful for hiding other user interface items to prepare for aligning them with the columns of the browser (for example, an icon path or a series of pop-up lists). See also browserDidScroll:.

NSBrowserCell

Inherits From:	NSCell : NSObject
Conforms To:	NSCoding, NSCopying (NSCell), NSObject (NSObject)
Declared In:	AppKit / NSBrowserCell.h

Class Description

NSBrowserCell is the subclass of NSCell used by default to display data in the columns of an NSBrowser. (Each column contains an NSMatrix filled with NSBrowserCells.) Many of NSBrowserCell's methods are designed to interact with NSBrowser and NSBrowser's delegate. The delegate implements

methods for loading the NSCells in NSBrowser by setting their values and status. If your code needs access to a specific NSBrowserCell, you can use the NSBrowser method loadedCellAtRow:column:.

You may find it useful to create a subclass of NSBrowserCell to alter its behavior and to enable it to work with and display the type of data you wish to represent. Use NSBrowser's setCellClass: or setCellPrototype: methods to have it use your subclass. See NSBrowser for more details. In particular, the class description and the "Methods Implemented by the Delegate" section describes how the NSBrowser's delegate interacts with both NSBrowser and NSBrowserCells.

Method Types

Activity	Class Method
Accessing graphic attributes	+ branchImage + highlightedBranchImage – alternateImage – setAlternateImage:
Placing in the browser hierarchy	– isLeaf – setLeaf:
Determining loaded status	– isLoaded – setLoaded:
Setting state	– reset – set

Class Methods

branchImage

```
+ (NSImage *)branchImage
```

Returns the NSImage object named NSmenuArrow. This is the icon displayed to indicate a branch node in an NSBrowserCell. Override this method if you want your subclass to display a different branch icon. See also isLeaf.

highlightedBranchImage

```
+ (NSImage *)highlightedBranchImage
```

Returns the default NSImage for branch NSBrowserCells that are highlighted. This is the NSImage object named "NSmenuArrowH" and is the highlighted icon displayed to indicate a selected branch node in an NSBrowserCell. Override this method if you want your subclass to display a different branch icon. See also isLeaf.

Instance Methods

alternateImage

```
- (NSImage *)alternateImage
```

Returns the NSImage that appears on the browser cell when it's in its alternate (or highlighted) state, or returns nil if there's no such image. See also setAlternateImage:, setImage: (NSCell), image (NSCell).

isLeaf

```
- (BOOL)isLeaf
```

Determines whether the entry in the receiver represents a leaf node (such as a file) or branch node (such as a directory). This method is invoked by NSBrowser to check whether to display the branch icon in the cell and, when a browser cell is selected, whether to load a column to the right of the column containing the receiving cell. Returns YES if the cell represents a leaf, and NO if the cell represents a branch. See also setLeaf:.

isLoaded

```
- (BOOL)isLoaded
```

Returns YES if the browser cell is loaded and NO if it isn't. This method is used by NSBrowser to determine if a particular cell is loaded in a column. When a browser cell is created, this value is YES; however, if the browser cell is created by the NSBrowser, the browser sets the value to NO so that the delegate can properly set the loaded status. NSBrowser and its delegate change the value returned by this method using the setLoaded: method to reflect the current status of the cell. See also setLoaded:.

reset

```
- (void)reset
```

Unhighlights the browser cell and sets its state to 0. See also `set`.

set

```
- (void)set
```

Highlights the browser cell and sets its state to 1. See also `reset`.

setAlternateImage:

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

Sets the browser cell's alternate image to `anImage`. If an alternate image has been set, it is displayed when an browser cell is highlighted. This method frees the previous alternate image—if any—before the new image is set. Consequently, if you will be resetting the alternate image and you don't want a particular image freed, use a copy:

```
[theCell setAlternateImage:[[theImage imageNamed:sharedImage]
                                copy]];
```

See also `alternateImage`, `setImage:` (`NSCell`), `image` (`NSCell`).

setLeaf:

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

Invoked by `NSBrowser`'s delegate when it loads a browser cell, this method sets whether the browser cell is a leaf or a branch. If `flag` is `YES`, the browser cell is set to represent a leaf node; it will display without the branch icon. When `flag` is `NO`, the browser cell is set to represent a branch node; it will display with the branch icon. This method does not display the browser cell, even if autodisplay is on. See also `isLeaf`, `branchImage`, `highlightedBranchImage`.

setLoaded:

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

Indicates whether all the browser cell's state has been set and the cell is ready to display. This method is invoked by `NSBrowser` or its delegate to set the status of the browser cell. The delegate should send the `setLoaded:` message with `YES` as the argument when it loads the cell. See also `isLoaded`, `NSBrowser` delegate methods.

NSBundle Additions

Inherits From:	NSObject
Declared In:	AppKit/NSImage.h AppKit/NSNibLoading.h

Class Description

The Application Kit adds these methods to the Foundation Kit's `NSBundle` class. These methods become part of the class for all applications that use the Application Kit, but not for applications that don't.

Method Types

Activity	Class Method
Getting the location of images in the file system	– pathForImageResource:
Loading an Interface Builder file	+ loadNibFile:externalNameTable:withZone: – loadNibFile:externalNameTable:withZone: + loadNibNamed:owner:

Class Methods

loadNibFile:externalNameTable:withZone:

```
+ (BOOL)loadNibFile:(NSString *)fileName
    externalNameTable:(NSDictionary *)context
    withZone:(NSZone *)zone
```

Unarchives the contents of the nib file whose absolute path is `fileName`. Objects from the nib file are allocated in the specified `zone` of memory. The `context` argument is a name table—a dictionary whose keys are names like "`NSOwner`" and whose values are existing objects that can be referenced by the newly unarchived objects. Returns `YES` upon success.

`loadNibNamed:owner:`

```
+ (BOOL)loadNibNamed:(NSString *)aNibName owner:(id)owner
```

Similar to `loadNibFile:externalNameTable:withZone:`, but the name table's only element is the specified owner (stored with the key "`NSOwner`"). Objects from the nib file are allocated in `owner`'s zone. If there's a bundle for `owner`'s class, this method looks in that bundle for the nib file named `aNibName` (this argument need not include the "`.nib`" extension); otherwise, it looks in the main bundle.

Instance Methods

`loadNibFile:externalNameTable:withZone:`

```
- (BOOL)loadNibFile:(NSString *)fileName
    externalNameTable:(NSDictionary *)context
    withZone:(NSZone *)zone
```

Unarchives the contents of the nib file whose absolute path is `fileName`. Objects from the nib file are allocated in the specified `zone` of memory. The `context` argument is a name table—a dictionary whose keys are names like "`NSOwner`" and whose values are existing objects that can be referenced by the newly unarchived objects. Returns `YES` upon success.

`pathForImageResource:`

```
– (NSString *)pathForImageResource:(NSString *)name
```

Returns the absolute pathname of the file containing the specified image resource. (The `name` of the resource is simply the filename without the path of its bundle directory; the filename extension need not be included.)

NSButton

Class Description

inherits From:	NSControl : NSView : NSResponder : NSObject
Conforms To:	NSCoding (NSResponder), NSObject (NSObject)
Declared In:	AppKit/NSButton.h

`NSButton` is an `NSControl` subclass that intercepts mouse-down events and sends an action message to a target object when it's clicked or pressed. By virtue of its `NSButtonCell`, `NSButton` is a two-state `NSControl`—it's either "off" or "on"—and it displays its state depending on the configuration of the `NSButtonCell`. `NSButton` acquires other attributes of `NSButtonCell`. The state is used as the value, so `NSControl` methods such as `setIntValue:` actually set the state. The methods `setState:` and `state` are provided as a more conceptually accurate way of setting and getting the state. The `NSButton` can send its action continuously and display highlighting in several different ways. An `NSButton` can also have a key equivalent that's eligible for triggering whenever the `NSButton`'s `NSPanel` or `NSWindow` is the key window.

`NSButton` and `NSMatrix` both provide a control view, which is needed to display an `NSButtonCell` object. However, while `NSMatrix` requires you to access the `NSButtonCell`s directly, most of `NSButton`'s methods are "covers" for identically declared methods in `NSButtonCell`. In other words, the implementation of the `NSButton` method invokes the corresponding `NSButtonCell` method for you, allowing you to be unconcerned with the `NSButtonCell`'s existence. The only `NSButtonCell` methods that don't have covers relate to the font used to display the key equivalent, and to specific methods for highlighting or showing the `NSButton`'s state. These last are usually set together with `NSButton`'s `setButtonType:` method.

Creating a Subclass of NSButton

Override the designated initializer (`NSView`'s `initWithFrame:` method) if you create a subclass of `NSButton` that performs its own initialization. If you want to use a custom `NSButtonCell` subclass with your `NSButton` subclass,

you have to override the `setCellClass:` method, as described in "Creating New `NSControls`" in the `NSControl` class specification. See the `NSButtonCell` class specification for more on `NSButton`'s behavior.

Method Types

Activity	Class Method
Initializing the NSButton factory	+ cellClass + setCellClass:
Setting the button type	– setButtonType:
Setting the state	– setState: – state
Setting the repeat interval	– getPeriodicDelay:interval: – setPeriodicDelay:interval:
Setting the titles	– alternateTitle – setAlternateTitle: – setTitle: – title
Setting the images	– alternateImage – image – imagePosition – setImage: – setImagePosition:
Modifying graphic attributes	– isBordered – isTransparent – setBordered: – setTransparent:
Displaying	– highlight:
Setting the key equivalent	– keyEquivalent – keyEquivalentModifierMask – setKeyEquivalent: – setKeyEquivalentModifierMask:
Handling events and action messages	– performClick: – performKeyEquivalent:

Class Methods

cellClass

```
+ (Class)cellClass
```

Returns the NSButtonCell subclass used by NSButton.

setCellClass:

```
+ (void)setCellClass:(Class)classId
```

Configures the NSButton class to use instances of classId for its cells. classId should be the id of an NSButtonCell subclass, obtained by sending the class message to either the cell subclass object or to an instance of that subclass. The default cell class is NSButtonCell. If this method isn't overridden by an NSButton subclass, then when it's sent to that subclass, NSButton and any other NSButton subclasses that don't override the methods mentioned below will use the new cell subclass as well. To safely set a cell class for your NSButton subclass, override this method to store the cell class in a static id. Also, override the designated initializer to replace the NSButton subclass instance's cell with an instance of the cell subclass stored in that static id. See "Creating New NSControls" on page 180 in the NSControl class specification's class description for more information.

Instance Methods

alternateImage

```
- (NSImage *)alternateImage
```

Returns the NSImage that appears on the button when it's in its alternate state, or nil if there is no alternate NSImage. NSButton only displays its alternate NSImage if it highlights or shows its alternate state by displaying its alternate contents. See also setAlternateImage:, setImagePosition:, image, setButtonType:.

alternateTitle

```
- (NSString *)alternateTitle
```

Returns an NSString containing the title that appears on the button when it's in its alternate state, or NULL if there isn't one. The alternate title is only displayed if the button highlights or shows its alternate state by displaying its alternate contents. See also setAlternateTitle:, title, setButtonType:.

getPeriodicDelay:interval:

```
- (void)getPeriodicDelay:(float *)delay interval:(float *)interval
```

Returns by reference the delay and interval periods for a continuous button. delay is the amount of time (in seconds) that a continuous button will pause before starting to periodically send action messages to the target object. interval is the amount of time (also in seconds) between those messages. See also setContinuous: (NSControl), setPeriodicDelay:interval:.

highlight:

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

If the highlight state of the cell is not equal to flag, the button is highlighted and the highlight state of the cell is set to flag. Highlighting may involve the button appearing "pushed in" to the screen, displaying its alternate title or icon, or lighting. This method issues a flushWindow message after highlighting the button. See also setButtonType:

image

```
- (NSImage *)image
```

Returns the NSImage that appears on the button when it's in its normal state, or nil if there is no such NSImage. This NSImage is always displayed on a button that doesn't change its contents when highlighting or showing its alternate state. See also setImage:, setImagePosition:, alternateImage, setButtonType:.

imagePosition

```
- (NSCellImagePosition)imagePosition
```

Returns the position of the button's image relative to the button's title. The return value can be any of the following enumeration constants:

- NSNoImage
- NSImageOnly
- NSImageLeft
- NSImageRight
- NSImageBelow
- NSImageAbove
- NSImageOverlaps

See also `setImagePosition:`.

isBordered

```
- (BOOL)isBordered
```

Returns `YES` if the button has a border and returns `NO` otherwise. A button's border isn't the single line of most other `NSControl`s' borders; instead, it's a raised bezel. In other objects *bezel* usually refers to a depressed bezel, as seen on `NSFormCell`s, for example.

Note – You shouldn't use the `setBezeled:` method with a button.

See also `setBordered:`.

isTransparent

```
- (BOOL)isTransparent
```

Returns `YES` if the button is transparent and `NO` otherwise. A transparent button never draws itself, but it receives mouse-down events and tracks the mouse properly. See also `setTransparent:`.

keyEquivalent

```
- (NSString *)keyEquivalent
```

Returns the button's key equivalent character, or 0 if one hasn't been defined. See also `setKeyEquivalent:`, `performKeyEquivalent:`, `keyEquivalentModifierMask`.

keyEquivalentModifierMask

```
- (unsigned int)keyEquivalentModifierMask
```

Returns a mask indicating the possible modifier keys for the button's key equivalent. See `setKeyEquivalentModifierMask:` (`NSButtonCell`) for a list of these masks.

performClick:

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

Highlights the button, sends its action message to the target object, and then unhighlights the button. Invoke this method when you want the button to behave exactly as if the user had clicked it with the mouse. See also `performKeyEquivalent:`.

performKeyEquivalent:

```
- (BOOL)performKeyEquivalent:(NSEvent *)anEvent
```

If the character in `theEvent` matches the button's key equivalent, this method simulates the user clicking the button by sending `performClick:` to `self`, and returns `YES`. Otherwise this method does nothing and returns `NO`. The button won't perform the key equivalent if there's a modal panel present that the button is not located on. See also `keyEquivalent`, `performClick:`.

setAlternateImage:

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

Makes `anImage` the button's alternate image. A button displays its alternate image only if it highlights or displays its alternate state by using its alternate contents. See also `alternateImage`, `setImagePosition:`, `setImage:`, `setButtonType:`.

setAlternateTitle:

```
- (void)setAlternateTitle:(NSString *)aString
```

Sets the title that the button displays in its alternate state to the title stored in `aString`. The alternate title is shown only if the button changes its contents when highlighting or displaying its alternate state. See also `alternateTitle`, `setTitle:`, `setButtonType:`.

setBordered:

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

If `flag` is `YES`, the button displays a border; if `NO`, the button doesn't display a border. A button's border is not the single line like most other `NSControl`s' borders; instead, it's a raised bezel. In other objects *bezel* usually refers to a depressed bezel, as seen on `NSFormCell`s, for example. This method redraws the button if the bordered state changes. See also `isBordered`.

setImage:

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

Makes `anImage` the button's icon and redraws the button. A button's icon is displayed when the button is in its normal state, or at all times if the button doesn't highlight or show state by changing its contents. See also `image`, `setImagePosition:`, `setImage:`, `setAlternateImage:`, `imageNamed:` (`NSImage`), `setButtonType:`.

setImagePosition:

```
- (void)setImagePosition:(NSCellImagePosition)aPosition
```

Sets the position of the button's image when a button simultaneously displays both text and an icon and redraws the button. `aPosition` can be one of the following constants:

- `NSNoImage`
- `NSImageOnly`
- `NSImageLeft`
- `NSImageRight`
- `NSImageBelow`
- `NSImageAbove`
- `NSImageOverlaps`

If the image is positioned above or below the title, the alignment of the text will be changed to NSCenterAlignment. This behavior can be overridden with a subsequent setAlignment: method. See also imagePosition, setAlignment: (NSControl).

setKeyEquivalent:

```
- (void)setKeyEquivalent:(NSString *)aKeyEquivalent
```

Makes aKeyEquivalent the button's key equivalent and redraws the button's inside if there is no image or alternate image set for the button. The key equivalent isn't displayed if the image position is set to NSNoImage, NSImageOnly, or NSImageOverlaps (see setImagePosition:); that is, the button must display both its title and its "image" (the key equivalent, in this case), and they must not overlap. To display a key equivalent on a button, set the image and alternate image to nil, set the key equivalent, and then set the image position. See also keyEquivalent, setImagePosition:, performKeyEquivalent:, setImage:, setAlternateImage:.

setKeyEquivalentModifierMask:

```
- (void)setKeyEquivalentModifierMask:(unsigned int)mask
```

Sets the mask that determines the possible modifier keys for button's key equivalent. See setKeyEquivalentModifierMask: (NSButtonCell) for a list of these masks.

setPeriodicDelay:interval:

```
- (void)setPeriodicDelay:(float)delay interval:(float)interval
```

Sets the message delay and interval for the button. These two values are used if the button is configured (by a setContinuous: message) to continuously send the action message to the target object while tracking the mouse. delay is the amount of time (in seconds) that a continuous button will pause before starting to periodically send action messages to the target object. interval is the amount of time (also in seconds) between those messages. The maximum value allowed for both the delay and the interval is 60.0 seconds. See also getPeriodicDelay:interval:, setContinuous: (NSControl).

setState:

```
- (void)setState:(int)value
```

Sets the button's state to `value` (0 or 1) and redraws the button. 0 is the normal or "off" state, and 1 is the alternate or "on" state. See also `state`.

setTitle:

```
- (void)setTitle:(NSString *)aString
```

Sets the title displayed by the button, when in its normal state, to the title stored in `aString`. This title is always shown on buttons that don't use their alternate contents when highighting or displaying their alternate state. Redraws the button's inside. See also `title`, `setButtonType:`.

setTransparent:

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

Sets whether the button is transparent, and redraws the button if `flag` is `NO`. A transparent button tracks the mouse and sends its action, but doesn't draw. A transparent button is useful for sensitizing an area on the screen so that an action gets sent to a target when the area receives a mouse click. See also `isTransparent`.

setButtonType:

```
- (void)setButtonType:(int)aType
```

Sets how the button highlights and shows its state, and redraws the button. The types available are for the most common button types, which are also accessible in Interface Builder; you can configure different behavior with `NSButtonCell`'s `setHighlightsBy:` and `setShowsStateBy:` methods. `aType` can be one of constants shown in Table 1-5.

Table 1-5 NSButton Types

Constant	Corresponding Button Type
NSMomentaryPushButton	The default type. While the button is held down it's displayed as lit, and also "pushed in" to the screen if the button is bordered. This type of button is best for simply triggering actions, as it doesn't show its state; it always displays its normal image or title. This option is called "Momentary Push" in Interface Builder's Button Inspector.
NSPushOnPushOffButton	Holding the button down causes it to be shown as lit, and also "pushed in" to the screen if the button is bordered. The button displays itself as lit while in its alternate state. This option is called "Push On/Push Off" in Interface Builder's Button Inspector.
NSToggleButton	Highlighting is performed by changing to the alternate title or image "pushing in." The alternate state is shown by displaying the alternate title or image. This option is called "Toggle" in Interface Builder's Button Inspector.
NSSwitchButton	A variant of NSToggleButton that has no border, and that has a default image called "switch" and an alternate image called "switchH." These are identical to the "NSswitch" and "NSswitchH" system bitmaps. This type of button is available as a separate palette item in Interface Builder.
NSRadioButton	Like NSSwitchButton, but the default image is "radio" and the alternate icon is "radioH" (identical to the "NSradio" and "NSradioH" system bit maps). This type of button is available as a separate palette item in Interface Builder.

Table 1-5 NSButton Types

Constant	Corresponding Button Type
NSMomemtaryChangeButton	While the button is pressed, the alternate image or alternate title is displayed. This type always displays its normal title or icon, that is, it doesn't display its state. The miniaturize button in a window's title bar is a good example of this type of button. This option is called "Momentary Change" in Interface Builder's Button Inspector.
NSOnOffButton	Highlights while pressed by lighting, and stays lit in its alternate state. This option is called "On / Off" in Interface Builder's Button Inspector.
NSMomentaryLight	Provides momentary light behavior. This is the equivalent of: [aButtonCell setButtonType:NSMomentaryPushButton]; [aButtonCell setCellAttribute:NSPushInCell to:No];

See also `setButtonType:` (`NSButtonCell`), `setHighlightsBy:` (`NSButtonCell`), `setShowsStateBy:` (`NSButtonCell`).

`state`

```
- (int)state
```

Returns the button's current state, either 0 for normal or "off," or 1 for alternate or "on." See also `setState:`.

`title`

```
- (NSString *)title
```

Returns the title displayed on the button when it's in its normal state, or always if the button doesn't use its alternate contents for highlighting or displaying the alternate state. Returns `NULL` if there is no title. See also `setTitle:`, `alternateImage`, `setButtonType:`.

1

NSButtonCell

Inherits From:	NSActionCell : NSCell : NSObject
Conforms To:	NSCoding, NSCopying (NSCell), NSObject (NSObject)
Declared In:	AppKit/NSButtonCell.h

Class Description

`NSButtonCell`, an `NSActionCell` subclass, implements the user interfaces of push buttons, switches, and radio buttons. It can also be used for any other region of a view that's designed to send a message to a target when clicked. `NSButton` (an `NSControl` subclass) uses a single `NSButtonCell`. To create groups of switches or radio buttons, use an `NSMatrix` holding a set of `NSButtonCell`s.

An `NSButtonCell` is a two-state cell: it's either "off" or "on," and can be configured to display the two states differently, with a separate title and/or image for either state. The two states are more often referred to as "normal" and "alternate." A button cell's state is also used as its value, so `NSCell` methods that set the value (`setIntValue:` and so on) actually set the button cell's state to "on" if the value provided is nonzero (or non-`NULL` for strings), and to "off" if the value is zero or `NULL`. Similarly, methods that retrieve the value return 1 for the "on" or alternate state (an empty string in the case of `stringValue`), or 0 or `NULL` for the "off" or normal state. You can also use `NSCell`'s `setState:` and `state` methods to set or retrieve the state directly. After changing the state, send a `display` message to show the button cell's new appearance. (`NSButton` does this automatically.)

An `NSButtonCell` sends its action message to its target once if its view is clicked and it gets the mouse-down event, but can also send the action message continuously as long as the mouse is held down with the cursor inside the button cell. The button cell can show that it's being pressed by highlighting in several ways—for example, a bordered button cell can appear pushed into the screen, or the image or title can change to an alternate form while the button cell is pressed.

An NSButtonCell can also have a key equivalent (like a menu item). If the button cell is displayed in the key window, the button cell gets the first chance to receive events related to key equivalents. This feature is used quite often in

modal panels that have an “OK” button containing the image that represents the Return key. Usually an `NSButtonCell` displays a key equivalent as its image; if you ever set an image for the `NSButtonCell`, the key equivalent remains, but doesn’t get displayed. For more information on `NSButtonCell`’s behavior, see `NSButton` and `NSMatrix`.

Exceptions

In its implementation of the `compare:` method (declared in `NSCell`), `NSButtonCell` raises `NSBadComparisonException` if the `otherCell` argument is not of the `NSButtonCell` class.

Method Types

Activity	Class Method
Setting the titles	– alternateTitle – setAlternateTitle: – setFont: – setTitle: – title
Setting the images	– alternateImage – imagePosition – setAlternateImage: – setImagePosition:
Setting the repeat interval	– getPeriodicDelay:interval: – setPeriodicDelay:interval:
Setting the key equivalent	– keyEquivalent – keyEquivalentFont – keyEquivalentModifierMask – setKeyEquivalent: – setKeyEquivalentFont: – setKeyEquivalentFont:size: – setKeyEquivalentModifierMask:
Modifying graphic attributes	– isOpaque – isTransparent – setTransparent:
Modifying display behavior	– highlightsBy – setHighlightsBy: – showsStateBy – setShowsStateBy: – buttonType – setButtonType:
Simulating a click	– performClick:

Instance Methods

alternateImage

```
- (NSImage *)alternateImage
```

Returns the NSImage that appears on the NSButtonCell when it's in its alternate state, or nil if there is no alternate NSImage. This NSButtonCell only displays its alternate NSImage if it highlights or shows its alternate state by displaying its alternate contents. See also setAlternateImage:, setImagePosition:, setButtonType:.

alternateTitle

```
- (NSString *)alternateTitle
```

Returns an NSString containing the title that appears on the NSButtonCell when it's in its alternate state, or NULL if there isn't one. The alternate title is only displayed if the NSButtonCell highlights or shows its alternate state by displaying its alternate contents. See also setAlternateTitle:, title, setButtonType:.

buttonType

```
- (NSButtonType)buttonType
```

Returns the button cell's button type. See also setButtonType:.

getPeriodicDelay:interval:

```
- (void)getPeriodicDelay:(float *)delay interval:(float *)interval
```

Returns by reference the delay and interval periods for a continuous button cell. delay is the amount of time (in seconds) that a continuous button cell will pause before starting to periodically send action messages to the target object. interval is the amount of time (also in seconds) between those messages. See also setContinuous: (NSCell), setPeriodicDelay:interval:.

highlightsBy

```
- (int)highlightsBy
```

Returns the logical OR of flags that indicate the way the button cell highlights when it gets a mouse-down event. See setHighlightsBy: for the list of flags. See also setHighlightsBy:, showsStateBy, setShowsStateBy:.

imagePosition

```
- (NSCellImagePosition)imagePosition
```

Returns the position of the button cell's image (if any) relative to the cell's title. The possible return values are:

- NSNoImage
- NSImageOnly
- NSImageLeft
- NSImageRight
- NSImageBelow
- NSImageAbove
- NSImageOverlaps

See also setImagePosition:.

isOpaque

```
- (BOOL)isOpaque
```

Returns YES if the button cell draws over every pixel in its frame and NO if it doesn't. The button cell is opaque only if it is not transparent and if it has a border. See also isTransparent, setTransparent:.

isTransparent

```
- (BOOL)isTransparent
```

Returns YES if the NSButtonCell is transparent and NO if it isn't. A transparent NSButtonCell never draws anything, but it does receive mouse-down events and track the mouse properly. See also setTransparent:, isOpaque.

keyEquivalent

```
- (NSString *)keyEquivalent
```

Returns an NSString containing the key-equivalent character of the NSButtonCell. The default key equivalent is the empty string (@"").See also setKeyEquivalent:, setKeyEquivalentFont:, setKeyEquivalentFont:size:, keyEquivalentModifierMask.

keyEquivalentFont

```
- (NSFont *)keyEquivalentFont
```

Returns the NSFont used to draw the key equivalent. See also setKeyEquivalentFont:, keyEquivalent.

keyEquivalentModifierMask

```
- (unsigned int)keyEquivalentModifierMask
```

Returns the mask indicating the possible modifier keys for button cell's key equivalent. See also setKeyEquivalentModifierMask:.

performClick:

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

If this button cell is contained in an NSControl, then invoking this method causes the button cell to act as if the user has clicked it.

setAlternateImage:

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

Makes anImage the button cell's alternate image and redraws the cell if possible. A button cell displays its alternate NSImage only if it highlights or displays its alternate state by using its alternate contents. See also alternateImage, setImagePosition:, setButtonType:.

setAlternateTitle:

```
- (void)setAlternateTitle:(NSString *)aString
```

Makes a copy of aString and uses it as the button cell's alternate title. Doesn't display the button cell even if autodisplay is on in the button cell's NSView. The alternate title is shown only if the button cell changes its contents when highlighting or displaying its alternate state. See also alternateTitle, setTitle:, setButtonType:.

setFont:

```
- (void)setFont:(NSFont *)fontObject
```

Sets the NSFont used in displaying the title and alternate title. Does nothing if the cell has no title or alternate title. If the button cell has a key equivalent, its font is not changed, but the key equivalent's font size is changed to match the new title NSFont. See also setKeyEquivalentFont:, setKeyEquivalentFont:size:.

setHighlightsBy:

```
- (void)setHighlightsBy:(int)aType
```

Sets how the button cell highlights when pressed. aType can be the logical OR of one or more of the constants shown in Table 1-6:

Table 1-6 Showing an NSButtonCell's On State

Constant	Action
NSNoCellMask	The NSButtonCell doesn't change. This flag is ignored if any others are set in *aType*.
NSContentsCellMask	The NSButtonCell displays its alternate image and/or title.
NSPushInCellMask	The default case. The NSButtonCell "pushes in" when pressed if it has a border.
NSChangeGrayCellMask	The NSButtonCell swaps the light gray and white pixels on its background and image.
NSChangeBackgroundCellMask	Similar to NSChangeGrayCellMask except that only the background pixels are changed.

If both NSChangeGrayCellMask and NSChangeBackgroundCellMask are specified, both are recorded, but which behavior is used depends on the button cell's image. If there is no image, or the image has no alpha (tranparency) data, NSChangeGrayCellMask is used. If the image does have alpha data, NSChangeBackgroundCellMask is used; this behavior allows the gray/white swap of the background to show through the images's transparent pixels. See also highlightsBy, setShowsStateBy:, showsStateBy.

setImagePosition:

```
- (void)setImagePosition:(NSCellImagePosition)aPosition
```

Sets the position of the NSButtonCell's image in relation to its title in cases where the cell displays both at the same time. aPosition can be one of the constants shown in Table 1-7:

Table 1-7 Image Positions for an NSButtonCell

Constant	Meaning
NSNoImage	Title only. No image on the button.
NSImageOnly	Image only. No title on the button.
NSImageLeft	Image is to the left of the title.
NSImageRight	Image is to the right of the title.
NSImageBelow	Image is below the title.
NSImageAbove	Image is above the title.
NSImageOverlaps	Title is drawn on top of image.

If the position is above or below the title, the alignment of the title will be changed to NSCenterTextAlignment. This behavior can be overridden with a subsequent setAlignment: method. See also imagePosition, setAlignment: (NSActionCell).

setKeyEquivalent:

```
- (void)setKeyEquivalent:(NSString *)aKeyEquivalent
```

Sets the key equivalent character of the button cell; the default is the empty string (@""). Has the button cell redrawn if needed. The key equivalent isn't displayed if the icon position is set to NSNoImage, NSImageOnly, or NSImageOverlaps (see setImagePosition:). The key equivalent isn't displayed on a button cell that has an image. To make sure it gets displayed, set the image and alternate image to nil before using this method. See also keyEquivalent, setKeyEquivalentFont:, setKeyEquivalentFont:size:, performKeyEquivalent: (NSButton, NSMatrix)

setKeyEquivalentModifierMask:

```
- (void)setKeyEquivalentModifierMask:(unsigned int)mask
```

Sets the `mask` that determines the possible modifier keys for `NSButtonCell`'s key equivalent. Use the following enumeration constants in constructing `mask`:

- `NSAlphaShiftKeyMask`
- `NSShiftKeyMask`
- `NSControlKeyMask`
- `NSAlternateKeyMask`
- `NSCommandKeyMask`
- `NSNumericPadKeyMask`
- `NSHelpKeyMask`
- `NSFunctionKeyMask`

setKeyEquivalentFont:

```
- (void)setKeyEquivalentFont:(NSFont *)fontObj
```

Sets the `NSFont` used to draw the key equivalent, and has the button cell redrawn if possible. This method does nothing if there is already an image associated with this button cell. The default `NSFont` is the same as that used to draw the title. See also `keyEquivalentFont`, `setKeyEquivalentFont:size:`.

setKeyEquivalentFont:size:

```
- (void)setKeyEquivalentFont:(NSString *)fontName
    size:(float)fontSize
```

Sets by name and size the `NSFont` used to draw the key equivalent, and has the button cell redrawn if possible. Does nothing if there is already an image associated with this button cell. The default `NSFont` is the same as that used to draw the title. See also `setKeyEquivalentFont:`.

setKeyEquivalentModifierMask:

```
- (void)setKeyEquivalentModifierMask:(unsigned int)mask
```

Sets the `mask` that determines the possible modifier keys for `NSButtonCell`'s key equivalent. Use the following enumeration constants in constructing `mask`:

- NSAlphaShiftKeyMask
- NSShiftKeyMask
- NSControlKeyMask
- NSAlternateKeyMask
- NSCommandKeyMask
- NSNumericPadKeyMask
- NSHelpKeyMask
- NSFunctionKeyMask

setPeriodicDelay:interval:

```
- (void)setPeriodicDelay:(float)delay interval:(float)interval
```

Sets the message delay and interval for the button cell. These two values are used if the button cell has been set—by a `setContinuous:` message—to continuously send its action message to its target object while tracking the mouse. `delay` is the amount of time (in seconds) that a continuous button cell will pause before starting to periodically send action messages to the target object. `interval` is the amount of time (also in seconds) between those messages. The maximum value allowed for both `delay` and the `interval` is 60.0 seconds. See also `getPeriodicDelay:interval:`, `setContinuous:` (`NSCell`).

setShowsStateBy:

```
- (void)setShowsStateBy:(int)aType
```

Sets the way the button cell indicates its alternate (pressed) state. `aType` should be the logical OR of one or more of the constants shown in Table 1-6 on page 102. If both `NSChangeGrayCellMask` and `NSChangeBackgroundCellMask` are specified, both are recorded, but the actual behavior depends on the button cell's image. If there is no image, or if the image has no alpha (tranparency) data, `NSChangeGrayCellMask` is used. If the image exists and has alpha data, `NSChangeBackgroundCellMask` is used; this allows the gray / white swap of the background to show through the image's transparent pixels. See also `showsStateBy`, `setHighlightsBy:`, `highlightsBy`.

setTitle:

```
- (void)setTitle:(NSString *)aString
```

Makes a copy of `aString` and uses it as the button cell's title when the cell is in its normal state. This title is always shown on button cells that don't use their alternate contents when highighting or displaying their alternate state. See also `title`, `setAlternateTitle:`.

setTransparent:

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

Sets whether the button cell is transparent. A transparent button cell never draws, but does track the mouse and send its action normally. A transparent button cell is useful for sensitizing an area on the screen so that an action gets sent to a target when the area receives a mouse click. See also `isTransparent`, `isOpaque`.

setButtonType:

```
- (void)setButtonType:(NSButtonType)aType
```

Sets the way the button cell highlights while pressed, and how it shows its state. Redraws the button cell if possible. `aType` can be one of the following constants (as described in the `NSButton`'s `setButtonType:` method description):

- `NSMomentaryPushButton`
- `NSPushOnPushOffButton`
- `NSToggleButton`
- `NSSwitchButton`
- `NSRadioButton`
- `NSMomentaryChangeButton`
- `NSOnOffButton`
- `NSMomentaryLight`

See also `setButtonType:` (`NSButton`), `setHighlightsBy:`, `setShowsStateBy:`.

showsStateBy

```
- (int)showsStateBy
```

Returns the logical OR of flags that indicate the way the button cell shows its alternate (pressed) state. See Table 1-6 on page 102 for a list of possible flags. See also `setShowsStateBy:`, `highlightsBy`, `setHighlightsBy:`.

`title`

```
- (NSString *)title
```

Returns the title displayed on the button cell when it's in its normal state, or always if the button cell doesn't use its alternate contents for highlighting or displaying the alternate state. Returns `NULL` if there is no title. See also `setTitle:`.

NSCachedImageRep

Inherits From:	NSImageRep : NSObject
Conforms To:	NSCoding, NSCopying (NSImageRep) NSObject (NSObject)
Declared In:	AppKit/NSCachedImageRep.h

Class Description

`NSCachedImageRep`, a subclass of `NSImageRep`, defines an object that stores its source data as a rendered image in a window, typically a window that stays off screen. The only data that's available for reproducing the image is the image itself. An `NSCachedImageRep` differs from the other kinds of `NSImageRep`s defined in the Application Kit, all of which can reproduce an image from the information originally used to draw it. Instances of this class are generally used indirectly, through an `NSImage` object.

Method Types

Activity	Class Method
Initializing an NSCachedImageRep	– initWithSize:depth:separate:alpha: – initWithWindow:rect:
Getting the representation	– rect – window

Instance Methods

initWithSize:depth:separate:alpha:

```
– (id)initWithSize:(NSSize)aSize depth:(NSWindowDepth)aDepth
    separate:(BOOL)separate alpha:(BOOL)alpha
```

Initializes a new `NSCachedImageRep` for an image of the specified size and depth. The `separate` argument specifies whether the image will get its own unique cache, instead of possibly sharing one with other images. For best performance (although it's not essential), the `alpha` argument should be set according to whether the image will have a channel for transparency information. See also `initWithWindow:rect:`.

initWithWindow:rect:

```
– (id)initWithWindow:(NSWindow *)aWindow rect:(NSRect)aRect
```

Initializes the new `NSCachedImageRep` for an image to be drawn in the rectangle `aRect` of the specified window. This method retains `aWindow`. The rectangle is specified in `aWindow`'s base coordinate system, and the size of the image is set from the size of the rectangle. You must draw the image in the rectangle yourself; there are no `NSCachedImageRep` methods for this purpose. See also `initWithSize:depth:separate:alpha:`.

rect

```
– (NSRect)rect
```

Returns the rectangle where the image is cached. See also `window`.

window

```
- (NSWindow *)window
```

Returns the NSWindow where the image is cached. See also rect.

NSCell

Inherits From:	NSObject
Conforms To:	NSCoding, NSCopying, NSObject (NSObject)
Declared In:	AppKit/NSCell.h

Class Description

The NSCell class provides a mechanism for displaying text or images in an NSView without the overhead of a full NSView subclass. In particular, it provides much of the functionality of the NSText class by providing access to a shared NSText object used by all instances of NSCell in an application. NSCells are also extremely useful for placing titles or images at various locations in a custom subclass of NSView.

NSCell is used heavily by most of the NSControl classes to implement their internal workings. For example, NSSlider uses an NSSliderCell, NSTextField uses an NSTextFieldCell, and NSBrowser uses an NSBrowserCell. Sending a message to the NSControl is often simpler than dealing directly with the corresponding NSCell. For instance, NSControl objects typically invoke updateCell: (causing the cell to be displayed) after changing a cell attribute; whereas if you directly call the corresponding method of the NSCell, the NSCell might not automatically display itself again.

Some subclasses of NSControl (notably NSMatrix) allow multiple NSCells to be grouped and to act together in some cooperative manner. With an NSMatrix, a group of radio buttons can be implemented without needing an NSView for each button and without needing an NSText object for the text on each button.

The NSCell class provides primitives for displaying text or an image, editing text, formatting floating-point numbers, maintaining state, highlighting, and tracking the mouse. NSCell's method

`trackMouse:inRect:ofView:untilMouseUp:` supports the target object and action method used to implement controls. However, `NSCell` implements target/action features abstractly, deferring the details of implementation to subclasses of `NSActionCell`.

The `initImageCell:` method is the designated initializer for `NSCell`s that display images. The `initTextCell:` method is the designated initializer for `NSCell`s that display text. Override one or both of these methods if you implement a subclass of `NSCell` that performs its own initialization. If you need to use target and action behavior, you may prefer to subclass `NSActionCell`, which provides the default implementation of this behavior. For more information on how `NSCell` is used, see the `NSControl` class specification.

Method Types

Activity	Class Method
Initializing an NSCell	– initImageCell: – initTextCell:
Determining component sizes	– calcDrawInfo: – cellSize – cellSizeForBounds: – drawingRectForBounds: – imageRectForBounds: – titleRectForBounds:
Setting the NSCell's type	– setType: – type
Setting the NSCell's state	– setState: – state
Enabling and disabling the NSCell	– isEnabled – setEnabled:
Setting the image	– image – setImage:

Activity	Class Method
Setting the NSCell's value	– doubleValue – floatValue – intValue – objectValue – stringValue – setDoubleValue: – setFloatValue: – setIntValue: – setObjectValue: – setStringValue:
Interacting with other NSCells	– takeDoubleValueFrom: – takeFloatValueFrom: – takeIntValueFrom: – takeObjectValueFrom: – takeStringValueFrom:
Modifying text attributes	– alignment – isEditable – isSelectable – isScrollable – setAlignment: – setEditable: – setFont: – setSelectable: – setScrollable: – setUpFieldEditorAttributes: – setWraps: – wraps
Editing text	– editWithFrame:inView:editor:delegate:event: – endEditing: – selectWithFrame:inView:editor:delegate: start:length:
Validating input	– entryType – hasValidObjectValue – isEntryAcceptable: – setEntryType:
Formatting data	– formatter – setFloatingPointFormat:left:right: – setFormatter:

Activity	Class Method
Modifying graphic attributes	– isBezeled – isBordered – isOpaque – setBezeled: – setBordered:
Setting parameters	– cellAttribute: – setCellAttribute:to:
Displaying	– controlView – drawInteriorWithFrame:inView: – drawWithFrame:inView: – highlight:withFrame:inView: – isHighlighted
Target and action	– action – isContinuous – sendActionOn: – setAction: – setContinuous: – setTarget: – target
Assigning a tag	– setTag: – tag
Handling keyboard alternatives	– keyEquivalent
Tracking the mouse	+ prefersTrackingUntilMouseUp – continueTracking:at:inView: – mouseDownFlags – getPeriodicDelay:interval: – startTrackingAt:inView: – stopTracking:at:inView:mouseIsUp: – trackMouse:inRect:ofView:untilMouseUp:
Managing the cursor	– resetCursorRect:inView:
Comparing to another NSCell	– compare:
Using the NSCell to represent an object	– representedObject – setRepresentedObject:

Class Methods

prefersTrackingUntilMouseUp

```
+ (BOOL)prefersTrackingUntilMouseUp
```

Normally, returns NO so that tracking stops when the mouse leaves the NSCell. Subclasses may override this method to return YES if the cell's NSView should allow it, after a mouse-down event, to track mouse-dragged and mouse-up events even if they occur outside the NSCell's frame. For example, this method is overridden by NSSliderCell to ensure that an NSSliderCell in a NSMatrix doesn't stop responding to user input (and its neighbor start responding) just because its knob isn't dragged in a perfectly straight line. See also trackMouse:inRect:ofView:untilMouseUp:.

Instance Methods

action

```
- (SEL)action
```

Returns a null selector. This method is overridden by NSActionCell and subclasses that actually implement a target object and action method. See also setAction:, target.

alignment

```
- (NSTextAlignment)alignment
```

Returns the alignment of text in the cell. The return value can be one of the following constants:

- NSLeftTextAlignment
- NSRightTextAlignment
- NSCenterTextAlignment
- NSJustifiedTextAlignment
- NSNaturalTextAlignment

See also setAlignment:.

calcDrawInfo:

```
- (void)calcDrawInfo:(NSRect)aRect
```

This method is implemented by subclasses of `NSCell` to recalculate drawing sizes. Objects using `NSCell`s generally maintain a flag that informs them if any of their `NSCell`s have been modified in such a way that the location or size of the cell requires recomputing. If so, `calcSize` is automatically invoked before displaying the cell; that method invokes `NSCell`'s `calcDrawInfo:` for each cell. See also `calcSize` (`NSControl`).

cellAttribute:

```
- (int)cellAttribute:(NSCellAttribute)aParameter
```

Returns the value of one of the frequently accessed flags for a cell. See `setCellAttribute:to:` for a list of the parameters and corresponding methods. Since the parameters are also accessible through methods such as `isEnabled` and `isHighlighted`, you shouldn't need to use this method often. See the Application Kit's Types and Constants chapter for a description of cell attributes. See also `setCellAttribute:to:`.

cellSize

```
- (NSSize)cellSize
```

Returns the minimum width and height required for displaying the cell. This method invokes `cellSizeForBounds:` with the rectangle argument set to a rectangle with very large width and height. Override this method if that isn't the proper way to calculate the minimum width and height required for displaying the cell. See also `cellSizeForBounds:`.

cellSizeForBounds:

```
- (NSSize)cellSizeForBounds:(NSRect)aRect
```

Returns the minimum width and height required for displaying the cell in the given rectangle. If it's not possible to fit the cell, the width and/or height could be bigger than the ones of the provided rectangle. The computation is done by trying to size the cell so that it fits in the rectangle argument (for example, by wrapping the text). If a choice must be made between extending the width or height of `aRect` to fit text, the height will be extended. See also `cellSize`.

compare:

```
- (NSComparisonResult)compare:(id)otherCell
```

Compares the string values of this cell and `otherCell` (which must be a kind of `NSCell`). Raises `NSBadComparisonException` if `otherCell` is not of the `NSCell` class.

Note – The reason this method takes an `id` argument instead of an `NSCell *` is so that within the `compare:` method one can invoke methods such as `title` that are valid for subclasses of `NSCell` but not for `NSCell` (without generating a compiler warning).

---

continueTracking:at:inView:

```
- (BOOL)continueTracking:(NSPoint)lastPoint
    at:(NSPoint)currentPoint inView:(NSView *)controlView
```

Determines whether or not the cell should keep tracking the mouse based on the positions provided. Returns `YES` if it can keep tracking and `NO` if should not. This method is invoked by `trackMouse:inRect:ofView:untilMouseUp:` as the mouse is dragged around inside the cell. `lastPoint` and `currentPoint` should be in `controlView`'s coordinate system. By default, this method returns `YES` when the cell is continuous, that is, when it should continually send action messages while the mouse is pressed or dragged. This method is often overridden to provide more sophisticated tracking behavior. See also `trackMouse:inRect:ofView:untilMouseUp:`, `startTrackingAt:inView:`, `stopTracking:at:inView:mouseIsUp:`.

controlView

```
- (NSView *)controlView
```

Returns `nil`. This method is implemented abstractly, since `NSCell` doesn't record the `NSView` in which it's drawn. This method is overridden by `NSActionCell` and its subclasses to return the `NSView` last drawn in (normally an `NSControl`). An action cell uses the control `NSView` as the only argument in an action message when it's sent to the target. See also `controlView` (`NSActionCell`), `drawWithFrame:inView:`, `drawInteriorWithFrame:inView:`.

doubleValue

```
- (double)doubleValue
```

Returns the receiving text NSCell's value as a double-precision floating-point number, by converting its string contents to a double using the standard C function atof(). Returns 0.0 if the cell isn't a text NSCell. See also setDoubleValue:, floatValue, intValue, stringValue, type.

drawInteriorWithFrame:inView:

```
- (void)drawInteriorWithFrame:(NSRect)cellFrame
    inView:(NSView *)controlView
```

Draws the area within the cell's border in controlView. For the base NSCell class, it's the same as drawWithFrame:inView: except that it doesn't draw the bezel or border if there is one. cellFrame should be the frame of the cell, that is, the same as the cellFrame passed to drawWithFrame:inView:, *not* the rectangle returned by drawingRectForBounds:. The PostScript focus must be locked on controlView when this method is invoked. If the cell's highlight flag is YES, then the cell is highlighted.

drawInteriorWithFrame:inView: is usually invoked from the NSControl class's drawCellInside: method and is used to cause minimal drawing to be done in order to update the value displayed by the cell when the contents are changed. This becomes more important in more complex cells such as NSButtonCell and NSSliderCell. All NSCell subclasses that override drawWithFrame:inView: *must* override drawInteriorWithFrame:inView:. drawInteriorWithFrame:inView: should never invoke drawWithFrame:inView:, but drawWithFrame:inView: can—and often does—invoke drawInteriorWithFrame:inView:. See also drawWithFrame:inView:, lockFocus (NSView), highlight:withFrame:inView:, isHighlighted, compositerect (Display PostScript operator).

drawWithFrame:inView:

```
- (void)drawWithFrame:(NSRect)cellFrame
    inView:(NSView *)controlView
```

Displays the contents of a cell in a given rectangle of a given view. Your code must lock the focus on `controlView` before invoking this method. It draws the border or bezel (if any), then invokes `drawInteriorWithFrame:inView:`. A text cell displays its text in the rectangle by using a global `NSText` object. An image `NSCell` displays its image centered in the rectangle if it fits in the rectangle, or by setting the image origin on the rectangle origin if it doesn't fit. Nothing is displayed for an `NSCell` of type `NSNullCellType`. Override this method if you want a display that is specific to your own `NSCell` subclass. See also `drawInteriorWithFrame:inView:`, `lockFocus` (`NSView`).

drawingRectForBounds:

```
- (NSRect)drawingRectForBounds:(NSRect)theRect
```

Given the bounds of a cell in `theRect`, this method returns the rectangle into which the cell draws its interior. The interior is everything but a bezel or border. In other words, this method calculates the rectangle which is touched by `drawInteriorWithFrame:inView:`. However, your code should *not* use the rectangle returned by this method as the argument to `drawInteriorWithFrame:inView:`. See also `imageRectForBounds:`, `titleRectForBounds:`, `drawInteriorWithFrame:inView:`.

editWithFrame:inView:editor:delegate:event:

```
- (void)editWithFrame:(NSRect)aRect inView:(NSView *)controlView
    editor:(NSText *)textObject delegate:(id)anObject
    event:(NSEvent *)theEvent
```

Begins editing a cell's text by using the `NSText` object `textObject` in response to an `NSLeftMouseDown` or `NSRightMouseDown` event. `aRect` must be the one you have used when displaying the cell. `theEvent` is the mouse-down event. `anObject` is made the delegate of the `NSText` object `textObject` used for the editing: it will receive `NSText` delegate messages (such as `textDidEndEditing:`, `textWillEnd`, `textDidResize`, `textWillResize`), and others sent by the `NSText` object while editing. If the receiver isn't a text `NSCell`, no editing is performed; otherwise the `NSText` object is sized to `aRect` and its superview is set to `controlView`, so that it exactly covers the cell. Then it's activated and editing begins. It's the responsibility of the delegate to end the editing, remove any data from `textObject`, and invoke `endEditing:` on the cell in the `textDidEditing:`

method. See also endEditing:, selectWithFrame:inView:editor:delegate: start:length:, NSText (Methods Implemented by the Delegate)

endEditing:

```
- (void)endEditing:(NSText *)textObject
```

Ends editing begun with editWithFrame:inView:editor:delegate:event: or selectWithFrame:inView:editor:delegate:start:length:. Usually this method is invoked by the textDidEndEditing: method of the object you are using as the delegate for the NSText object (most often an NSMatrix or NSTextField). This method should remove the NSText object from the view hierarchy and set its delegate to nil. See also editWithFrame:inView:editor:delegate:event:, selectWithFrame:inView:editor:delegate: start:length:, textDidEndEditing: (NSText class delegate method).

entryType

```
- (int)entryType
```

Returns the type of data the user can type into the cell. The possible return values are the following values:

- NSAnyType
- NSIntType
- NSPositiveIntType
- NSFloatType
- NSPositiveFloatType
- NSDoubleType
- NSPositiveDoubleType

See also setEntryType:.

floatValue

```
- (float)floatValue
```

Returns the receiving text cell's value as a single-precision floating point number, by converting its string contents to a `double` using the C function `atof()` and then casting the result to a `float`. Returns 0.0 if the receiver isn't a text cell. See also `setFloatValue:`, `doubleValue`, `intValue`, `stringValue`, `type`.

font

```
- (NSFont *)font
```

Returns the font used to display cell text . Returns `nil` if the receiver isn't a text cell. See also `setFont:`, `type`, `NSFont`.

formatter

```
- (id)formatter
```

Returns the cell's formatter, or returns nil if the cell has no associated formatter. This method is not part of the OpenStep specification. See also `setFormatter:`.

getPeriodicDelay:interval:

```
- (void)getPeriodicDelay:(float *)delay interval:(float *)interval
```

Returns by reference two values: the amount of time (in seconds) that a continuous button will pause before starting to periodically send action messages to the target object, and the interval (also in seconds) at which those messages are sent. Periodic messaging behavior is controlled by `NSCell`'s `sendActionOn:` and `setContinuous:` methods. (By default, a cell sends the action message only on mouse-up events.) Override this method to return your own values. See also `setContinuous:`, `sendActionOn:`.

hasValidObjectValue

```
- (BOOL)hasValidObjectValue
```

Returns `YES` if the cell has a valid content object, and returns `NO` otherwise. This method is not part of the OpenStep specification. See also `objectValue`.

highlight:withFrame:inView:

```
- (void)highlight:(BOOL)lit withFrame:(NSRect)cellFrame
    inView:(NSView *)controlView
```

If the cell's highlight status is different from `lit`, then the cell's highlight status is set to `lit` and, if `lit` is `YES`, the rectangle `cellFrame` in `controlView` is highlighted. Your code must lock focus on `controlView` before invoking this method. This method composites with `NSCompositeHighlight` inside the bounds of `cellFrame`. Override this method if you want more sophisticated highlighting behavior in an `NSCell` subclass.

Note – `NSCell` class highlighting does *not* appear when printed (although subclasses like `NSTextFieldCell` and `NSButtonCell` can print themselves highlighted). This is true because the base `NSCell` class is transparent, and there is no concept of transparency in printed output.

See also `isHighlighted`, `drawWithFrame:inView:`, `drawInteriorWithFrame:inView:`.

image

```
- (NSImage *)image
```

Returns the cell's image, if any, or `NULL` if the receiver isn't an image cell. See also `setImage:`.

imageRectForBounds:

```
- (NSRect)imageRectForBounds:(NSRect)theRect
```

Given the bounds of the cell in `theRect`, this method returns the rectangle into which the cell draws its icon. Your code should *not* use the rectangle returned by this method as the argument to `drawInteriorWithFrame:inView:`. See also `drawingRectForBounds:`, `titleRectForBounds:`, `drawInteriorWithFrame:inView:`.

initImageCell:

```
- (id)initImageCell:(NSImage *)anImage
```

Initializes and returns the receiver, a new image cell instance,that is, its type is NSImageCellType. The image is set to an NSImage with the name anImage. If anImage is NULL or an image for anImage is not found, the cell will be initialized with a default icon, "NSsquare16". This method is the designated initializer for NSCells that display an image. If the cell later has text assigned, its type will automatically change. See also image, setImage:, initTextCell:, imageNamed: (NSImage), name (NSImage).

initTextCell:

```
- (id)initTextCell:(NSString *)aString
```

Initializes and returns the receiver, a new text cell instance, that is, its type is NSTextCellType. The cell's title is set to aString, or "Cell" if aString is NULL. This method is the designated initializer for text NSCells. See also initImageCell:, setImage:.

intValue

```
- (int)intValue
```

Returns the receiving text cell's value as an integer by converting its string contents to an int using the C function atoi(). Returns 0 if the receiver isn't a text NSCell. See also setIntValue:, doubleValue, floatValue, stringValue, type.

isBezeled

```
- (BOOL)isBezeled
```

Returns YES if the cell draws itself with a bezeled border and NO otherwise. The default is NO. See also setBezeled:, isBordered.

isBordered

```
- (BOOL)isBordered
```

Returns YES if the cell draws itself surrounded by a one-pixel-wide black frame and NO otherwise. The default is NO. See also setBordered:, isBezeled.

isContinuous

```
- (BOOL)isContinuous
```

Returns YES if the cell continuously sends its action message to the target object when tracking. This usually has meaning only for NSCell subclasses that implement instance variables and methods for target/action functionality, such as NSActionCell; certain NSControl subclasses, specifically NSMatrix, send a default action to a default target even if the NSCell doesn't have a target and action. See also setContinuous:, target, action.

isEditable

```
- (BOOL)isEditable
```

Returns YES if text in the cell is editable (and therefore also selectable) and NO otherwise. The default is NO. See also setEditable:, isSelectable.

isEnabled

```
- (BOOL)isEnabled
```

Returns YES if the cell is enabled and NO otherwise. The default is YES. A cell's enabled status is used primarily in event handling and display: it affects the behavior of methods for mouse tracking and text editing, by allowing or disallowing changes to the cell within those methods, and only allows the cell to highlight or set a cursor rectangle if it's enabled. You can still affect many cell attributes programmatically (setState:, for example, will still work). See also setEnabled:, trackMouse:inRect:ofView:untilMouseUp:.

isEntryAcceptable:

```
- (BOOL)isEntryAcceptable:(NSString *)aString
```

Tests whether aString matches the cell's entry type, as set by the setEntryType: method. Returns YES if aString is acceptable for the entry type and NO otherwise. For example, a text NSCell of type NSIntType accepts strings that represent integers, but not floating point numbers or words. If aString is NULL or empty, this method returns YES. This method is invoked by NSForm, NSMatrix, and other NSControls to see if a new text

string is acceptable for an NSCell. This method doesn't check for overflow. It can be overridden to enforce specific restrictions on what the user can type into the NSCell. See also setEntryType:.

isHighlighted

```
- (BOOL)isHighlighted
```

Returns YES if the cell is highlighted and NO otherwise. See also highlight:withFrame:inView:.

isOpaque

```
- (BOOL)isOpaque
```

Returns YES if the cell is opaque, that is, if it draws over every pixel in its frame, and NO otherwise. The base NSCell class is opaque if, and only if, it draws a bezel. Subclasses that draw differently should override this method based on how they perform their drawing. See also setBezeled:.

isScrollable

```
- (BOOL)isScrollable
```

Returns YES if typing past an end of the cell text will cause the cell to scroll to follow the typing. The default return value is NO. See also setScrollable:.

isSelectable

```
- (BOOL)isSelectable
```

Returns YES if the cell text is selectable and NO otherwise. The default is NO. See also setSelectable:, isEditable.

keyEquivalent

```
- (NSString *)keyEquivalent
```

Returns 0 because NSCell provides no support for key equivalents. Subclasses can implement key equivalents, and can override this method to return the key equivalent for the receiver. See also setKeyEquivalent: (NSButtonCell), keyEquivalent (NSButtonCell).

mouseDownFlags

```
- (int)mouseDownFlags
```

Returns the event flags (for example, `NSShiftKeyMask`) that were set when the mouse went down to start the current tracking session. This method is only valid during tracking. It doesn't work if the `NSCell` target initiates another `NSCell` tracking loop as part of its action method. See also `sendActionOn:`.

objectValue

```
(id <NSCopying>)objectValue
```

Returns the cell's content object. This method is not part of the OpenStep specification. See also `setObjectValue:`.

representedObject

```
- (id)representedObject
```

Returns the object that the receiver represents, if any. See also `setRepresentedObject:`.

resetCursorRect:inView:

```
- (void)resetCursorRect:(NSRect)cellFrame
    inView:(NSView *)controlView
```

If the receiver is a text cell, then a cursor rectangle is added to `controlView` (with `NSView`'s `addCursorRect:cursor:`). This allows the cursor to change to an I-beam when it passes over the cell. Override this method to change the cursor for an image cell, or to provide a different cursor for a text cell.

selectWithFrame:inView:editor:delegate: start:length:

```
- (void)selectWithFrame:(NSRect)aRect inView:(NSView *)controlView
    editor:(NSText *)textObject delegate:(id)anObject
    start:(int)selStart length:(int)selLength
```

Uses `textObject` to select text in the cell identified by `selStart` and `selLength`, which will be highlighted and selected as though the user had dragged the cursor over it. This method is similar to `editWithFrame:inView:editor:delegate:event:`, except that it can be invoked in any situation, not only on a mouse-down event.

sendActionOn:

```
- (int)sendActionOn:(int)mask
```

Sets flags to determine when an action is sent to the target while tracking. These can be any logical combination of:

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

This method returns an event mask built from the old flags. See also `setContinuous:`.

setAction:

```
- (void)setAction:(SEL)aSelector
```

Does nothing. This method is overridden by `NSActionCell` and its subclasses, which actually implement the target object and action method. It is also overriden by `NSBrowserCell` to provide access to its `NSBrowser`'s action method. See also `action`, `setTarget:`.

setAlignment:

```
- (void)setAlignment:(NSTextAlignment)mode
```

Sets the cell text alignment to `mode`. `mode` should be one of these constants:

- `NSLeftTextAlignment`
- `NSRightTextAlignment`
- `NSCenterTextAlignment`
- `NSJustifiedTextAlignment`
- `NSNaturalTextAlignment`

See also `alignment`, `setWraps:`.

setBezeled:

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

If `flag` is `YES`, the cell draws itself surrounded by a bezel; if `NO`, it doesn't. `setBordered:` and `setBezeled:` are mutually exclusive. See also `isBezeled`, `setBordered:`.

setBordered:

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

If `flag` is `YES`, the cell draws itself surrounded by a one-pixel-wide black frame; if `NO`, it doesn't. `setBordered:` and `setBezeled:` are mutually exclusive. See also `isBordered`, `setBezeled:`.

setCellAttribute:to:

```
- (void)setCellAttribute:(NSCellAttribute)aParameter to:(int)value
```

Sets the value of one of the cell's parameters to `value`. You don't normally use this method, since these parameters can be set using specific methods such as `setEditable:`. In this method, the parameters is identified by `aParameter`, a

symbolic constant defined in the header file NSCell.h. The following table lists these constants with the corresponding methods for setting and getting the value of the related parameters:

Table 1-8 NSCell Attribute Constants

Parameter Constant	Equivalent Methods
NSCellDisabled	setEnabled:, isEnabled
NSCellHighlighted	highlight:withFrame:inView:, isHighlighted
NSCellState	setState:, state
NSCellEditable	setEditable:, isEditable

Use of this method is discouraged as it could produce unpredictable results in subclasses. It's much safer to invoke the appropriate parameters-specific method. See also cellAttribute:.

setContinuous:

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

Determines whether a cell continuously sends its action message to the target object when tracking. Normally, this method will set the continuous flag or a mouse-dragged flag, depending on which setting is appropriate to the subclass implementing it. In the base NSCell class, this method sets the continuous flag. These settings usually have meaning only for NSActionCell and its subclasses that implement the instance variables and methods that provide target-action functionality. Some NSControl subclasses, specifically NSMatrix, send a default action to a default target when an NSCell doesn't provide a target or action. See also isContinuous, sendActionOn:.

setDoubleValue:

```
- (void)setDoubleValue:(double)aDouble
```

Sets the cell contents to the string value representing the double-precision floating-point number aDouble, ignoring the cell entry type. Does nothing if the receiver isn't a text cell. See also doubleValue, setFloatValue:, setIntValue:, setStringValue:, entryType, type.

setEditable:

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

If `flag` is `YES`, then the cell's text is made both editable and selectable. If `flag` is `NO`, and the text was not selectable before editing was last enabled, that is, before this message was last sent with an argument of `YES`, then the text is returned to not being selectable. See also `isEditable`, `setSelectable:`, `editWithFrame:inView:editor:delegate:event:`.

setEnabled:

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

Sets the cell's enabled status. A cell's enabled status is used primarily in event handling and display. It affects the behavior of methods for mouse tracking and text editing, by allowing or disallowing changes to the cell within those methods, and only allows the cell to highlight or set a cursor rectangle if it's enabled. Many cell attributes can still be altered programmatically (`setState:`, for example, will still work). See also `isEnabled`.

setEntryType:

```
- (void)setEntryType:(int)aType
```

Sets the type of data the user can type into the cell. `aType` can be any of the eight constants shown in the following table:

Table 1-9 Numeric Data Types for an NSCell

Constant	Allowable Numeric String Value
NSAnyType	No restrictions
NSIntType	Integer values
NSPositiveIntType	Positive integer values
NSFloatType	Single-precision floating point values
NSPositiveFloatType	Positive single-precision floating point values
NSDoubleType	Double-precision floating point values
NSPositiveDoubleType	Positive double-precision floating point values

If the receiver isn't a text cell, it's converted to type NSTextCellType, in which case its font is set to the user's system font at 12.0 point, and its string value is set to "Cell" (even for text cells that display numbers). The entry type is checked by the isEntryAcceptable: method. That method is used by NSControls that contain editable text (such as NSMatrix and NSTextField) to validate that what the user has typed is correct. If you want to have a custom NSCell accept some specific type of data (other than those listed above), override the isEntryAcceptable: method to check for the validity of the data the user has entered. See also entryType, isEntryAcceptable:, setFloatingPointFormat:left:right:.

setFloatingPointFormat:left:right:

```
- (void)setFloatingPointFormat:(BOOL)autoRange
    left:(unsigned int)leftDigits right:(unsigned int)rightDigits
```

Sets whether floating-point numbers are autoranged, and sets the sizes of the fields to the left and right of the decimal point. leftDigits specifies the maximum number of digits to the left of the decimal point, and rightDigits specifies the number of digits to the right (the fractional digit places will be padded with zeros to fill this width). However, if a number is too large to fit its integer part in leftDigits digits, as many places as are needed on the left are effectively removed from rightDigits when the number is displayed.

If autoRange is YES, leftDigits and rightDigits are simply added to form a maximum total field width for the cell (plus 1 for the decimal point). The fractional part will be padded with zeros on the right to fill this width, or truncated as much as possible (up to removing the decimal point and displaying the number as an integer). The integer portion of a number is never truncated—that is, it is displayed in full no matter what the field width limit is. leftDigits must be between 0 and 10. rightDigits must be between 0 and 14. If leftDigits is 0, then the default printf() formatting applies. If rightDigits is 0, then the decimal and the fractional part of the floating-point number are truncated, that is, the floating-point number is printed as if it were an integer. If the cell entry type isn't already NSFloatType, NSPositiveFloatType, NSDoubleType, or NSPositiveDoubleType, it's set to NSFloatType. See also setEntryType:.

setFloatValue:

```
- (void)setFloatValue:(float)aFloat
```

Sets the cell's contents to the string value representing the single-precision floating point number `aFloat`, ignoring the cell entry type. Does nothing if the receiver isn't a text cell. See also `floatValue`, `setDoubleValue:`, `setIntValue:`, `setStringValue:`, `entryType`, `type`.

setFont:

```
- (void)setFont:(NSFont *)fontObject
```

Sets the cont used to display text in the cell to `fontObject`. Does nothing if the receiver isn't a text cell. See also `font`.

setFormatter:

```
(void)setFormatter:(NSFormatter *)newFormatter
```

Sets the cells `NSFormatter`-derived object to `newFormatter`, replacing the old formatter if one existed. This method is not part of the OpenStep specification. See also `formatter`.

setImage:

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

Sets the cell's icon to `anImage` (an `NSImage` object with that name). `anImage` is stored as the `NSCell`'s contents, and the `NSImage` is stored as its support. If the cell isn't an image cell, it's converted; if the cell was a text cell, the text string is freed if necessary. If `anImage` is `NULL` or an empty string, or if an image can't be found for `anImage`, the cell has its image set to the standard system bitmap "`NSsquare16`". If you specify a name for which an image can't be found, no change is made. Your code can verify that the image was properly changed by comparing the values returned by the `type` or `image` methods before and after invoking `setImage:`. See also `image`, `imageNamed:` (`NSImage`), `initImageCell:`.

setIntValue:

```
- (void)setIntValue:(int)anInt
```

Sets the cell contents to the string value representing the integer `anInt`. Does nothing if the receiver isn't a text cell. This method ignores the entry type of the cell. See also `intValue`, `setDoubleValue:`, `setStringValue:`, `type`, `entryType`.

setObjectValue:

```
(void)setObjectValue:(id <NSCopying>)obj
```

Sets the cell's content object. This method is not part of the OpenStep specification. See also `objectValue`.

setRepresentedObject:

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

Creates an association between the receiver and `anObject`. `anObject` will be retained, released, archived, and unarchived whenever the receiver is. If another cell is already associated with `anObject`, that association is broken, and the receiver is associated with the object.

setScrollable:

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

Sets whether the cell scrolls to follow typing while being edited. See also `isScrollable`, `editWithFrame:inView:editor:delegate:event:`.

setSelectable:

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

If `flag` is `YES`, then the cell's text is made selectable but not `editable`. If `NO`, then the text is static (neither editable nor selectable). To make text in a cell both selectable and editable, send it a `setEditable:` message. See also `isSelectable`, `isEditable`, `editWithFrame:inView:editor:delegate:event:`.

setStringValue:

```
- (void)setStringValue:(NSString *)aString
```

Sets the cell's value to a copy of `aString`. If the receiver isn't a text cell, this method converts it to that type, setting its font to the user's system font at 12.0 points. If the receiver was an image cell, the `NSImage` for that image is *not* freed; your code should retrieve it beforehand and free it after sending this message. If floating point formatting has been set (with `setFloatingPointParameters:left:right:`) and the cell entry type is a floating point number type, then the string is tested to determine whether it represents a floating point number; if so, the string is displayed according to that floating point format. See also `stringValue`, `setDoubleValue:`, `setIntValue:`, `setFloatingPointFormat:left:right:`.

setState:

```
- (void)setState:(int)value
```

Sets the cell state to 0, if `value` is 0, and sets it to 1 otherwise. See also `state`.

setTag:

```
- (void)setTag:(int)anInt
```

Does nothing. This method is overridden by `NSActionCell` and its subclasses to support `NSControl`s with multiple `NSCell`s (`NSMatrix` and `NSForm`). Override this method to provide a way to identify `NSCell`s. See also `tag`, `cellWithTag:` (`NSMatrix`, `NSMenu`).

setTarget:

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

Does nothing. This method is one of several overridden by `NSActionCell` and its subclasses to implement target/action functionality. The method enables these subclasses to set their target objects. See also `setAction:`, `target`, `action`.

setType:

```
- (void)setType:(NSCellType)aType
```

Sets the cell's type to `aType`. `aType` should be one of the following values:

- `NSNullCellType`

- NSTextCellType
- NSImageCellType

If aType is NSTextCellType and the receiver isn't currently a text cell, then the font is set to the user's system font in 12.0 point; its string value is set to "Cell". If aType is NSImageCellType and the receiver isn't an image cell, then the image is set to the default, "NSsquare16". See also type, initImageCell:, initTextCell:, setImage:.

setUpFieldEditorAttributes:

```
- (NSText *)setUpFieldEditorAttributes:(NSText *)textObject
```

This method is invoked just before any drawing or editing occurs in the cell. This method is intended to be overridden. If you do override it, you must include this line first:

```
[super setUpFieldEditorAttributes:textObject];
```

If you don't, you risk inheriting drawing attributes from the last cell that drew any text. You should invoke only the setBackgroundColor: and setTextColor: NSText instance methods. Don't set any other parameters in the NSText object. This method normally returns textObject. If you want to substitute some other NSText object to draw with (but not edit, since editing always uses the window's field editor), you can return that object instead of textObject, and it will be used for the draw that caused setUpFieldEditorAttributes: to be invoked.

NSTextFieldCell, a subclass of NSActionCell, allows you to set the colors without creating your own NSCell subclass. You only need to subclass NSCell to control the color values if you don't want all the functionality (and instance variable usage) of an NSActionCell.

Note that most other text object attributes can be set with NSCell methods (setFont:, setAlignment:, setWraps:), so you need only override this method if you need to set the color values.

setWraps:

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

Sets whether the cell's text is word-wrapped. If `flag` is `YES`, text will be wrapped to word breaks. If `flag` is `NO`, the text will be truncated. The default is `YES`. This setting has effect only when displaying text, not when editing, and only applies to cells whose alignment is `NSLeftTextAlignment` (centered and right-aligned text always wraps to word breaks). See also `setAlignment:`.

startTrackingAt:inView:

```
- (BOOL)startTrackingAt:(NSPoint)startPoint
    inView:(NSView *)controlView
```

This method is invoked from `trackMouse:inRect:ofView:untilMouseUp:` the first time the mouse appears in the cell needing to be tracked. Override to provide implementation-specific tracking behavior. This method should return `YES` if it's okay to track based on this starting point, and *only* if the cell is continuous; otherwise, it should return `NO`. See also `trackMouse:inRect:ofView:untilMouseUp:`, `continueTracking:at:inView:`, `stopTracking:at:inView:mouseIsUp:`, `mouseDownFlags`.

state

```
- (int)state
```

Returns the cell state (0 or 1). The default is 0. See also `setState:`.

stopTracking:at:inView:mouseIsUp:

```
- (void)stopTracking:(NSPoint)lastPoint at:(NSPoint)stopPoint
    mouseIsUp:(BOOL)flag
```

Allows the cell to update itself to end tracking, based on `lastPoint` and `stopPoint`. Invoked from `trackMouse:inRect:ofView:untilMouseUp:` when the mouse has left the cell bound, or the mouse button has gone up. `flag` is `YES` if the mouse button went up to cause this method to be invoked. The default behavior is to do nothing. This method is often overridden to provide more sophisticated tracking behavior. See also `trackMouse:inRect:ofView:untilMouseUp:`, `continueTracking:at:inView:`.

stringValue

```
- (NSString *)stringValue
```

Returns the cell's value as a string. See also `setStringValue:`, `doubleValue`, `floatValue`, `intValue`.

tag

```
- (int)tag
```

Returns –1. This method is overridden by `NSActionCell` and its subclasses to support multiple-cell controls (`NSMatrix` and `NSForm`). Override this method if you want to use tags to identifiy cells. See also `setTag:`, `cellWithTag:` (`NSMatrix`, `NSMenu`).

takeDoubleValueFrom:

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

Sets the cell's double-precision floating point value to the value returned by `sender`'s `doubleValue` method. `sender` must be of a class that implements the `doubleValue` method. This method can be used in action messages between cells. It permits one cell (the sender) to affect the value of another cell (the receiver). For example, an `NSTextFieldCell` can be made the target of an `NSSliderCell`, which will send it a `takeDoubleValueFrom:` action message. The `NSTextFieldCell` will get the return value of the `NSSliderCell`'s `doubleValue` method, turn it into a text string, and display it. See also `takeDoubleValueFrom:` (`NSControl`), `setDoubleValue:`.

takeFloatValueFrom:

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

Sets the cell's single-precision floating-point value to the value returned by `sender`'s `floatValue` method. `sender` must be of a class that implements the `floatValue` method. This method is similar to `takeDoubleValueFrom:` except that it works with floats rather than doubles. See also `takeFloatValueFrom:` (`NSControl`).

takeIntValueFrom:

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

Sets the cell's integer value to the value returned by `sender`'s `intValue` method. `sender` must be of a class that implements the `intValue` method. This method is similar to `takeDoubleValueFrom:` except that it works with ints rather than doubles. See also `takeIntValueFrom:` (`NSControl`), `setIntValue:`.

takeObjectValueFrom:

```
(void)takeObjectValueFrom:(id)sender
```

Sets the cell's content object to the content object of `sender`. This method is not part of the OpenStep specification.

takeStringValueFrom:

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

Sets the cell's string value to the value returned by `sender`'s `stringValue` method. `sender` must be of a class that implements the `stringValue` method. This method is similar to `takeDoubleValueFrom:` except that it works with strings rather than doubles. See also `takeStringValueFrom:` (`NSControl`), `setStringValue:`.

target

```
- (id)target
```

Returns `nil`. This method is one of those overridden by `NSActionCell` and its subclasses to implement target-action functionality, in this case to return the target object. See also `setTarget:`, `action`, `NSActionCell`.

titleRectForBounds:

```
- (NSRect)titleRectForBounds:(NSRect)theRect
```

Returns the rectangle where the cell's title is drawn.

trackMouse:inRect:ofView:untilMouseUp:

```
- (BOOL)trackMouse:(NSEvent *)theEvent inRect:(NSRect)cellFrame
    ofView:(NSView *)controlView untilMouseUp:(BOOL)flag
```

Tracks the mouse, returning YES if the mouse goes up while in `cellFrame`. This method is usually invoked by an `NSControl`'s `mouseDown:` method, which passes the mouse-down event in `theEvent`. If `flag` is YES, the method keeps tracking until the mouse goes up; otherwise it tracks until the mouse leaves `cellFrame`. This method is generally not overridden since the default implementation invokes other cell methods that can be overridden to handle specific events in a dragging session.

This method first invokes `startTrackingAt:inView:`. If that method returns YES, then as mouse-dragged events are intercepted, `continueTracking:at:inView:` is invoked, and, finally, when the mouse leaves the bounds or if the mouse button goes up, `stopTracking:at:inView:mouseIsUp:` is invoked. If `cellFrame` is `NULL`, then the bounds are considered infinitely large. You usually override one or more of these methods to respond to specific mouse events.

If the other tracking methods are insufficient for your requirements, override this method directly. This method's responsibility is to invoke `controlView`'s `sendAction:to:` method when appropriate (before, during, or after tracking) and to return YES if and only if the mouse goes up within the cell during tracking. If the cell's action is sent on a mouse-down event, then `startTrackingAt:inView:` is invoked *before* the action is sent, and the mouse is tracked until it goes up or out of bounds. If the cell sends its action periodically, then the action is sent periodically to the target even if the mouse isn't moving (although `continueTracking:at:inView:` is only invoked when the mouse changes position). If the cell's action is sent on a mouse-dragged event, then `continueTracking:at:inView:` is invoked *before* the action is sent. See also `continueTracking:at:inView:`, `stopTracking:at:inView:mouseIsUp:`.

type

```
- (NSCellType)type
```

Returns the cell's type which can be one of the following:

- `NSNullCellType`
- `NSTextCellType`

- `NSImageCellType`

See also `setType:`.

`wraps`

`- (BOOL)wraps`

Returns `YES` if the cell's text is word-wrapped, and returns `NO` otherwise.

NSClipView

Inherits From:	NSView : NSResponder : NSObject
Conforms To:	NSCoding (NSResponder), NSObject (NSObject)
Declared In:	AppKit/NSClipView.h

Class Description

An `NSClipView` object lets you scroll a document that may be larger than the `NSClipView`'s frame rectangle, clipping the visible portion of the document to the frame. You don't normally use the `NSClipView` class directly; it's provided primarily as the scrolling machinery for the `NSScrollView` class. However, you might use the `NSClipView` class to implement a class similar to `NSScrollView`. The document, which must be an `NSView`, is called the `NSClipView`'s *document view*. An `NSClipView`'s document view, which is set through the `setDocumentView:` method, is the `NSClipView`'s only subview. You can set the cursor that's displayed when the mouse enters an `NSClipView`'s frame (in other words, when it's poised over the document view) through the `setDocumentCursor:` method.

When the `NSClipView` is instructed to scroll its document view, it normally copies that portion of the document view that's visible both before and after the scrolling, so that this part won't need to be redrawn from scratch. However, you can turn off this behavior and force the entire visible area to be redrawn by sending the `NSClipView` a `setCopiesOnScroll:NO` message.

After scrolling, the NSClipView sends itself a setNeedsDisplayInRect: message to indicate that some part of the document view should be displayed again. The argument to this message is the freshly exposed area of the document view unless the NSClipView received a setCopiesOnScroll:NO message, in which case the argument is the entire visible area.

The NSClipView sends its superview (usually an NSScrollView) a reflectScrolledClipView: message whenever the relationship between the NSClipView and the document view has changed. This allows the superview to update itself to reflect the change—for example, the NSScrollView class uses this method to change the position of its scrollers when the user causes the document to autoscroll.

Note – Do not send the addSubview:, addSubview:positioned:relativeTo:, and replaceSubview:with: methods, inherited from NSView, to an NSClipView object. Use NSClipView's setDocumentView: method instead.

Method Types

Activity	Class Method
Managing the document view	– documentRect – documentView – documentVisibleRect – setDocumentView:
Setting the cursor	– documentCursor – setDocumentCursor:
Setting the background color	– backgroundColor – setBackgroundColor:
Scrolling	– autoscroll: – constrainScrollPoint: – copiesOnScroll – scrollToPoint: – setCopiesOnScroll:
Responding to a changed frame	– viewFrameChanged:

Instance Methods

autoscroll:

```
- (BOOL)autoscroll:(NSEvent *)theEvent
```

Performs automatic scrolling of the document. Returns YES if the scrolling occurs and NO otherwise. You never invoke this method directly; instead, the NSClipView's document view should send autoscroll: to itself while inside a modal event loop initiated by a mouse-down event when the mouse is dragged outside the NSClipView's frame. The NSView class implements autoscroll: to forward the message to the NSView's superview; the message forwarded to the NSClipView. See also autoscroll: (NSView).

backgroundColor

```
- (NSColor *)backgroundColor
```

Returns the clip view's background color. If no background color has been set, the background color of the clip view's window is returned. See also setBackgroundColor:, backgroundColor (NSWindow), NSColor.

constrainScrollPoint:

```
- (NSPoint)constrainScrollPoint:(NSPoint)newOrigin
```

Ensures that the document view is not scrolled to an undesirable position. This method is invoked by the private method that all scrolling messages go through before it invokes `scrollToPoint:`. The default implementation keeps as much of the document view visible as possible. You may want to override this method to provide alternate constraining behavior. `newOrigin` is the desired new origin of the clip view's bounds rectangle, given in `NSClipView`'s coordinate system. See also `scrollToPoint:`.

copiesOnScroll

```
- (BOOL)copiesOnScroll
```

Indicates whether the visible portions of the document view are copied when scrolling occurs. If not, the document view is responsible for redrawing the entire visible portion. The default is `YES`.

documentCursor

```
-(NSCursor *)documentCursor
```

Returns the cursor for the document view.

documentRect

```
- (NSRect)documentRect
```

Returns the smallest rectangle that encloses both the document view's frame and the clip view's frame. The origin of the rectangle is always set to that of the document view's frame. The document rectangle is used in conjunction with the clip view's bounds rectangle to determine values for any indicators of relative position and size between the clip view and the document view. The `NSScrollView` uses these rectangles to set the size and position of the `NSScroller`s' knobs. See also `reflectScrolledClipView:` (`NSScrollView`).

documentView

```
- (id)documentView
```

Returns the clip view's document view. See also `setDocumentView:`.

documentVisibleRect

`- (NSRect)documentVisibleRect`

Returns the document view portion that's visible within the clip view. The visible rectangle is given in the document view's coordinate system. Note that this rectangle doesn't reflect the effects of any clipping that might occur above the clip view itself. To get the portion of the document view that's guaranteed to be visible, send it a `visibleRect` message. See also `visibleRect` (`NSView`).

scrollToPoint:

`- (void)scrollToPoint:(NSPoint)newOrigin`

Performs scrolling of the document view. This method sets the clip view's bounds rectangle origin to `newOrigin`. Then it copies as much of the previously visible document as possible, unless it received a `setCopiesOnScroll:NO` message. It then sends its document view a message to either display or invalidate the newly exposed region(s) of the clip view. The `scrollToPoint:` method doesn't send a `reflectScrolledClipView:` (`NSScrollView`) message to its superview; that message is sent by the method that invokes `scrollToPoint:`. Note also that while the clip view provides clipping to its frame, it doesn't clip to the update rectangles.

This method is used by a private method through which all scrolling passes, and is invoked if the clip view's superview does not implement the `scrollClipView:toPoint:` (`NSView`) method. If the clip view's superview does implement `scrollClipView:toPoint:`, that method should invoke `scrollToPoint:`. This mechanism is provided so that the clip view's superview can coordinate scrolling of multiple tiled clip views. (Note that NSScrollView doesn't implement the `scrollClipView:toPoint:` method.)

setBackgroundColor:

`- (void)setBackgroundColor:(NSColor *)color`

Sets the clip view's background color. This color is used to fill the area inside the clip view that's not covered by opaque portions of the document view. See also `backgroundColor`.

setCopiesOnScroll:

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

Determines whether visible portions of the document view will be copied when scrolling occurs. If `flag` is `YES`, scrolling will copy as much of the document as possible to scroll the `NSView`, allowing the document view to update only the newly exposed portions of itself. If `flag` is `NO`, the document view is responsible for redrawing its entire visible portion. This should rarely be changed from the default value (`YES`).

setDocumentCursor:

```
- (void)setDocumentCursor:(NSCursor *)anObject
```

Sets the cursor for the document view. See also `documentCursor`.

setDocumentView:

```
- (void)setDocumentView:(NSView *)aView
```

Makes `aView` the clip view's document view. An clip view can have only one document view; invoking this method removes the previous document view, if any. The origin of the document view's frame is initially set to be coincident with the origin of the clip view's bounds. If the clip view is contained within an `NSScrollView`, you should send the `NSScrollView` the `setDocumentView:` message and have the `NSScrollView` pass this message on to the clip view. See also `setDocumentView:` (`NSScrollView`).

viewFrameChanged:

```
- (void)viewFrameChanged:(NSNotification *)notification
```

Sends notification that the document view's frame has changed. See also `NSNotification`.

1

NSCoder Additions

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

Class Description

The Application Kit adds this method to the Foundation Kit's `NSCoder` class. This method becomes part of the class for all applications that use the Application Kit, but not for applications that don't use the Application Kit.

Instance Methods

decodeNXColor

```
- (NSColor *)decodeNXColor
```

Returns an autoreleased `NSColor` object equivalent to the archived `NXColor` structure.

Note – This method is needed to read colors from archives that were created by pre-OpenStep versions of NeXTSTEP.

NSColor

Inherits From:	NSObject
Conforms To:	NSCoding, NSCopying, NSObject (NSObject)
Declared In:	AppKit/NSColor.h

An `NSColor` object represents a color. The color can be a grayscale value and can include alpha (transparency) information. By sending a `set` message to an `NSColor` object, you set the color for the current PostScript drawing context. This causes subsequently drawn graphics to have the color represented by the `NSColor` instance.

A color is defined in some particular *color space*. A color space consists of a set of dimensions—such as red, green, and blue in the case of RGB space. Each point in the space represents a unique color, and the point's location along each dimension is called a *component*. An individual color is usually specified by the numeric values of its components, which range from 0.0 to 1.0. For instance, a pure red is specified in RGB space by the component values 1.0, 0.0, and 0.0.

Some color spaces include an *alpha* component, which defines the color's transparency. This component tells Display PostScript how to blend colors. An alpha value of 1.0 means completely opaque, and 0.0 means completely transparent. The alpha component is ignored when the color is used on a device that doesn't support alpha, such as a printer.

There are three kinds of color space in OpenStep:

- *Device-dependent*. This means that a given color might not look the same on different displays and printers.
- *Device-independent*, also known as *calibrated.* With this sort of color space, a given color should look the same on all devices.
- *Named*. The "named color space" has components that aren't numeric values, but simply names in various catalogs of colors. Named colors come with lookup tables that provide the ability to generate the correct color on a given device.

OpenStep includes six different color spaces, referred to by these enumeration constants:

Table 1-10 OpenStep Color Spaces

Enumeration Constant	Color Components
NSDeviceCMYKColorSpace	Cyan, magenta, yellow, black, and alpha components
NSDeviceWhiteColorSpace	White and alpha components
NSDeviceRGBColorSpace	Red, green, blue, and alpha components Hue, saturation, brightness, and alpha components

Table 1-10 OpenStep Color Spaces

Enumeration Constant	Color Components
NSCalibratedWhiteColorSpace	White and alpha components
NSCalibratedRGBColorSpace	Red, green, blue, and alpha components Hue, saturation, brightness, and alpha components
NSNamedColorSpace	Catalog name and color name components

Color spaces whose names start with "`NSDevice`" are device-dependent; those with "`NSCalibrated`" are device-independent.

Note – It's illegal to ask a color for components that are not defined for its colorspace (for example, you cannot ask a CMYK color for its RGB values).

There's usually no need to retrieve the individual components of a color, however when needed, you can either retrieve a set of components using such methods as `getRed:green:blue:alpha:`, or an individual component (using such methods as `redComponent`). Remember, however, it's illegal to ask an `NSColor` for components that aren't defined for its color space. You can identify the color space by sending a `colorSpaceName` method to the `NSColor`. If you need to ask an `NSColor` for components that aren't in its color space (for instance, you need to know the RGB components of a color you've gotten from the color panel), first convert the color to the appropriate color space using the `colorUsingColorSpaceName:` method. For example:

```
NSColor *rgbColor =
    [someColor colorUsingColorSpaceName:NSCalibratedRGBColorSpace];
```

Then you can retrieve the needed components:

```
brightness = [rgbColor brightnessComponent];
hue = [rgbColor hueComponent];
saturation = [rgbColor saturationComponent];
alpha = [rgbColor alphaComponent];
```

If the color is already in the specified color space, you get the same color back; otherwise, you get a conversion that's usually lossy or that's correct only for the current device. You might also get back `nil` if the specified conversion can't be done.

`NSColor` subclasses need to implement the `colorSpaceName` and `set` methods, as well as the methods that return the components for that color space and the methods in the `NSCoding` protocol. Some other methods—such as `colorWithAlphaComponent:`, `isEqual:`, and `colorUsingColorSpaceName:device:`—may also be implemented if they make sense for the color space. Mutable subclasses (if any) should additionally implement `copyWithZone:` to provide a true copy.

Method Types

Activity	Class Method
Creating anNSColor from component values	+ colorWithCalibratedHue:saturation:brightness: alpha: + colorWithCalibratedRed:green:blue:alpha: + colorWithCalibratedWhite:alpha: + colorWithCatalogName:colorName: + colorWithDeviceCyan:magenta:yellow: black:alpha: + colorWithDeviceHue:saturation:brightness: alpha: + colorWithDeviceRed:green:blue:alpha: + colorWithDeviceWhite:alpha:
Creating an NSColor with preset components	+ blackColor + blueColor + brownColor + clearColor + cyanColor + darkGrayColor + grayColor + greenColor + lightGrayColor + magentaColor + orangeColor + purpleColor + redColor + whiteColor + yellowColor
Ignoring alpha components	+ ignoresAlpha + setIgnoresAlpha:
Retrieving a set of components	– getCyan:magenta:yellow:black:alpha: – getHue:saturation:brightness:alpha: – getRed:green:blue:alpha: – getWhite:alpha:

Activity	Class Method
Retrieving individual components	– alphaComponent – blackComponent – blueComponent – brightnessComponent – catalogNameComponent – colorNameComponent – cyanComponent – greenComponent – hueComponent – localizedCatalogNameComponent – localizedColorNameComponent – magentaComponent – redComponent – saturationComponent – whiteComponent – yellowComponent
Converting to another color space	– colorSpaceName – colorUsingColorSpaceName: – colorUsingColorSpaceName:device:
Changing the color	– blendedColorWithFraction:ofColor: – colorWithAlphaComponent:
Copying and pasting	+ colorFromPasteboard: – writeToPasteboard:
Drawing	– drawSwatchInRect: – set

Class Methods

`blackColor`

```
+ (NSColor *)blackColor
```

Returns a color in `NSCalibratedWhiteColorSpace` whose grayscale value is 0.0 and whose alpha value is 1.0.

`blueColor`

```
+ (NSColor *)blueColor
```

Returns a color in NSCalibratedRGBColorSpace whose RGB value is 0.0, 0.0, 1.0 and whose alpha value is 1.0.

brownColor

```
+ (NSColor *)brownColor
```

Returns a color in NSCalibratedRGBColorSpace whose RGB value is 0.6, 0.4, 0.2 and whose alpha value is 1.0.

clearColor

```
+ (NSColor *)clearColor
```

Returns a color in NSCalibratedWhiteColorSpace whose grayscale and alpha values are both 0.0.

colorFromPasteboard:

```
+ (NSColor *)colorFromPasteboard:(NSPasteboard *)pasteBoard
```

Returns the color currently on the pasteboard, or nil if the pasteboard doesn't contain color data. The returned color's alpha component is set to 1.0 if ignoresAlpha returns YES.

colorWithCalibratedHue:saturation:brightness:alpha:

```
+ (NSColor *)colorWithCalibratedHue:(float)hue
    saturation:(float)saturation brightness:(float)brightness
    alpha:(float)alpha
```

Creates and returns a new color whose color space is NSCalibratedRGBColorSpace, whose opacity value is alpha, and whose components in HSB space would be hue, saturation, and brightness. All values are legal, but values less than 0.0 are set to 0.0, and values greater than 1.0 are set to 1.0.

colorWithCalibratedRed:green:blue:alpha:

```
+ (NSColor *)colorWithCalibratedRed:(float)red green:(float)green
    blue:(float)blue alpha:(float)alpha
```

Creates and returns a new color whose color space is NSCalibratedRGBColorSpace, whose opacity value is alpha, and whose RGB components are red, green, and blue. All values are legal, but values less than 0.0 are set to 0.0, and values greater than 1.0 are set to 1.0.

colorWithCalibratedWhite:alpha:

```
+ (NSColor *)colorWithCalibratedWhite:(float)white
    alpha:(float)alpha
```

Creates and returns a new color whose color space is NSCalibratedWhiteColorSpace, whose opacity value is alpha, and whose grayscale value is white. All values are legal, but values less than 0.0 are set to 0.0, and values greater than 1.0 are set to 1.0.

colorWithCatalogName:colorName:

```
+ (NSColor *)colorWithCatalogName:(NSString *)listName
    colorName:(NSString *)colorName
```

Creates and returns a new NSColor whose color space is NSNamedColorSpace, by finding the color named colorName in the catalog named listName (for example Pantone).

colorWithDeviceCyan:magenta:yellow: black:alpha:

```
+ (NSColor *)colorWithDeviceCyan:(float)cyan magenta:(float)magenta
    yellow:(float)yellow black:(float)black alpha:(float)alpha
```

Creates and returns a new NSColor whose color space is NSDeviceCMYKColorSpace, whose opacity value is alpha, and whose CMYK components are cyan, magenta, yellow, and black. All values are legal, but values less than 0.0 are set to 0.0, and values greater than 1.0 are set to 1.0. This color space corresponds to the setcmykcolor PostScript operator.

colorWithDeviceHue:saturation:brightness:alpha:

```
+ (NSColor *)colorWithDeviceHue:(float)hue
    saturation:(float)saturation
    brightness:(float)brightness alpha:(float)alpha
```

Creates and returns a new `NSColor` whose color space is `NSDeviceRGBColorSpace`, whose opacity value is `alpha`, and whose components in HSB space would be `hue`, `saturation`, and `brightness`. All values are legal, but values less than 0.0 are set to 0.0, and values greater than 1.0 are set to 1.0. This color space corresponds to the `sethsbcolor` PostScript operator.

colorWithDeviceRed:green:blue:alpha:

```
+ (NSColor *)colorWithDeviceRed:(float)red green:(float)green
    blue:(float)blue alpha:(float)alpha
```

Creates and returns a new `NSColor` whose color space is `NSDeviceRGBColorSpace`, whose opacity value is `alpha`, and whose RGB components are `red`, `green`, and `blue`. All values are legal, but values less than 0.0 are set to 0.0, and values greater than 1.0 are set to 1.0. This color space corresponds to the `setrgbcolor` PostScript operator.

colorWithDeviceWhite:alpha:

```
+ (NSColor *)colorWithDeviceWhite:(float)white alpha:(float)alpha
```

Creates and returns a new `NSColor` whose color space is `NSDeviceWhiteColorSpace`, whose opacity value is `alpha`, and whose grayscale value is `white`. All values are legal, but values less than 0.0 are set to 0.0, and values greater than 1.0 are set to 1.0. This color space corresponds to the `setgray` PostScript operator.

cyanColor

```
+ (NSColor *)cyanColor
```

Returns an `NSColor` in `NSCalibratedRGBColorSpace` whose RGB value is 0.0, 1.0, 1.0 and whose alpha value is 1.0.

darkGrayColor

```
+ (NSColor *)darkGrayColor
```

Returns an NSColor in NSCalibratedWhiteColorSpace whose grayscale value is 1/3 and whose alpha value is 1.0. See also grayColor, lightGrayColor.

grayColor

```
+ (NSColor *)grayColor
```

Returns an NSColor in NSCalibratedWhiteColorSpace whose grayscale value is 0.5 and whose alpha value is 1.0. See also lightGrayColor, darkGrayColor.

greenColor

```
+ (NSColor *)greenColor
```

Returns an NSColor in NSCalibratedRGBColorSpace whose RGB value is 0.0, 1.0, 0.0 and whose alpha value is 1.0.

ignoresAlpha

```
+ (BOOL)ignoresAlpha
```

Returns YES (the default) if the application hides the color panel's opacity slider and sets imported colors' alpha values to 1.0. See also setIgnoresAlpha:.

lightGrayColor

```
+ (NSColor *)lightGrayColor
```

Returns an NSColor in NSCalibratedWhiteColorSpace whose grayscale value is 2/3 and whose alpha value is 1.0. See also darkGrayColor.

magentaColor

```
+ (NSColor *)magentaColor
```

Returns an NSColor in NSCalibratedRGBColorSpace whose RGB value is 1.0, 0.0, 1.0 and whose alpha value is 1.0.

orangeColor

```
+ (NSColor *)orangeColor
```

Returns an NSColor in NSCalibratedRGBColorSpace whose RGB value is 1.0, 0.5, 0.0 and whose alpha value is 1.0.

purpleColor

```
+ (NSColor *)purpleColor
```

Returns an NSColor in NSCalibratedRGBColorSpace whose RGB value is 0.5, 0.0, 0.5 and whose alpha value is 1.0.

redColor

```
+ (NSColor *)redColor
```

Returns an NSColor in NSCalibratedRGBColorSpace whose RGB value is 1.0, 0.0, 0.0 and whose alpha value is 1.0.

setIgnoresAlpha:

```
+ (void)setIgnoresAlpha:(BOOL)flag
```

If flag is YES, no opacity slider is displayed in the color panel, and colors dragged in or pasted have their alpha values set to 1.0. See also ignoresAlpha.

whiteColor

```
+ (NSColor *)whiteColor
```

Returns an NSColor in NSCalibratedWhiteColorSpace whose grayscale and alpha values are both 1.0.

yellowColor

```
+ (NSColor *)yellowColor
```

Returns an NSColor in NSCalibratedRGBColorSpace whose RGB value is 1.0, 1.0, 0.0 and whose alpha value is 1.0.

Instance Methods

alphaComponent

```
- (float)alphaComponent
```

Returns the alpha (opacity) component (1.0 by default).

blackComponent

```
- (float)blackComponent
```

Returns the black component. It's an error if the receiver isn't a CMYK color.

blendedColorWithFraction:ofColor:

```
- (NSColor *)blendedColorWithFraction:(float)fraction
    ofColor:(NSColor *)aColor
```

Returns a newly created NSColor in NSCalibratedRGBColorSpace whose component values are a weighted sum of the receiver's and aColor's. The method converts aColor and a copy of the receiver to RGB, and then sets each component of the returned color to fraction of aColor's value plus 1 – fraction of the receiver's. If the colors can't be converted to NSCalibratedRGBColorSpace, nil is returned.

blueComponent

```
- (float)blueComponent
```

Returns the blue component. It's an error if the receiver isn't an RGB color.

brightnessComponent

```
- (float)brightnessComponent
```

Returns the brightness component of the HSB color equivalent to the receiver. It's an error if the receiver isn't an RGB color.

catalogNameComponent

```
- (NSString *)catalogNameComponent
```

Returns the name of the catalog containing this color, or `nil` if the receiver's color space isn't `NSNamedColorSpace`.

colorNameComponent

```
- (NSString *)colorNameComponent
```

Returns the name of this color, or `nil` if the receiver's color space isn't `NSNamedColorSpace`.

colorSpaceName

```
- (NSString *)colorSpaceName
```

Returns the name of the `NSColor`'s color space.

colorUsingColorSpaceName:

```
- (NSColor *)colorUsingColorSpaceName:(NSString *)colorSpace
```

Returns a newly created `NSColor` whose color is the same as the receiver's, except that the new `NSColor` is in the color space named `colorSpace`. This method calls `colorUsingColorSpaceName:device:` with the current device, indicating that the color is appropriate for the current device (the current window if drawing, or the current printer if printing). See the Class Description for an example of using this method.

colorUsingColorSpaceName:device:

```
- (NSColor *)colorUsingColorSpaceName:(NSString *)colorSpace
    device:(NSDictionary *)deviceDescription
```

Returns a newly created `NSColor` whose color is the same as the receiver's, except that the new `NSColor` is in the color space named `colorSpace` and is specific to the device described by `deviceDescription`. If `deviceDescription` is `nil`, then current device is used (as obtained from the currently focused view's window, or if printing, the current printer). If `colorSpace` is `nil`, then the most appropriate color space is used.

colorWithAlphaComponent:

```
- (NSColor *)colorWithAlphaComponent:(float)alpha
```

Returns a newly created `NSColor` that has the same color space and component values as the receiver, except that its alpha component is `alpha`. If the receiver's color space doesn't include an alpha component, the receiver is returned.

cyanComponent

```
- (float)cyanComponent
```

Returns the cyan component. It's an error if the receiver isn't a CMYK color.

drawSwatchInRect:

```
- (void)drawSwatchInRect:(NSRect)rect
```

Draws the current color in the rectangle `rect`. Subclasses adorn the rectangle in some manner to indicate the type of color. This method is invoked by color wells, swatches, and other user-interface objects that need to display colors.

getCyan:magenta:yellow:black:alpha:

```
- (void)getCyan:(float *)cyan magenta:(float *)magenta
    yellow:(float *)yellow black:(float *)black
    alpha:(float *)alpha
```

Returns the CMYK and alpha values in the respective arguments. If `NULL` is passed in as an argument, the method doesn't set that value. It is an error to send this message to a receiver that isn't a CMYK color.

getHue:saturation:brightness:alpha:

```
- (void)getHue:(float *)hue saturation:(float *)saturation
    brightness:(float *)brightness alpha:(float *)alpha
```

Returns the HSB and alpha values in the respective arguments. If `NULL` is passed in as an argument, the method doesn't set that value. It is an error to send this message to a receiver that isn't a CMYK color.

getRed:green:blue:alpha:

```
- (void)getRed:(float *)red green:(float *)green blue:(float *)blue
    alpha:(float *)alpha
```

Returns the RGB and alpha values in the respective arguments. If NULL is passed in as an argument, the method doesn't set that value. It is an error to send this message to a receiver that isn't an RGB color.

getWhite:alpha:

```
- (void)getWhite:(float *)white alpha:(float *)alpha
```

Returns the grayscale and alpha values in the respective arguments. If NULL is passed in as an argument, the method doesn't set that value. It is an error to send this message to a receiver that isn't in NSCalibratedWhiteColorSpace.

greenComponent

```
- (float)greenComponent
```

Returns the green component. It is an error to send this message to a receiver that isn't an RGB color.

hueComponent

```
- (float)hueComponent
```

Returns the hue component of the HSB color equivalent to the receiver. It is an error to send this message to a receiver that isn't an RGB color.

localizedCatalogNameComponent

```
- (NSString *)localizedCatalogNameComponent
```

Similar to catalogNameComponent, but returns a localized string.

localizedColorNameComponent

```
- (NSString *)localizedColorNameComponent
```

Similar to colorNameComponent, but returns a localized string.

magentaComponent

```
- (float)magentaComponent
```

Returns the magenta component. It is an error to send this message to a receiver that isn't a CYMK color.

redComponent

```
- (float)redComponent
```

Returns the red component. It is an error to send this message to a receiver that isn't an RGB color.

saturationComponent

```
- (float)saturationComponent
```

Returns the saturation component of the HSB color equivalent to the receiver. It is an error to send this message to a receiver that isn't an RGB color.

set

```
- (void)set
```

Sets the color of subsequent PostScript drawing to the color that the receiver represents. If the application is drawing to the screen rather than printing, this method also sets the current drawing context's alpha value to the value returned by `alphaComponent`. This method should be implemented by subclasses.

whiteComponent

```
- (float)whiteComponent
```

Returns the white component. It is an error to send this message to a receiver that isn't a grayscale color.

writeToPasteboard:

```
- (void)writeToPasteboard:(NSPasteboard *)pasteBoard
```

Writes the receiver's data to the pasteboard unless the pasteboard doesn't support color data, in which case the method does nothing.

yellowComponent

```
- (float)yellowComponent
```

Returns the yellow component. It is an error to send this message to a receiver that isn't a CYMK color.

NSColorList

Inherits From:	NSObject
Conforms To:	NSCoding, NSObject (NSObject)
Declared In:	AppKit / NSColorList.h

Class Description

Instances of `NSColorList` are used to manage named lists of `NSColor`s. `NSColorPanel`'s list-mode color picker uses instances of `NSColorList` to represent any lists of colors that come with the system, as well as any lists created by the user. An application can `use NSColorList` to manage document-specific color lists, which may be added to an application's `NSColorPanel` using its `attachColorList:` method.

An `NSColorList` is similar to a dictionary object: An `NSColor` is added to, looked up in, and removed from the list by specifying its key, which is an `NSString`. In addition, colors can be inserted at specified positions in the list. The list itself has a name, specified when you create the object using either `initWithName:` or `initWithName:fromFile:`.

An `NSColorList` saves and retrieves its colors from files with the extension `.clr` in directories defined by a standard search path. To access all the color lists in the standard search path, use the `availableColorLists` method; this returns an array of `NSColorList`s, from which you can retrieve the individual color lists by name.

`NSColorList` reads color list files in several different formats; it saves color lists using the archiver API.

Method Types

Activity	Class Method
Initializing an NSColorList	– initWithName: – initWithName:fromFile:
Getting all color lists	+ availableColorLists
Getting a color list by name	+ colorListNamed: – name
Managing colors by key	– allKeys – colorWithKey: – insertColor:key:atIndex: – removeColorWithKey: – setColor:forKey:
Editing	– isEditable
Writing and removing files	– writeToFile: – removeFile

Class Methods

availableColorLists

```
+ (NSArray *)availableColorLists
```

Returns an array of all color lists found in the standard color list directories. Color lists created at run time aren't included in this list unless they're saved into one of the standard color list directories.

colorListNamed:

```
+ (NSColorList *)colorListNamed:(NSString *)name
```

Searches the array that's returned by `availableColorLists` and returns the color list `name`, or `nil` if no such color list exists. `name` must not include the `.clr` suffix.

Instance Methods

allKeys

```
- (NSArray *)allKeys
```

Returns an array of NSString objects that contains all the keys by which the NSColors are stored in the color list. The length of this array equals the number of colors, and its contents are arranged according to the ordering specified when the colors were inserted.

colorWithKey:

```
- (NSColor *)colorWithKey:(NSString *)key
```

Returns the color associated with key, or nil if there is none.

initWithName:

```
- (id)initWithName:(NSString *)name
```

Initializes and returns the receiver, registering it under the specified name if the name isn't in use already.

initWithName:fromFile:

```
- (id)initWithName:(NSString *)name fromFile:(NSString *)path
```

Initializes and returns the receiver, registering it under the specified name if the name isn't in use already. path should be the full path to the file for the color list; name should be the name of the file for the color list minus the .clr extension.

insertColor:key:atIndex:

```
- (void)insertColor:(NSColor *)color key:(NSString *)key
    atIndex:(unsigned)location
```

Inserts color at the specified location in the list, which is numbered starting with 0. If the list already contains a color with the same key at a different location, it's removed from the old location. This method posts the NSColorListDidChangeNotification notification to the default

notification center. Raises `NSColorListNotEditableException` if the color list is not editable. This method posts the `NSColorListDidChangeNotification` notification to the default notification center.

isEditable

```
- (BOOL)isEditable
```

Returns `YES` if the color list can be modified. This depends on the source of the list: If it came from a write-protected file, this method returns `NO`.

name

```
- (NSString *)name
```

Returns the color list name.

removeColorWithKey:

```
- (void)removeColorWithKey:(NSString *)key
```

Removes the color associated with `key` from the list. This method does nothing if the list doesn't contain the key. This method posts the `NSColorListDidChangeNotification` notification to the default notification center. Raises `NSColorListNotEditableException` if the color list is not editable.

removeFile

```
- (void)removeFile
```

Deletes the file from which the list was created unless the user doesn't own the color list. The receiver is removed from the list of available colors, but isn't released.

setColor:forKey:

```
- (void)setColor:(NSColor *)aColor forKey:(NSString *)key
```

Associates the specified color with the key `key`. If the list already contains `key`, this method sets the corresponding color to `aColor`; otherwise, it inserts `aColor` at the end of the list.

writeToFile:

```
- (BOOL)writeToFile:(NSString *)path
```

If `path` is a directory, saves the color list in a file named `listname.clr` (where `listname` is the name with which the color list was initialized). If `path` includes a file name, this method saves the file under that name. If `path` is `nil`, this method saves the file as `listname.clr` in the standard location. Returns `YES` upon success.

NSColorPanel

Inherits From:	NSPanel : NSWindow : NSResponder : NSObject
Conforms To:	NSCoding (NSResponder) NSObject (NSObject)
Declared In:	AppKit/NSColorPanel.h

Class Description

`NSColorPanel` provides a standard user interface for selecting color in an application. It provides a number of standard color selection modes, and, with the `NSColorPickingDefault` and `NSColorPickingCustom` protocols, allows an application to add its own color selection modes. It allows the user to save swatches containing frequently used colors. Once set, these swatches are displayed by `NSColorPanel` in any application where it is used, giving the user color consistency between applications. `NSColorPanel` enables users to capture a color anywhere on the screen for use in the active application, and allows dragging colors from itself into views in an application. `NSColorPanel`'s action message is sent to the target object when the user changes the current color.

An application has only one instance of `NSColorPanel`, the shared instance. Invoking the `sharedColorPanel:` method returns the shared instance of `NSColorPanel`, instantiating it if necessary. You can also initialize an `NSColorPanel` for your application by invoking `NSApplication`'s `orderFrontColorPanel` method.

You can put `NSColorPanel` in any application created with Interface Builder by adding the "Colors..." item from the Menu palette to the application's menu.

Color Mask and Color Modes

The color mask determines which of the color modes are enabled for `NSColorPanel`. This mask is set before you initialize a new instance of `NSColorPanel`. `NSColorPanelAllModesMask` represents the logical OR of the other color mask constants: it causes the `NSColorPanel` to display all standard color pickers. When initializing a new instance of `NSColorPanel`, you can logically OR any combination of color mask constants to restrict the available color modes.

Table 1-11 Color Mask Constants

Color Mode	Color Mask Constant
Grayscale-Alpha	NSColorPanelGrayModeMask
Red-Green-Blue	NSColorPanelRGBModeMask
Cyan-Yellow-Magenta-Black	NSColorPanelCMYKModeMask
Hue-Saturation-Brightness	NSColorPanelHSBModeMask
TIFF image	NSColorPanelCustomPaletteModeMask
Custom color lists	NSColorPanelColorListModeMask
Color wheel	NSColorPanelWheelModeMask
All of the above	NSColorPanelAllModesMask

The `NSColorPanel`'s color mode mask is set using the class method `setPickerMask:`. The mask must be set before creating an application's instance of `NSColorPanel`.

When an application's instance of `NSColorPanel` is masked for more than one color mode, your program can set its active mode by invoking the `setMode:` method with a color mode constant as its argument; the user can set the mode by clicking buttons on the panel. Here are the standard color modes and mode constants:.

Table 1-12 Color Mask Constants

Color Mode	Color Mask Constant
Grayscale-Alpha	NSGrayModeColorPanel
Red-Green-Blue	NSRGBModeColorPanel
Cyan-Yellow-Magenta-Black	NSCMYKModeColorPanel
Hue-Saturation-Brightness	NSHSBModeColorPanel
TIFF image	NSCustomPaletteModeColorPanel
Color lists	NSColorListModeColorPanel
Color wheel	NSWheelModeColorPanel

In grayscale-alpha, red-green-blue, cyan-magenta-yellow-black, and hue-saturation-brightness modes, the user adjusts colors by manipulating sliders. In the custom palette mode, the user can load a TIFF file into the `NSColorPanel`, then select colors from the TIFF image. In custom color list mode, the user can create and load lists of named colors. The two custom modes provide `NSPopUpLists` for loading and saving files. Finally, color wheel mode provides a simplified control for selecting colors. If a color panel has been used, it uses whatever mode it was in last as the default mode when `NSColorPanelAllModesMask` is used to initialize the `NSColorPanel`. Otherwise, it uses color wheel mode.

Associated Classes and Protocols

The `NSColorList` class provides an application programming interface (API) for managing custom color lists. The `NSColorPanel` methods `attachColorList:` and `detachColorList:` let your application add and remove custom lists from the `NSColorPanel`'s user interface.

The protocols `NSColorPickingDefault` and `NSColorPickingCustom` provide an API for adding custom color selection to the user interface. The `NSColorPicker` class implements the `NSColorPickingDefault` protocol;

you can subclass `NSColorPicker` and implement the `NSColorPickingCustom` protocol in your subclass to create your own user interface for color selection.

See also `NSColorList`, `NSColorPickingDefault`, `NSColorPicker`, `NSColorPickingDefault` protocol, `NSColorPickingCustom` protocol, `NSColorWell`.

Method Types

Activity	Class Method
Creating the NSColor Panel	+ sharedColorPanel + sharedColorPanelExists
Setting the NSColorPanel	+ setPickerMask: + setPickerMode: – isContinuous – mode – setAccessoryView: – setAction: – setContinuous: – setMode: – setShowsAlpha: – setTarget: – showsAlpha
Attaching a color list	– attachColorList: – detachColorList:
Setting color	+ dragColor:withEvent:fromView: – alpha – color – setColor:

Class Methods

`dragColor:withEvent:fromView:`

```
+(BOOL)dragColor:(NSColor **)aColor withEvent:(NSEvent *)anEvent
    fromView:(NSView *)sourceView
```

Drags aColor into a destination view from sourceView. This method is usually invoked by the mouseDown: method of sourceView, for example NSColorWell. The dragging mechanism handles all subsequent events. Because it is a class method, this method can be invoked whether or not the instance of NSColorPanel exists. See also alpha, color, setColor:.

setPickerMask:

```
+ (void)setPickerMask:(int)mask
```

Sets the mask that determines which color selection modes are available in the color panel. Accepts as a parameter one or more logically OR'd color mode masks described in the "Class Description". This determines which color selection modes will be available in an application's NSColorPanel. This method only has an effect before NSColorPanel is instantiated. If you create a class that implements the color picking protocols (NSColorPickingDefault and NXColorPickingCustom), you may want to give it a unique mask—one different from those defined for the standard color pickers. To display your color picker, your application will need to logically OR that unique mask with the standard color mask constants when invoking this method. See also setPickerMode:, NSColorPicker, NSColorPickingDefault (protocol), NSColorPickingCustom (protocol).

setPickerMode:

```
+ (void)setPickerMode:(int)mode
```

Sets the color panel's initial picker mode (see the "Class Description"). The mode determines which picker will initially be visible. This method may be called at any time, whether or not an application's NSColorPanel has been instantiated. See also setPickerMask:, setMode:, setMode: (NSColorPicker).

sharedColorPanel

```
+ (NSColorPanel *)sharedColorPanel
```

Creates the shared NSColorPanel instance, if an instance doesn't already exist, and returns the shared NSColorPanel. Each application shares one instance of this object.

sharedColorPanelExists

```
+ (BOOL)sharedColorPanelExists
```

Returns YES if the NSColorPanel has been created already, and NO otherwise.

Instance Methods

accessoryView

```
- (NSView *)accessoryView
```

Returns the accessory view, or nil if there is none. See setAccessoryView:.

alpha

```
- (float)alpha
```

Returns the NSColorPanel's current alpha value, or 1.0 (opaque) if the panel has no opacity slider. See also dragColor:withEvent:fromView:, color, setColor:.

attachColorList:

```
- (void)attachColorList:(NSColorList *)aColorList
```

Adds the specified list of NSColors to all the color pickers (that conform to the NSColorPickingDefault and NSColorPickingCustom protocols) in the color panel that display color lists. See detachColorList:.

color

```
- (NSColor *)color
```

Returns the currently displayed color. See also attachColorList:, detachColorList:, dragColor:withEvent:fromView:, alpha, color, setColor:.

detachColorList:

```
- (void)detachColorList:(NSColorList *)aColorList
```

Removes the specified list of NSColors from all the color pickers in the color panel that display color lists. See also attachColorList:.

isContinuous

```
- (BOOL)isContinuous
```

Returns YES if the NSColorPanel continuously sends the action message to the target, that is, whether or not the NSColorPanel's color is being set continuously as the user manipulates the color picker. See also setContinuous:.

mode

```
- (int)mode
```

Returns the mode of the NSColorPanel. Returns the current color picker mode for the NSColorPanel. The mode constants for the standard color pickers are listed in the "Class Description". See also setMode:.

setAccessoryView:

```
- (void)setAccessoryView:(NSView *)aView
```

Sets the accessory NSView displayed in the NSColorPanel to aView. The accessory NSView can be any custom NSView that you want to display with NSColorPanel, such as a view offering color blends in a drawing program. The accessory NSView is displayed below the color picker and above the color swatches in the NSColorPanel. The NSColorPanel automatically resizes to accommodate the accessory NSView. See also accessoryView.

setAction:

```
- (void)setAction:(SEL)aSelector
```

Sets the action message sent to the target to aSelector. See also setTarget:.

setColor:

```
- (void)setColor:(NSColor *)aColor
```

Sets the color to be displayed and redraws the panel. This method posts the `NSColorPanelChangedNotification` notification with the receiving object to the default notification center. See also `color`.

setContinuous:

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

Sets the `NSColorPanel` to continuously send the action message to the target as the color of the `NSColorPanel` is set by the user. Send this message with `flag` set to `YES` if, for example, you want to continuously update the color of the target. See also `isContinuous`.

setMode:

`- (void)setMode:(int)mode`

Sets the mode of the `NSColorPanel`. See the Class Description for a list of modes. See also `mode`.

setShowsAlpha:

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

If `flag` is `YES`, sets the `NSColorPanel` to show alpha. See also `showsAlpha`, `alpha`.

setTarget:

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

Sets the target of the `NSColorPanel`'s action methods. See also `setAction:`.

showsAlpha

`- (BOOL)showsAlpha`

Returns `YES` if the `NSColorPanel` shows alpha values; retuns `NO` otherwise. See also `alpha`, `setShowsAlpha:`.

1

NSColorPicker

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

Class Description

`NSColorPicker` is an abstract superclass that implements the `NSColorPickingDefault` protocol. The `NSColorPickingDefault` and `NSColorPickingCustom` protocols define a way to add color pickers (custom user interfaces for color selection) to the `NSColorPanel`. The simplest way to implement a color picker is to create a subclass of `NSColorPicker`, instead of implementing the `NSColorPickingDefault` protocol in another kind of object. (To add functionality, implement the `NSColorPickingCustom` methods in your subclass).

The `NSColorPickingDefault` protocol specification describes the details of implementing a color picker and adding it to your application's `NSColorPanel`. Look there first for an overview of how `NSColorPicker` works. This specification is provided to document the specific behavior of `NSColorPicker`'s methods.

Method Types

Activity	Class Method
Initializing an NSColorPicker	– initWithPickerMask:colorPanel:
Getting the color panel	– colorPanel
Adding button images	– insertNewButtonImage:in: – provideNewButtonImage
Setting the mode	– setMode:
Using color lists	– attachColorList: – detachColorList:
Responding to a resized view	– setMode:

Instance Methods

attachColorList:

```
- (void)attachColorList:(NSColorList *)colorList
```

Override this method to attach a color list to a color picker. See also `detachColorList:`, `NSColorList`.

colorPanel

```
- (NSColorPanel *)colorPanel
```

Returns the `NSColorPanel` that owns this `NSColorPicker`.

detachColorList:

```
- (void)detachColorList:(NSColorList *)colorList
```

Override this method to detach a color list from a color picker. See also `attachColorList:`, `NSColorList`.

initWithPickerMask:colorPanel:

```
- (id)initWithPickerMask:(int)aMask
    colorPanel:(NSColorPanel *)colorPanel
```

Initializes the receiver for the specified mask and color panel, caching the `colorPanel` value so it can later be returned by the `colorPanel` method. Override this method to respond to the values in `aMask` or do other custom initialization. If you override this method in a subclass, you should forward the message to `super` as part of the implementation. See also `setPickerMask:` (`NSColorPanel`).

insertNewButtonImage:in:

```
- (void)insertNewButtonImage:(NSImage *)newImage
    in:(NSButtonCell *)newButtonCell
```

Called by the color panel to insert a new image into the specified cell. Override this method to customize `newImage` before insertion in `newButtonCell`. See also `provideNewButtonImage`.

provideNewButtonImage

```
- (NSImage *)provideNewButtonImage
```

Returns the button image for the color picker. The color panel will place this image in the mode button that the user uses to select this picker. (This is the same image that the color panel uses as an argument when sending the `insertNewButtonImage:in:` message.) The default implementation looks in the color picker's bundle for a TIFF file named after the color picker's class, with the extension `.tiff`.

setMode:

```
- (void)setMode:(int)mode
```

Override this method to set the color picker's mode. See also `setPickerMode:` (`NSColorPanel`).

viewSizeChanged:

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

Override to respond to a size change.

NSColorWell

Inherits From:	NSControl : NSView : NSResponder : NSObject
Conforms To:	NSCoding (NSResponder) NSObject (NSObject)
Declared In:	AppKit / NSColorWell.h

Class Description

`NSColorWell` is an `NSControl` for selecting and displaying a single color value. An example of an `NSColorWell` object (or simply color well) is found in `NSColorPanel`, which uses a color well to display the current color selection. `NSColorWell` is available from the Palettes panel of Interface Builder.

An application can have one or more active `NSColorWell`s. You can activate multiple `NSColorWell`s by invoking the `activate:` method with `NO` as its argument. When a mouse-down event occurs on an `NSColorWell`'s border, it becomes the only active color well. When a color well becomes active, it brings up the color panel also.

The `mouseDown:` method enables an instance of `NSColorWell` to send its color to another `NSColorWell` or any other subclass of `NSView` that implements the `NSDraggingDestination` protocol. See also `NSColorPanel`.

Method Types

Activity	Class Method
Drawing	– drawWellInside:
Activating	– activate: – deactivate – isActive
Managing color	– color – setColor: – takeColorFrom:
Managing borders	– isBordered – setBordered:

Instance Methods

activate:

```
- (void)activate:(BOOL)exclusive
```

Activates the `NSColorWell`, displays the Color panel, and makes the `NSColorPanel`'s current color the same as its own. If `exclusive` is `YES`, it deactivates any other `NSColorWells`; if `NO`, it keeps them active.

color

```
- (NSColor *)color
```

Returns the color of the color well.

deactivate

```
- (void)deactivate
```

Deactivates the color well.

drawWellInside:

```
- (void)drawWellInside:(NSRect)insideRect
```

Draws the colored area inside the color well at the location specified by `insideRect` without drawing borders.

isActive

```
- (BOOL)isActive
```

Returns `YES` if the color well is active. Returns `NO` otherwise.

isBordered

```
- (BOOL)isBordered
```

Indicates whether the color well is bordered.

setBordered:

```
- (void)setBordered:(BOOL)bordered
```

Places or removes a border, depending on `bordered`.

setColor:

```
- (void)setColor:(NSColor *)color
```

Sets the color of the well to `color`.

takeColorFrom:

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

Changes the color of the well to that of `sender`.

1

NSControl

Inherits From:	NSView : NSResponder : NSObject
Conforms To:	NSCoding (NSResponder) NSObject (NSObject)
Declared In:	AppKit / NSControl.h

Class Description

`NSControl` is an abstract superclass that provides three fundamental features for implementing user-interface devices. First, as a subclass of `NSView`, `NSControl` allows the on-screen representation of the device to be drawn. Second, it receives and responds to user-generated events within its bounds by overriding `NSResponder`'s `mouseDown:` method and providing a position in the responder chain. Third, it implements the `sendAction:to:` method to send an action message to the `NSControl`'s target object. Subclasses of `NSControl` defined in the Application Kit are `NSBrowser`, `NSButton` (and its subclass `NSPopUpButton`), `NSColorWell`, `NSMatrix` (and its subclass `NSForm`), `NSScroller`, `NSSlider`, and `NSTextField`.

Target and Action

Target objects and action methods provide the mechanism by which `NSControl`s interact with other objects in an application. A target is an object that an `NSControl` has effect over. The target class defines an action method to enable its instances to respond to user input. An action method takes only one argument: the `id` of the sender. The sender may be either the `NSControl` that sends the action message or another object that the target should treat as the sender. When it receives an action message, a target can return messages to the sender requesting additional information about its status. `NSControl`'s `sendAction:to:` asks the `NSApplication` object, `NSApp`, to send an action message to the `NSControl`'s target object. The method used for this is `NSApplication`'s `sendAction:to:from:`. You can also set the target to `nil` and allow it to be determined at run time. When the target is `nil`, the `NSApplication` object must look for an appropriate receiver. It conducts its search in a prescribed order, by following the responder chain until it finds an object that can respond to the message:

- It begins with the first responder in the key window and follows `nextResponder` links up the responder chain to the `NSWindow` object. After the `NSWindow` object, it tries the `NSWindow`'s delegate.
- If the main window is different from the key window, it then starts over with the first responder in the main window and works its way up the main window's responder chain to the `NSWindow` object and its delegate.
- Next, it tries to respond itself. If the `NSApplication` object can't respond, it tries its own delegate. `NSApp` and its delegate are the receivers of last resort.

`NSControl` provides methods for setting and using the target object and the action method. However, these methods require that an `NSControl` have an associated subclass of `NSCell` that provides a target and an action, such as `NSActionCell` and its subclasses.

Target objects and action methods demonstrate the close relationship between `NSControl`s and `NSCell`s. In most cases, a user interface device consists of an instance of an `NSControl` subclass paired with one or more instances of an `NSCell` subclass. Each implements specific details of the user interface mechanism. For example, `NSControl`'s `mouseDown:` method sends a `trackMouse:inRect:ofView:untilMouseUp:` message to an `NSCell`, which handles subsequent mouse and keyboard events; an `NSCell` sends an `NSControl` a `sendAction:to:` message in response to particular events. `NSControl`'s `drawRect:` method is implemented by sending a `drawWithFrame:inView:` message to the `NSCell`. As another example, `NSControl` provides methods for setting and formatting its contents; these methods send corresponding messages to `NSCell`, which actually owns the contents.

See the `NSActionCell` class specification for more on the implementation of target and action behavior.

Changing the `NSCell` *Class*

Since `NSControl` uses the `NSCell` class to implement most of its actual functionality, you can usually implement a unique user interface device by creating a subclass of `NSCell` rather than `NSControl`. As an example, let's say you want all your application's `NSSlider`s to have a type of cell other than the generic `NSSliderCell`. First, you create a subclass of `NSCell`,

NSActionCell, or NSSliderCell. For explanation purposes call it MyCellSubclass. Then, you can simply invoke NSSlider's setCellClass: class method:

```
[NSSlider setCellClass:[MyCellSubclass class]];
```

All NSSliders created thereafter will use MyCellSubclass until you call setCellClass: again.

If you want to create generic NSSliders (ones that use NSSliderCell) in the same application as the customized NSSliders that use MyCellSubclass, there are two possible approaches. One is to invoke setCellClass: as above whenever you're about to create a custom NSSlider, resetting the cell class to NSSliderCell afterwards. The other approach is to create a custom subclass of NSSlider that automatically uses MyCellSubclass, as explained in the following section.

Creating New NSControls

If you create a custom NSControl subclass that uses a custom subclass of NSCell, you should override NSControl's cellClass method:

```
+ (Class) cellClass
{
    return [MyCellSubclass class];
}
```

NSControl's initWithFrame: method will use the return value of cellClass to allocate and initialize an NSCell of the correct type.

If you want to be able to change the type of cell that your subclass uses without changing the type that its superclass uses, override setCellClass: to store the NSCell subclass in a global variable, and modify cellClass to return that variable:

```
static id myStoredCellClass;

+ setCellClass:classId
{
    myStoredCellClass = classId;
}
+ (Class) cellClass
{
```

```
    return (myStoredCellClass ? myStoredCellClass : [MyCellSubclass
class]);
}
```

An `NSControl` subclass doesn't have to use an `NSCell` subclass to implement itself; `NSScroller` and `NSColorWell` are examples of `NSControl`s that don't. However, such subclasses have to take care of details that `NSCell` would otherwise handle. Specifically, they have to override methods designed to work with an `NSCell`. What's more, the lack of an `NSCell` means you can't make use of `NSMatrix`—a subclass of `NSControl` designed specifically for managing multi-cell arrays such as radio buttons.

Override the designated initializer (`initWithFrame:`) if you create a subclass of `NSControl` that performs its own initialization.

Method Types

Activity	Class Method
Initializing an NSControl object	– initWithFrame:
Setting the control's cell	+ cellClass + setCellClass: – cell – setCell:
Enabling and disabling the control	– isEnabled – setEnabled:
Indentifying the selected cell	– selectedCell – selectedTag
Setting the control's value	– doubleValue – floatValue – intValue – setDoubleValue: – setFloatValue: – setIntValue: – setNeedsDisplay – setStringValue: – stringValue
Interacting with other controls	– takeDoubleValueFrom: – takeFloatValueFrom: – takeIntValueFrom: – takeStringValueFrom:
Formatting text	– alignment – font – setAlignment: – setFont: – setFloatingPointFormat:left:right:
Managing the field editor	– abortEditing – currentEditor – validateEditing
Resizing the control	– calcSize – sizeToFit
Displaying the control and cell	– drawCell: – drawCellInside: – selectCell: – updateCell: – updateCellInside:

Activity	Class Method
Target and action	– action – isContinuous – sendAction:to: – sendActionOn: – setAction: – setContinuous: – setTarget: – target
Assigning a tag	– setTag: – tag
Tracking the mouse	– mouseDown: – ignoresMultiClick – setIgnoresMultiClick:
Methods Implemented by the Delegate	– control:didFailToFormatString: errorDescription: – control:didFailToValidatePartialString: errorDescription: – control:isValidObject: – control:textShouldBeginEditing: – control:textShouldEndEditing: – controlTextDidBeginEditing: – controlTextDidEndEditing: – controlTextDidChange:

Class Methods

`cellClass`

```
+ (Class)cellClass
```

Returns `nil`; overridden by subclasses. See also `setCellClass:`, `cell`, `setCell:`.

`setCellClass:`

```
+ (void)setCellClass:(Class)factoryId
```

Implemented by subclasses to set the `NSCell` class used. See also `cellClass`, `cell`, `setCell:`.

Instance Methods

abortEditing

```
- (BOOL)abortEditing
```

Terminates and discards any editing of text displayed by the receiving NSControl. Returns YES, or NO if no editing was going on in the receiving NSControl. This method doesn't redisplay the old value of the NSControl. See also currentEditor, validateEditing.

action

```
- (SEL)action
```

Returns the action message sent by the NSControl's NSCell, or the default action message for an NSControl with multiple NSCells (such as an NSMatrix or NSForm). To retrieve the action message, this method sends an action message to the NSCell. For NSControls with multiple NSCells, it's better to get the action message for a particular NSCell using:

```
someAction = [[theControl selectedCell] action];
```

See also isContinuous, sendAction:to:, sendActionOn:, setAction:, setContinuous:, setTarget:, target, action (NSCell).

alignment

```
- (NSTextAlignment)alignment
```

Returns the alignment of text in the control's cell (via the cell's alignment method). The return value can be one of the follow values:

- NSLeftTextAlignment
- NSLeftRightAlignment
- NSCenterTextAlignment
- NSJustifiedTextAlignment
- NSNaturalTextAlignment

See the "Text" section of the Application Kit's "Types and Constants" chapter for more information on text alignment. See also setAlignment:, font, setFont:, setFloatingPointFormat:left:right:.

calcSize

```
- (void)calcSize
```

Recomputes any internal sizing information for the NSControl, if necessary, by invoking its NSCell's calcDrawInfo: method. This method doesn't actually draw. It can be used for more sophisticated sizing operations as well for example, NSForm. calcSize is automatically invoked whenever the NSControl is displayed and something has changed; you need never invoke it.

cell

```
- (id)cell
```

Returns the control's NSCell. It is better to use selectedCell in the action method of the target of the NSControl, since an NSControl may have multiple NSCells. See also cellClass, setCellClass:, setCell:.

currentEditor

```
- (NSText *)currentEditor
```

If the receiving NSControl is being edited (that is, has an NSText object acting as its editor, and is the first responder in its NSWindow), this method returns the NSText object being used to perform that editing. If the NSControl isn't being edited, this method returns nil. See also abortEditing, validateEditing.

doubleValue

```
- (double)doubleValue
```

Returns the value of the NSControl's selected NSCell as a double-precision floating point number. If the NSControl contains many cells (for example NSMatrix), then the value of the currently selectedCell is returned. If the NSControl is in the process of editing the affected NSCell, then validateEditing is invoked before the value is extracted and returned. See also floatValue, intValue, setDoubleValue:, setFloatValue:, setIntValue:, setNeedsDisplay, setStringValue:.

drawCell:

```
- (void)drawCell:(NSCell *)aCell
```

If `aCell` is the cell used to implement this `NSControl`, then the `NSControl` is displayed. This method is provided primarily to support a consistent set of methods between `NSControl`s with single and multiple `NSCell`s, since an `NSControl` with multiple `NSCell`s needs to be able to draw a single `NSCell` at a time. See also `drawCellInside:`, `selectCell:`, `updateCell:`, `updateCellInside:`.

drawCellInside:

```
- (void)drawCellInside:(NSCell *)aCell
```

Draws the inside of a control (the area within a bezel or border). This method invokes cell's `drawInteriorWithFrame:inView:` method. `drawCellInside:` is used to provide a minimal update of the control when its value is changed. See also `drawCell:`.

floatValue

```
- (float)floatValue
```

Returns the value of the control's selected cell as a single-precision `float`. See also `doubleValue`.

font

```
- (NSFont *)font
```

Returns the `NSFont` used to draw text in the control's cell. See also `setFont:`, `alignment`.

ignoresMultiClick

```
- (BOOL)ignoresMultiClick
```

Returns YES if multiple clicks are ignored, and returns NO otherwise. By default, double-clicks (and higher order clicks) are treated the same as single clicks. You can use this method to “debounce” an NSControl, so that it won’t inadvertently send its action message twice when double-clicked. See also setIgnoresMultiClick:, mouseDown:.

initWithFrame:

```
- (id)initWithFrame:(NSRect)frameRect
```

Initializes and returns a new instance of NSControl, by setting frameRect as its frame rectangle. Since NSControl is an abstract class, messages to perform this method should appear only in subclass methods; that is, there should always be a more specific designated initializer for the subclass. initWithFrame: is the designated initializer for the NSControl class.

intValue

```
- (int)intValue
```

Returns the value of the control's selected cell as a int. See also doubleValue.

isContinuous

```
- (BOOL)isContinuous
```

Returns YES if the control’s NSCell continuously sends its action message to its target during mouse tracking. Returns NO otherwise. See also setContinuous:, action.

isEnabled

```
- (BOOL)isEnabled
```

Returns yes if the NSControl reacts to mouse events, and NO otherwise. See also setEnabled:.

mouseDown:

```
- (void)mouseDown:(NSEvent *)theEvent
```

Invoked when the mouse button goes down while the cursor is within the contro bounds. This method highlights the control's cell and sends it a `trackMouse:inRect:ofView:untilMouseUp:` message. Whenever the cell finishes tracking the mouse (for example, because the cursor has left the cell's bounds), the cell is unhighlighted. If the mouse button is still down and the cursor reenters the bounds, the cell is again highlighted and a new `trackMouse:inRect:ofView:untilMouseUp:` message is sent. This behavior repeats until the mouse button goes up.

selectCell:

```
- (void)selectCell:(NSCell *)aCell
```

If `aCell` is an `NSCell` of the receiving `NSControl` and is deselected, this method selects `aCell` and redraws the `NSControl`. See also `selectedCell`.

selectedCell

```
- (id)selectedCell
```

Returns the control's selected `NSCell`. The target of the `NSControl` should use this method when it wants to get the `NSCell` of the sending `NSControl`. Note that even though the `cell` method will return the same value for `NSControl`s with only a single `NSCell`, it is strongly suggested that this method be used since it will work for `NSControl`s with either a single or multiple `NSCell`s. See also `selectCell:`, `selectedTag`, `selectedCell` (`NSMatrix`).

selectedTag

```
- (int)selectedTag
```

Returns the tag of the control's selected cell. This is equivalent to:

```
myTag = [[theControl selectedCell] tag];
```

This method returns –1 if there is no selected `NSCell`. The cell's tag can be set with `NSActionCell`'s `setTag:` method. You should only use the `setTag:` and `tag` methods in conjunction with `viewWithTag:` (`NSView`). See also `selectedCell`.

sendAction:to:

```
- (BOOL)sendAction:(SEL)theAction to:(id)theTarget
```

Sends a `sendAction:to:from:` message to `NSApp` (the `NSApplication` object), which in turn sends a message to `theTarget` to perform `theAction`. `sendAction:to:from:` adds the `NSControl` as the `from:` argument. If `theAction` is `NULL`, no message is sent. `sendAction:to:` is invoked primarily (and indirectly) by `NSCell`'s `trackMouse:inRect:ofView:`.

If `theTarget` is `nil`, `NSApp` looks for an object that can respond to the message by following the responder chain, as detailed in the Class Description. This method returns `YES` if no object that responds to `theAction` could be found, beeps and returns `NO` if `NSApp` is in a modal event loop, and otherwise returns `NO`. See also `action`.

sendActionOn:

```
- (int)sendActionOn:(int)mask
```

Uses `mask` to record the events that cause `sendAction:to:` to be invoked during tracking of the mouse, which is performed in `NSCell`'s `trackMouse:inRect:ofView:`. `mask` can contain the following values:

- `NSLeftMouseUpMask`
- `NSLeftMouseDownMask`
- `NSLeftMouseDraggedMask`
- `NSPeriodicMask`

Returns the old event mask. See the Event Handling section of the Application Kit's Types and Constants chapter for more information on event mask values.

setAction:

```
- (void)setAction:(SEL)aSelector
```

Makes `aSelector` the `NSControl`'s action method. If `aSelector` is `NULL`, then no action messages will be sent from the `NSControl`. See also `action`, `setTarget:`.

setAlignment:

```
- (void)setAlignment:(NSTextAlignment)mode
```

Sets the alignment mode of the text in the NSControl's cell (or of all the NSControl's cells if it has more than one) to mode, and redraws the NSControl. mode should be one of the following values:

- NSLeftTextAlignment
- NSRightTextAlignment
- NSCenterTextAlignment
- NSJustifiedTextAlignment
- NSNaturalTextAlignment

See the "Text" section of the Application Kit's "Types and Constants" chapter for more information on text alignment. See also alignment.

setCell:

```
- (void)setCell:(NSCell *)aCell
```

Sets the control's NSCell to aCell. Use this method with care as it can irrevocably damage your NSControl; specifically, only use this method in initializers for subclasses of NSControl. See also cellClass, setCellClass:, cell.

setContinuous:

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

Sets whether the control's NSCell continuously sends its action to its target as the mouse is tracked. See also action, setContinuous: (NSCell).

setDoubleValue:

```
- (void)setDoubleValue:(double)aDouble
```

Sets the value of the NSControl's selected cell to aDouble (a double-precision floating point number). If the affected NSCell is being edited, that editing is aborted and the value being typed is discarded in favor of aDouble. If autodisplay is on, then the NSCell's inside the area within a bezel or border is redrawn. See also doubleValue.

setEnabled:

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

Sets whether the NSControl is active or not (that is, whether it tracks the mouse and sends its action to its target). If flag is NO, any editing is aborted. Redraws the entire NSControl if autodisplay is on. Subclasses may want to override this to redraw only a portion of the NSControl when the enabled state changes. See also isEnabled.

setFloatValue:

```
- (void)setFloatValue:(float)aFloat
```

Same as setDoubleValue:, but sets the control's selected cell's value to aFloat, a single-precision floating point number. See also doubleValue.

setFloatingPointFormat:left:right:

```
- (void)setFloatingPointFormat:(BOOL)autoRange
    left:(unsigned)leftDigits right:(unsigned)rightDigits
```

Sets the floating-point autoranging and display format for the control's cell, so that at most leftDigits are displayed to the left of the decimal point, and rightDigits to the right. If the NSControl has more than one NSCell, they're all affected. See the description of this method in the NSCell class specification for more detail. This method doesn't redraw the NSControl but marks it as needing redrawing, and affects only subsequent invocations of setFloatValue:. See also alignment.

setFont:

```
- (void)setFont:(NSFont *)fontObject
```

Sets the NSFont object used to draw the text (if any) in the NSControl's NSCell, or in all the NSCells if the NSControl has more than one. You only need to use this method if you don't want to use the user's default system font. Marks the cell as needing redrawing. See also alignment.

setIgnoresMultiClick:

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

Sets the NSControl to ignore multiple clicks if flag is YES. By default, double-clicks (and higher order clicks) are treated the same as single clicks. You can use this method to "debounce" an NSControl, so that it won't inadvertently send its action message twice when double-clicked. See also ignoresMultiClick, mouseDown:.

setIntValue:

```
- (void)setIntValue:(int)anInt
```

Sets the value of the control's selected cell to anInt, an integer. See also intValue, doubleValue.

setNeedsDisplay

```
- (void)setNeedsDisplay
```

Sets a flag that informs the control that it's state has changed, and that the control needs redrawing.

setStringValue:

```
- (void)setStringValue:(NSString *)aString
```

Sets the value of the control's selected cell to aString, a string. See also stringValue, doubleValue.

setTag:

```
- (void)setTag:(int)anInt
```

Sets the control's tag to anInt. See also tag.

setTarget:

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

Sets the NSControl's action-message target to anObject. If anObject is nil, then when an action message is sent, NSApp looks for an object that can respond to the message by following the responder chain, as detailed in the Class Description. See also target, action.

sizeToFit

```
- (void)sizeToFit
```

Changes the width and the height of the NSControl's frame so that they are the minimum needed to contain the NSCell. If the NSControl has more than one NSCell, then you must override this method. See also calcSize.

stringValue

```
- (NSString *)stringValue
```

Returns the value of the control's selected cell as an NSString. If the NSControl is in the process of editing the affected NSCell, then validateEditing is invoked before the value is extracted and returned. See also setStringValue:, doubleValue.

tag

```
- (int)tag
```

Returns the receiving control's tag. See also setTag:, selectedTag.

takeDoubleValueFrom:

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

Sets the receiving NSControl's selected cell to the value obtained by sending a doubleValue message to sender. This method can be used in action messages between NSControls. It permits one NSControl (the sender) to affect the value of another NSControl (the receiver) by invoking this method in an action message to the receiver. For example, an NSTextField can be made the target of an NSSlider. Whenever the slider is moved, it will send a takeDoubleValueFrom: message to the NSTextField. The NSTextField will then get the slider's floating-point value, turn it into a text string, and display it, thus tracking the value of the slider. See also setDoubleValue:, doubleValue, takeFloatValueFrom:, takeIntValueFrom:, takeStringValueFrom:.

takeFloatValueFrom:

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

Sets the receiving control's selected cell to the value obtained by sending a `floatValue` message to `sender`. See `takeDoubleValueFrom:` for an example. See also `setFloatValue:`, `floatValue`.

takeIntValueFrom:

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

Sets the receiving control's selected cell to the value obtained by sending a `intValue` message to `sender`. See `takeDoubleValueFrom:` for an example. See also `setIntValue:`, `intValue`.

takeStringValueFrom:

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

Sets the receiving `NSControl`'s selected cell to the value obtained by sending a `stringValue` message to `sender`. See `takeDoubleValueFrom:` for an example. See also `setStringValue:`.

target

```
- (id)target
```

Returns the target for the action message of the control's cell. If `nil`, then any action messages sent by the `NSControl` will be sent up the responder chain, as detailed in the Class Description. See also `setTarget:`, `target` (`NSCell`, `NSActionCell`).

updateCell:

```
- (void)updateCell:(NSCell *)aCell
```

Redisplays `aCell` or marks it for redisplay. See also `updateCellInside:`, `drawCell:`.

updateCellInside:

```
- (void)updateCellInside:(NSCell *)aCell
```

Redisplays the inside of `aCell` or marks it for redisplay. See also `updateCell:`, `drawCell:`.

validateEditing

```
- (void)validateEditing
```

Causes the value of the NSControl's selected cell to be set to the value of the field being edited, if any. "Being edited" does not necessarily mean that a user is typing; if a field (for example, an NSTextField object) has the application's global NSText object acting in its place as first responder, then the field is considered as being edited. This method is invoked automatically from stringValue, intValue, and other similar methods, so that a partially edited field's actual value will be correctly returned by those methods.

Methods Implemented by the Delegate

Note – NSControl itself doesn't have a delegate. These delegate methods are declared in NSControl.h but are intended for subclasses, such as NSTextField and NSMatrix, that do have delegates and that allow text editing.

control:didFailToFormatString: errorDescription:

```
- (BOOL)control:(NSControl *)control
    didFailToFormatString:(NSString *)string
    errorDescription:(NSString *)error
```

Implement this method to respond to string to cell object conversions. See NSFormatter.

control:didFailToValidatePartialString: errorDescription:

```
- (void)control:(NSControl *)control
    didFailToValidatePartialString:(NSString *)string
    errorDescription:(NSString *)error
```

Implement this method to respond to partial string validation failures. See NSFormatter.

control:isValidObject:

```
- (BOOL)control:(NSControl *)control isValidObject:(id)obj
```

This method is invoked when the cursor leaves a cell (that is, the associated control relinquishes first-responder status), but before the string value of the cell's object is displayed. Implementations should return YES to allow display of the string and NO to reject display and return the cursor to the cell. See the NSFormatter class description for an example.

control:textShouldBeginEditing:

```
- (BOOL)control:(NSControl *)control
    textShouldBeginEditing:(NSText *)fieldEditor
```

Sent directly by control to the delegate; returns YES if the control should be allowed to start editing the text.

control:textShouldEndEditing:

```
- (BOOL)control:(NSControl *)control
    textShouldEndEditing:(NSText *)fieldEditor
```

Sent directly by control to the delegate; returns YES if the control should be allowed to end its edit session.

controlTextDidBeginEditing:

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

Sent by the default notification center to the delegate; aNotification is always NSControlTextDidBeginEditingNotification. If the delegate implements this method, it's automatically registered to receive this notification.

controlTextDidEndEditing:

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

Sent by the default notification center to the delegate; aNotification is always NSControlTextDidEndEditingNotification. If the delegate implements this method, it's automatically registered to receive this notification.

controlTextDidChange:

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

Sent by the default notification center to the delegate; `aNotification` is always `NSControlTextDidChangeNotification`. If the delegate implements this method, it's automatically registered to receive this notification.

NSCStringText

Inherits From:	NSText : NSView : NSResponder : NSObject
Conforms To:	NSChangeSpelling (NSText) NSIgnoreMisspelledWords (NSText) NSCoding (NSResponder) NSObject (NSObject)
Declared In:	AppKit / NSCStringText.h

Class Description

The `NSCStringText` class declares the programmatic interface to objects that manage text using eight-bit character encodings. The encoding is the same as the default C string encoding provided by `defaultCStringEncoding` in the `NSString` class. `NSCStringText` can be used in situations where backwards compatibility with the detailed interfaces of the NeXTSTEP Text object is important. Applications that can use the interface of `NSText` should do so.

The `NSCStringText` class is unlike most other classes in the Application Kit in its complexity and range of features. One of its design goals is to provide a comprehensive set of text-handling features so that you'll rarely need to create a subclass. An `NSCStringText` object can (among other things):

- Control the color of its text and background.
- Control the font and layout characteristics of its text.
- Control whether text is editable.
- Wrap text on a word or character basis.
- Write text to, or read it from, a file, as either Rich Text Format (RTF) or plain ASCII data.

- Display graphic images within its text.
- Communicate with other applications through the Services menu.
- Let another object, the delegate, dynamically control its properties.
- Let the user copy and paste text within and between applications.
- Let the user copy and paste font and format information between `NSCStringText` objects.
- Let the user check the spelling of words in its text.
- Let the user control the format of paragraphs by manipulating a ruler.

`NSCStringText` can deal only with eight-bit characters. Therefore, it is not able to deal with Unicode character sets, and `NSCStringText` can't be fully internationalized.

Plain and Rich NSCStringText Objects

When you create an `NSCStringText` object directly, it allows by default only one font, line height, text color, and paragraph format for the entire text. You can set the default font used by new `NSCStringText` instances by sending the `NSCStringText` class object a `setDefaultFont:` message. Once an `NSCStringText` object is created, you can alter its global settings using methods such as `setFont:`, `setLineHeight:`, `setTextGray:`, and `setAlignment:`. For convenience, such an `NSCStringText` object will be called a *plain* `NSCStringText` object.

To allow multiple values for these attributes, you must send the `NSCStringText` object a `setRichText:YES` message. An `NSCStringText` object that allows multiple fonts also allows multiple paragraph formats, line heights, and so on. For convenience, such an `NSCStringText` object will be called a *rich* `NSCStringText` object.

A rich `NSCStringText` object can use RTF as an interchange format. Not all RTF control words are supported: On input, an `NSCStringText` object ignores any control word it doesn't recognize; some of those control words it can read and interpret it won't write out. Refer to the Class Description of `NSText` for a list of the RTF control words that an `NSCStringText` object recognizes.

Note – An NSCStringText object writes eight-bit characters in the default C string encoding, which differs somewhat from the ANSI character set.

In an NSCStringText object, each sequence of characters having the same attributes is called a *run*. A plain NSCStringText object has only one run for the entire text. A rich NSCStringText object can have multiple runs. Methods such as setSelFont: and setSelColor: let you programmatically modify the attributes of the selected sequence of characters in a rich NSCStringText object. As discussed the following, the user can set these attributes using the Font panel and the ruler.

NSCStringText objects are designed to work closely with various objects and services. Some of these—such as the delegate or an embedded graphic object—require a degree of programming on your part. Others—such as the Font panel, spelling checker, ruler, and Services menu—take no effort other than deciding whether the service should be enabled or disabled. The following sections discuss these interrelationships.

Notifying the NSCStringText Object's Delegate

Many of an NSCStringText object's actions can be controlled through an associated object, the NSCStringText object's delegate. If it implements any of the following methods, the delegate receives the corresponding message at the appropriate time:

- textWillResize:
- textDidResize:oldBounds:
- textWillSetSel:toFont:
- textWillConvert:fromFont:toFont:
- textWillStartReadingRichText:
- textWillFinishReadingRichText:
- textWillWrite:
- textDidRead:paperSize:

So, for example, if the delegate implements the textWillConvert:fromFont:toFont: method, it will receive notification upon the user's first attempt to change the font of the text. Moreover, depending on the method's return value, the delegate can either allow or

prohibit changes to the text. See the “Methods Implemented by the Delegate” section. The delegate can be any object you choose, and one delegate can control multiple `NSCStringText` objects.

Adding Graphics to the Text

A rich `NSCStringText` object allows graphics to be embedded in the text. Each graphic is treated as a single (possibly large) “character”: The text's line height and character placement are adjusted to accommodate the graphic “character.” Graphics are embedded in the text in either of two ways: programmatically or directly through user actions.

In the programmatic approach, you add an object—generally a subclass of `NSCell`—to the text. This object manages the graphic image by drawing it when appropriate. Although `NSCell` subclasses are commonly used, the only requirement is that the embedded object responds to these messages—see the “Methods Implemented by an Embedded Graphic Object” section for more information:

- `highlight:withFrame:inView:`
- `drawWithFrame:inView:`
- `trackMouse:inRect:ofView:untilMouseUp:`
- `cellSize:`
- `readRichText:forView:`
- `richTextforView:`

You place the graphic object in the text by sending the `NSCStringText` object a `replaceSelWithCell:` message.

An `NSCStringText` object displays a graphic in its text by sending the managing object a `drawWithFrame:inView:` message. To record the graphic to a file or to the pasteboard, the `NSCStringText` object sends the managing object a `richTextforView:` message. The object must then write an RTF control word along with any data such as the path of a TIFF file containing its image data it might need to recreate its image. To reestablish the text containing the graphic image from RTF data, an `NSCStringText` object must know which class to associate with particular RTF control words. You associate a control word with a class object by sending the `NSCStringText` class object a `registerDirective:forClass:` message. Thereafter, whenever an `NSCStringText` object finds the registered control word in the RTF data being read from a file or the pasteboard, it will create a new instance of the class and send the object a `readRichText:forView:` message.

An alternate means of adding an image to the text is for the user to drag an EPS or TIFF file icon directly into an NSCStringText object. The NSCStringText object automatically creates a graphic object to manage the display of the image. This feature requires a rich NSCStringText object that has been configured to receive dragged images. See the setImportsGraphics: method.

Images that have been imported in this way can be written as RTFD documents. Programmatic creation of RTFD documents is not supported in this version of OpenStep. RTFD documents use a file package, or directory, to store the components of the document (the "D" stands for "directory"). The file package has the name of the document plus a .rtfd extension. The file package always contains a file called TXT.rtf for the text of the document, and one or more TIFF or EPS files for the images. An NSCStringText object can transfer information in an RTFD document to a file and read it from a file. See the writeRTFDToFile:atomically: and readRTFDFromFile: methods in the NSText methods.

Cooperating with Other Objects and Services

NSCStringText objects are designed to work with the Application Kit's font conversion system. By default, an NSCStringText object keeps the Font panel updated with the font of the current selection. It also changes the font of the selection (for a rich NSCStringText object) or of the entire text (for a default NSCStringText object) to reflect the user's choices in the Font panel or menu. To disconnect an NSCStringText object from this service, send it a setUsesFontPanel:NO message.

If an NSCStringText object is a subview of an NSScrollView, it can cooperate with the NSScrollView to display and update a ruler that displays formatting information. The NSScrollView retiles its subviews to make room for the ruler, and the NSCStringText object updates the ruler with the format information of the paragraph containing the selection. The toggleRuler: method controls the display of this ruler. Users can modify paragraph formats by manipulating the components of the ruler.

By means of the Services menu, an NSCStringText object can make use of facilities outside the scope of its own application. By default, an NSCStringText object registers with the services system that it can send and receive RTF and plain ASCII data. If the application containing the NSCStringText object has a Services menu, a menu item is added for each

service provider that can accept or return these formats. To prevent `NSCStringText` objects from registering for services, send the `NSCStringText` class object an `excludeFromServicesMenu:YES` message before any `NSCStringText` objects are created.

Coordinates and sizes mentioned in the method descriptions that follow are in PostScript units—1/72 of an inch.

Method Types

Activity	Class Method
Initializing a new NSCStringText object	– initWithFrame:text:alignment:
Modifying the frame rectangle	– resizeTextWithOldBounds:maxRect:
Managing global characteristics	– setImportsGraphics: – setRichText:
Laying out the text	– calcLine – changeTabStopAt:to: – charWrap – defaultParagraphStyle – descentLine – getMarginLeft:right:top:bottom: – getMinWidth:minHeight:maxWidth:maxHeight: – lineHeight – paragraphStyleForFont:alignment: – setCharWrap: – setDescentLine: – setLineHeight: – setMarginLeft:right:top:bottom: – setNoWrap – setParagraphStyle: – setSelProp:to:
Reporting line and position	– lineFromPosition: – positionFromLine:
Reading and writing text	– finishReadingRichText – firstTextBlock – paragraphRect:start:end: – startReadingRichText
Editing text	– clear: – hideCaret – showCaret

Activity	Class Method
Managing the selection	– getSelectionStart:end: – replaceSel: – replaceSel:length: – replaceSel:length:runs: – scrollSelToVisible – selectError – selectNull – setSelectionStart:end: – selectText:
Setting the font	+ defaultFont + setDefaultFont: – setFont:paragraphStyle: – setSelFont: – setSelFont:paragraphStyle: – setSelFontFamily: – setSelFontSize: – setSelFontStyle:
Finding text	– findText:ignoreCase:backwards:wrap:
Modifying graphic attributes	– runColor: – selColor – setSelColor:
Reusing an NSCStringText object	– renewFont:text:frame:tag: – renewFont:size:style:text:frame:tag: – renewRuns:text:frame:tag:
Setting window attributes	– isRetainedWhileDrawing – setRetainedWhileDrawing:
Assigning a tag	– setTag: – tag
Handling event messages	– becomeKeyWindow – moveCaret: – resignKeyWindow
Displaying graphics within the text	+ registerDirective:forClass: – locationOfCell: – replaceSelWithCell: – setLocation:ofCell:
Using the services menu and the pasteboard	+ excludeFromServicesMenu: – readSelectionFromPasteboard: – validRequestorForSendType:returnType: – writeSelectionToPasteboard:types:

Activity	Class Method
Setting tables and functions	– breakTable – charCategoryTable – charFilter – clickTable – drawFunc – postSelSmartTable – preSelSmartTable – scanFunc – setBreakTable: – setCharCategoryTable: – setCharFilter: – setClickTable: – setDrawFunc: – setPostSelSmartTable: – setPostSelSmartTable: – setScanFunc: – setTextFilter: – textFilter
Printing	– adjustPageHeightNew:top:bottom:limit:
Implemented by an embedded graphic object	– cellSize – drawWithFrame:inView: – highlight:withFrame:inView: – readRichText:forView:: – richTextForView: – trackMouse:inRect:ofView:untilMouseUp:
Comparing methods	– cStringTextInternalState
Methods Implemented by the Delegate	– textDidRead:paperSize: – textDidResize:oldBounds: – textWillConvert:fromFont:toFont: – textWillFinishReadingRichText: – textWillResize: – textWillSetSel:toFont: – textWillStartReadingRichText: – textWillWrite:

Class Methods

`defaultFont`

```
+ (NSFont *)defaultFont
```

Returns the default font object tfor NSCStringText objects. Unless you've changed the default font by sending a setDefaultFont: message, this method returns a font object for a 12-point Helvetica font with a flipped font matrix. See also setDefaultFont:, setFont:paragraphStyle:, setSelFont:.

excludeFromServicesMenu:

```
+ excludeFromServicesMenu:(BOOL)flag
```

Controls whether NSCStringText objects will communicate with interapplication services through the Services menu. By default, as each new NSCStringText instance is initialized, it registers with the NSApplication object that it's capable of sending and receiving the pasteboard types identified by NSStringPboardType and NSRTFPboardType. If you want to prevent your applications NSCStringText objects from registering for services that can receive and send these types, send the text class object an excludeFromServicesMenu:YES message. If, for example, your application displays text but doesn't have editable text fields, you might use this method.

Send an excludeFromServicesMenu: message early in the execution of your application, either before sending the NSApplication object a run message or in the NSApplication delegate's appWillFinishLaunching: method. See also readSelectionFromPasteboard:, writeSelectionToPasteboard:types:, validRequestorForSendType:returnType:.

registerDirective:forClass:

```
+ registerDirective:(NSString *)directive forClass:class
```

Creates an association in the NSCStringText class object between the RTF control word directive and class, a class object (usually NSCell or a subclass). Thereafter, when a text object encounters directive while reading a stream of RTF text, it creates a new class instance. The new instance is sent a readRichText:forView: message to let it read its image data from the RTF text. Conversely, when a text object is writing RTF data and encounters an object of the class class, the text object sends the object a richTextForView: message to let it record its representation in the RTF text.

This method is instrumental in enabling a text object to read, display, and write an image within a text stream. An object of the `class` class must implement these methods:

- `cellSize:`
- `drawWithFrame:inView:`
- `highlight:withFrame:inView:`
- `readRichText:forView:`
- `richTextForView:`
- `trackMouse:inRect:ofView:untilMouseUp:`

See the "Methods Implemented by an Embedded Graphic Object" section for more information on these methods.

`setDefaultFont:`

```
+ (void)setDefaultFont:(NSFont *)anObject
```

Sets the default font for the `NSCStringText` class object. Since an `NSCStringText` object uses a flipped coordinate system, make sure the font object you specify uses a matrix that flips the y-axis of the characters. See also `defaultFont`.

Instance Methods

`adjustPageHeightNew:top:bottom:limit:`

```
- (void)adjustPageHeightNew:(float *)newBottom top:(float)oldTop
    bottom:(float)oldBottom limit:(float)bottomLimit
```

Assists with automatic pagination of text. During automatic pagination, this method is performed to help lay a grid of pages over the top-level view being printed. `newBottom` is passed in undefined and must be set by this method. `oldTop` and `oldBottom` are the current values for the horizontal strip being created. `bottomLimit` is the topmost value `newBottom` can be set to. If this limit is broken, the new value is ignored. By default, this method tries to prevent the view from being cut in two. All parameters are in the view's own coordinate system.

becomeKeyWindow

```
- (void)becomeKeyWindow
```

Activates the caret if the selection has a width of 0. This message is sent by an application's NSWindow object, which, upon receiving a mouse-down event, sends a becomeKeyWindow message to the first responder. You should never directly send this message to a text object. See also resignKeyWindow, moveCaret:.

breakTable

```
- (const NSFSM *)breakTable
```

Returns a pointer to the break table, the finite-state machine table that the NSCStringText object uses to determine word boundaries. See also setBreakTable:, charCategoryTable, clickTable, postSelSmartTable, preSelSmartTable.

calcLine

```
- (int)calcLine
```

Calculates the array of line breaks for the text. The text will then be redrawn if autodisplay is set. This message should be sent after the text object's frame is changed. These methods send a calcLine message as part of their implementation:

- initWithFrame:text:alignment:
- renewFont:text:frame:tag:
- renewRuns:text:frame:tag:
- setFont:paragraphStyle:
- setText:range:
- renewFont:size:style:text:frame:tag:
- setFont: (see NSText)
- setParagraphStyle:
- setText: (see NSText)

cellSize

```
- (NSSize)cellSize
```

Responds to a message from the text object by providing the graphic object's width and height. The text object uses this information to adjust character placement and line height to accommodate the display of the graphic object in the text. See also `cellSize` (`NSCell`).

changeTabStopAt:to:

```
- (BOOL)changeTabStopAt:(float)oldX to:(float)newX
```

Moves the tab stop from the receiving text object's x coordinate `oldX` to the coordinate `newX`. For a plain Text object, all paragraphs are affected. For a rich text object, only those paragraphs marked by the selection are affected. The text is rewrapped and redrawn. Returns `YES` upon successful completion.

charCategoryTable

```
- (const unsigned char *)charCategoryTable
```

Returns a pointer to the character category table, the table that maps ASCII characters to character categories. See also `setCharCategoryTable:`.

charFilter

```
- (NSCharFilterFunc)charFilter
```

Returns the current character filter function (the function that analyzes each character the user enters). By default, this function is `NSEditorFilter()`. See also `setCharFilter:`.

charWrap

```
- (BOOL)charWrap
```

Returns a flag indicating how words whose length exceeds the line length should be treated. If `YES` is returned, long words are wrapped on a character basis. If `NO` is returned, long words are truncated at the frame boundary. See also `setCharWrap:`.

clear:

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

Deletes the selected text.

clickTable

```
- (const NSFSM *)clickTable
```

Returns a pointer to the click table, the finite-state machine table that defines word boundaries for double-click selection. See also `setClickTable:`.

cStringTextInternalState

```
- (NSCStringTextInternalState *)cStringTextInternalState
```

Returns a structure that represents the instance variables of the NSCStringText object. The structure is defined in `appkit/NSCStringText.h`, and in the "Types and Constants" chapter of the Application Kit documentation. Note that this method is provided for applications that really must depend on changing the values of an `NSCStringText` object's instance variables.

defaultParagraphStyle

```
- (void *)defaultParagraphStyle
```

Returns the default paragraph style. The pointer that's returned refers to an `NSTextStyle` structure. The fields of this structure contain default paragraph indentation, alignment, line height, descent line, and tab information. The text object's default values for these attributes can be altered using methods such as `setParagraphStyle:`, `setLineHeight:`, and `setDescentLine:`.

descentLine

```
- (float)descentLine
```

Returns the distance from the bottom of a line of text to the base line of the text. See also `setDescentLine:`.

drawFunc

```
- (NSTextFunc)drawFunc
```

Returns the current draw function, the function that's called to draw each line of text. See also `setDrawFunc:`.

drawWithFrame:inView:

```
- (void)drawWithFrame:(NSRect)cellFrame inView:(NSView
*)controlView
```

The embedded object draws itself, including frame, in `cellFrame` within `controlView`. Don't send a this message directly, although you may want to override this method to change the way an embedded text object draws itself.

findText:ignoreCase:backwards:wrap:

```
- (BOOL)findText:(NSString *)textPattern
ignoreCase:(BOOL)ignoreCase
    backwards:(BOOL)backwards wrap:(BOOL)wrap
```

Searches for `textPattern` in the text, starting at the insertion point. `ignoreCase` instructs the search to disregard case; `backwards` means search backwards; `wrap` means that when the search reaches the beginning or end of the text (depending on the direction), it should continue by wrapping to the end or beginning of the text.

finishReadingRichText

```
- (void)finishReadingRichText
```

Sent after the `NSCStringText` object reads RTF data, this message notifies the text object that it has finished reading RTF data. The text object responds by sending its delegate a `textWillFinishReadingRichText:` message, assuming there is a delegate and it responds to this message. The delegate can then perform any required cleanup. Alternatively, a subclass could put these cleanup routines in its own implementation of this method. See also `startReadingRichText`, `firstTextBlock`.

firstTextBlock

```
- (NSTextBlock *)firstTextBlock
```

Returns a pointer to the first text block in the NSCStringText object. You can traverse this head of the linked list of text blocks to read the contents of the text object. See also startReadingRichText, paragraphRect:start:end:.

getMarginLeft:right:top:bottom:

```
- (void)getMarginLeft:(float *)leftMargin
    right:(float *)rightMargin top:(float *)topMargin
    bottom:(float *)bottomMargin
```

Calculates the dimensions of the text object's margins and returns by reference these values in its four arguments. See also setMarginLeft:right:top:bottom:.

getMinWidth:minHeight:maxWidth:maxHeight:

```
- (void)getMinWidth:(float *)width minHeight:(float *)height
    maxWidth:(float)widthMax maxHeight:(float)heightMax
```

Given the widthMax and heightMax (width and height maximums), this method calculates the minimum area needed to display the text and returns width and height by reference. This method doesn't rewrap the text. To get the absolute minimum dimensions of the text, send this message only after sending a calcLine message.

getSelectionStart:end:

```
- (void)getSelectionStart:(NSSelPt *)start end:(NSSelPt *)end
```

Copies the starting and ending character positions of the selection into the addresses referred to by start and end. start points to the beginning of the selection; end points to the end of the selection. See also setSelectionStart:end:.

hideCaret

```
- (void)hideCaret
```

Removes the caret from the text. The text object sends itself hideCaret messages whenever the display of the caret would be inappropriate; you rarely need to send a hideCaret message directly. Occasions when the hideCaret

message is sent include whenever the text object receives a `resignKeyWindow`, `mouseDown:` (`NSObject`), or `keyDown:` (`NSObject`, `NSWindow`) message.

highlight:withFrame:inView:

```
- (void)highlight:(BOOL)flag withFrame:(NSRect)cellFrame
    inView:(NSView *)controlView
```

Upon receiving this message, the embedded object highlights or unhighlights itself with `cellFrame` of `controlView`. If `flag` is `YES`, this method should draw the graphic object in its highlighted state; if `NO`, it should draw the graphic object in its normal state. See the `NSCell` class specification for one implementation of this method. See also `drawWithFrame:inView:`.

initWithFrame:text:alignment:

```
- (id)initWithFrame:(NSRect)frameRect text:(NSString *)theText
    alignment:(NSTextAlignment)mode
```

Returns a new `NSCStringText` object at `frameRect`, initialized with the contents of `theText` and with `mode` alignment. `mode` can be one of the following values:

- `NSLeftTextAlignment`
- `NSRightTextAlignment`
- `NSCenterTextAlignment`
- `NSJustifiedTextAlignment`
- `NSNaturalTextAlignment`

This method is the designated initializer for text objects. If you derive a subclass, your subclass's designated initializer must mantain the initializer chain by sending a message to `super` to invoke this method.

The text object returned by this method uses the class object's default font and uses `NSEditorFilter()` as its character filter. It wraps words whose length exceeds the line length. It sets its view properties to draw in its superview, to be flipped, and to be transparent. For more efficient editing, you can send a `setOpaque:` (`NSImageRep`) message to make the text object opaque.

Text editing is designed to work in buffered windows only. In a nonretained or retained window, editing text in a text object causes flickering. However, to get better drawing performance without causing flickering during editing, see `setRetainedWhileDrawing:`.

isRetainedWhileDrawing

```
- (BOOL)isRetainedWhileDrawing
```

Returns `YES` if the text object automatically changes its window's buffering type from buffered to retained whenever it redraws itself, and returns `NO` if not. See also `setRetainedWhileDrawing:`.

lineFromPosition:

```
- (int)lineFromPosition:(int)position
```

Returns the line number that contains the character at `position`. See also `positionFromLine:`.

lineHeight

```
- (float)lineHeight
```

Returns height of a line of text. See also `setLineHeight:`.

locationOfCell:

```
- (NSPoint)locationOfCell:(NSCell *)cell
```

Returns the x and y coordinates of `cell`. The coordinates are in the text object's coordinate system. `cell` is an `NSCell` object that's displayed as part of the text. See also `NSPoint`.

moveCaret:

```
- (void)moveCaret:(unsigned short)theKey
```

Moves the caret either left, right, up, or down if `theKey` is `NSLeftTextMovement`, `NSRightTextMovement`, `NSUpTextMovement`, or `NSDownTextMovement`. If `theKey` isn't one of these four values, the caret doesn't move. See also `hideCaret`, `showCaret`.

paragraphRect:start:end:

```
- (NSRect)paragraphRect:(int)paraNumber start:(int *)startPos
    end:(int *)endPos
```

Returns the location and size of a paragraph identified by `paraNumber`; also returns the starting and ending character positions by reference. A paragraph ends in a return character; the first paragraph is paragraph 0, the second is paragraph 1, and so on. See also `firstTextBlock`.

paragraphStyleForFont:alignment:

```
- (void *)paragraphStyleForFont:(NSFont *)fontId
    alignment:(int)alignment
```

Recalculates the paragraph style based on new font `fontId` and `alignment`. The text object sends this message for you after its font has been changed; you will rarely need to send this message directly. Returns a pointer to an `NSTextStyle` structure that describes the default style. See also `defaultParagraphStyle`.

positionFromLine:

```
- (int)positionFromLine:(int)line
```

Returns the character position of the line numbered `line`. Each line is terminated by a Return character, and the first line in a text object is line 1. To find the length of a line, you can send this message with two successive lines, and use the difference of the two to get the line length. See also `lineFromPosition:`.

postSelSmartTable

```
- (const unsigned char *)postSelSmartTable
```

Returns a pointer to the table that specifies which characters on the right end of a selection are treated as equivalent to a space character. See also `setPostSelSmartTable:`, `preSelSmartTable`.

preSelSmartTable

```
- (const unsigned char *)preSelSmartTable
```

Returns a pointer to the table that specifies which characters on the left end of a selection are treated as equivalent to a space character. See also `setPreSelSmartTable:`, `postSelSmartTable`.

`readRichText:forView:`

```
- (void)readRichText:(NSString *)stringObject forView:(NSView
*)view
```

Responds to a message sent by the text object when it encounters an RTF control word that's associated with the embedded graphic object's class (see `registerDirective:forClass:`). The text object passes its `id` as the `view` argument. See also `richTextForView:`.

`readSelectionFromPasteboard:`

```
- (BOOL)readSelectionFromPasteboard:(NSPasteboard *)pboard
```

Replaces the current selection with data from pasteboard `pboard`. When the user chooses a command in the Services menu, a `writeSelectionToPasteboard:types:` message is sent to the first responder. That message is followed by a `readSelectionFromPasteboard:` message, if the command requires the requesting application to replace its selection with data from the service provider. See also `writeSelectionToPasteboard:types:`, `validRequestorForSendType:returnType:`.

`renewFont:text:frame:tag:`

```
- (void)renewFont:(NSFont *)newFontObj text:(NSString *)newText
    frame:(NSRect)newFrame tag:(int)newTag
```

Resets the `NSCStringText` object to draw different text `newText` in font `newFontObj` within frame `newFrame`. `newTag` sets a text object's tag. If `newText` is `NULL`, the new text is the same as the previous text. This method is a convenient cover for the `renewRuns:text:frame:tag:` method. See also `renewFont:size:style:text:frame:tag:`.

renewFont:size:style:text:frame:tag:

```
- (void)renewFont:(NSString *)newFontName size:(float)newFontSize
    style:(int)newFontStyle text:(NSString *)newText
    frame:(NSRect)newFrame tag:(int)newTag
```

Resets the NSCStringText object to draw different text newText in the font identified by newFontName, newFontSize, and newFontStyle. Drawing occurs within frame newFrame. This method is a convenient cover for the renewRuns:text:frame:tag: method. See also renewFont:text:frame:tag:.

renewRuns:text:frame:tag:

```
- (void)renewRuns:(NSRunArray *)newRuns text:(NSString *)newText
    frame:(NSRect)newFrame tag:(int)newTag
```

Resets a text object so that it can be reused to draw or edit another piece of text. If newRuns is NULL, the new text uses the same runs as the previous text. If newText is NULL, the new text is the same as the previous text. newTag sets a text object's tag. See also renewFont:size:style:text:frame:tag:.

replaceSel:

```
- (void)replaceSel:(NSString *)aString
```

Replaces the current selection with text from aString, a null-terminated character string, and then rewraps and redisplays the text. See also replaceSel:length:.

replaceSel:length:

```
- (void)replaceSel:(NSString *)aString length:(int)length
```

Replaces the selection with length bytes of aString.

Replaces the current selection with length bytes of aString, and then rewraps and redisplays the text. See also replaceSel:, replaceSel:length:runs:.

replaceSel:length:runs:

```
- (void)replaceSel:(NSString *)aString length:(int)length
    runs:(NSRunArray *)insertRuns
```

Replaces the selection with `length` bytes of `aString`. `insertRuns` is a pointer to the current run in the run array. After replacing the selection, this method rewraps and redisplays the text. See also `replaceSel:`, `replaceSel:length:`.

replaceSelWithCell:

```
- (void)replaceSelWithCell:(NSCell *)cell
```

Replaces the current selection with the image provided by `cell`. This method works only with rich text objects. The image is treated like a single character. Its height and width are determined by sending the cell a `cellSize:` message. The height determines the line height of the line containing the image, and the width sets the character placement in the line. The image is drawn by sending the cell a `drawWithFrame:inView:` message. After receiving a `replaceSelWithCell:` message, a text object rewraps and redisplays its contents. See `replaceSel:`.

resignKeyWindow

```
- (void)resignKeyWindow
```

Deactivates the caret when the text object's window ceases to be the key window. A window, before it ceases to be the application's key window, sends this message to its first responder. Never directly send this message to a text object. See also `becomeKeyWindow`.

resizeTextWithOldBounds:maxRect:

```
- (void)resizeTextWithOldBounds:(NSRect)oldBounds
    maxRect:(NSRect)maxRect
```

Used by the `NSCStringText` object to resize and redisplay itself, after the text object's frame has changed in response to editing. Don't send this message directly, but you can override it.

richTextForView:

```
- (NSString *)richTextForView:(NSView *)view
```

Causes the embedded object to store its RTF representation within view as a string object and returns it. See also `readRichText:forView:`.

runColor:

```
- (NSColor *)runColor:(NSRun *)run
```

Returns the color of the specified text `run`. By definition, a run can have no more than one color. See also `selColor`, `NSColor`.

scanFunc

```
- (NSTextFunc)scanFunc
```

Returns the scan function, the function that calculates the contents of each line of text given the line width, font size, text alignment, and other factors. `NSScanALine()` is the default scan function. See also `setScanFunc:`, `drawFunc`.

scrollSelToVisible

```
- (void)scrollSelToVisible
```

Scrolls the text so that the current selection is visible within the frame rectangle. This method works by invoking the `scrollRectToVisible:` method (`NSView`).

selColor

```
- (NSColor *)selColor
```

Returns the color of the selected text. See also `setSelColor:`.

selectError

```
- (void)selectError
```

Makes the entire text the selection and highlights it. The text object applies this method if the delegate requires the text object to maintain its status as the first responder. You rarely need to send this message directly, although you may want to override it. To highlight a portion of the text, use `setSelectionStart:end:`. See also `selectNull`.

selectNull

```
- (void)selectNull
```

Removes the selection and makes the highlighting (or caret, if the selection is zero-length) disappear. The text object's delegate isn't notified of the change. The text object sends a this message whenever it needs to end the current selection but retain its status as the first responder; you rarely need to override this method or send `selectNull` messages directly. See also `selectError`.

selectText:

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

Attempts to make a text object the first responder and, if successful, then selects all of its text. See also `selectError`, `setSelectionStart:end:`.

setBreakTable:

```
- (void)setBreakTable:(const NSFSM *)aTable
```

Sets the break table, the finite-state machine table that the text object uses to determine word boundaries. See also `breakTable`.

setCharCategoryTable:

```
- (void)setCharCategoryTable:(const unsigned char *)aTable
```

Sets the table that maps ASCII characters to character categories used in the word wrap or click tables. See also `charCategoryTable`.

setCharFilter:

```
- (void)setCharFilter:(NSCharFilterFunc)aFunction
```

Sets the character filter function to aFunction. This function analyzes each character the user enters. The text object has two character filter functions: `NSFieldFilter()` and `NSEditorFilter()`. `NSFieldFilter()` interprets Tab and Return characters as commands to end the text object's status as the first responder. `NSEditorFilter()`, the default filter function, accepts Tab and Return characters into the text. See also `charFilter`.

setCharWrap:

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

Sets how to treat words whose length exceeds the line length. If `YES`, long words are wrapped on a character basis. If `NO`, long words are truncated at the frame boundary. See also `charWrap`.

setClickTable:

```
- (void)setClickTable:(const NSFSM *)aTable
```

Sets the finite-state machine table that defines word boundaries for double-click selection. See also `clickTable`.

setDescentLine:

```
- (void)setDescentLine:(float)value
```

Sets the distance from the base line to the bottom of line to `value`. This method neither rewraps nor redraws the text. Send a `calcLine` message if you want the text rewrapped and redrawn after you reset the descent line. See also `descentLine`.

setDrawFunc:

```
- (void)setDrawFunc:(NSTextFunc)aFunction
```

Makes `aFunction` the function that draws the text. `NSDrawALine()` is the default draw function. See also `drawFunc`.

setFont:paragraphStyle:

```
- (void)setFont:(NSFont *)fontObj
    paragraphStyle:(void *)paragraphStyle
```

Sets the font object and paragraph style for all text. The text is then rewrapped and redrawn. The paragraph style controls such features as tab stops and line indentation. See also `setSelFont:`.

setImportsGraphics:

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

Sets whether the text object can import TIFF and EPS images dragged into it by the user. A `setImportsGraphics:YES` message causes a `setRichText:YES` message to be sent also. This implemenation overrides the method inherited from `NSText`. See `setRichText:`.

setLineHeight:

```
- (void)setLineHeight:(float)value
```

Sets the minimum distance between adjacent lines. For a plain text object, this will be the same for all lines. For rich text objects, line heights will be increased for lines with larger fonts. Even if very small fonts are used, in no case will adjacent lines be closer than this minimum. This method doesn't rewrap or redraw the text. Send a `calcLine` message if you want the text rewrapped and redrawn after you reset the line height. If no line height is set, the default line height will be taken from the default font. See also `lineHeight`.

setLocation:ofCell:

```
- (void)setLocation:(NSPoint)origin ofCell:(NSCell *)cell
```

Sets the x and y coordinates for the `NSCell` object specified by `cell`. The coordinates are specified to by `origin` and are interpreted as being in the text object's coordinate system. This method is provided for programmers who want to write their own scan functions and need a way to position `NSCell` objects found in the text. Sending this message to a text object that uses the standard scan function will have no effect on the placement of `cell`. See also `locationOfCell:`, `replaceSelWithCell:`.

setMarginLeft:right:top:bottom:

```
- (void)setMarginLeft:(float)leftMargin right:(float)rightMargin
    top:(float)topMargin bottom:(float)bottomMargin
```

Adjusts the margins around the text. See also `getMarginLeft:right:top:bottom:`.

setNoWrap

```
- (void)setNoWrap
```

Disables word wrap. It also sets the text alignment to `NSLeftTextAlignment`. See also `charWrap`, `setCharWrap:`.

setParagraphStyle:

```
- (void)setParagraphStyle:(void *)paraStyle
```

Sets the default paragraph style for the entire text. The text is then rewrapped and redrawn. The paragraph style controls features such as tab stops and line indentation. See also `setFont:paragraphStyle:`, `setSelFont:`.

setPostSelSmartTable:

```
- (void)setPostSelSmartTable:(const unsigned char *)aTable
```

Sets the table that specifies which characters on the right end of a selection are treated as equivalent to a space character. See also `postSelSmartTable`, `setPreSelSmartTable:`.

setPreSelSmartTable:

```
- (void)setPreSelSmartTable:(const unsigned char *)aTable
```

Sets the table that specifies which characters on the left end of a selection are treated as equivalent to a space character. See also `preSelSmartTable`, `setPostSelSmartTable:`.

setRetainedWhileDrawing:

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

Sets whether the text object automatically changes its window's buffering type from buffered to retained whenever it redraws itself. Drawing directly to the screen improves the text object's perceived performance, especially if the text contains numerous fonts and formats. Rather than waiting until the entire text is flushed to the screen, the user sees the text being drawn line-by-line.

The window's buffering type changes to retained only while the text object is redrawing itself. In other cases, such as when a user is entering text, the window's buffering type is unaffected. This method is designed to work with text objects that are in buffered windows; don't send this message to a text object in a retained or nonretained window. See also `isRetainedWhileDrawing`.

`setRichText:`

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

Sets whether the text in the text object allows for multiple values of attributes, such as color and font (that is, RTF and RTFD). Sending a `setRichText:NO` message causes a `setImportsGraphics:NO` message to be sent also. This implemenation overrides the method inherited from `NSText`. See also `isRichText` (`NSText`), `setImportsGraphics:`.

`setScanFunc:`

```
- (void)setScanFunc:(NSTextFunc)aFunction
```

Sets the function that calculates the contents of each line of text given the line width, font size, type of text alignment, and other factors. `NSScanALine()` is the default scan function. See also `scanFunc`, `setDrawFunc:`.

`setSelColor:`

```
- (void)setSelColor:(NSColor *)color
```

Sets the text color of the selected text, assuming the text object allows more than one paragraph style and font. Otherwise, this method sets the text color for the entire text. After the text color is set, the text is redisplayed. See also `selColor`, `runColor:`, `setSelFont:`, `NSColor`.

setSelFont:

```
- (void)setSelFont:(NSFont *)fontObj
```

Sets the font object for the selection. The text is then rewrapped and redrawn. See also `setSelFont:paragraphStyle:`, `setSelFontFamily:`, `setSelFontSize:`.

setSelFont:paragraphStyle:

```
- (void)setSelFont:(NSFont *)fontObj
    paragraphStyle:(void *)paragraphStyle
```

Sets the `NSFont` object and paragraph style for the selection. If `fontObj` is `NULL`, no change is made to the selection's font. See also `setSelFont:`.

setSelFontFamily:

```
- (void)setSelFontFamily:(NSString *)fontName
```

Sets the current selection's font family to `fontName`. The text is then rewrapped and redrawn. See also `setSelFontSize:`, `setSelFont:`.

setSelFontSize:

```
- (void)setSelFontSize:(float)size
```

Sets the current selection's font size to `size`. The text is then rewrapped and redrawn. See also `setSelFontFamily:`, `setSelFont:`.

setSelFontStyle:

```
- (void)setSelFontStyle:(NSFontTraitMask)traits
```

Sets the current selection's font style. The text is then rewrapped and redrawn. See also `setSelFontFamily:`, `setSelFontSize:`, `setSelFont:`.

setSelProp:to:

```
- (BOOL)setSelProp:(NSParagraphProperty)property to:(float)value
```

Sets the paragraph style for one or more paragraphs. For a plain text object, all paragraphs are affected. For a rich text object, only those paragraphs marked by the selection are affected. `property` determines which property is modified, and `value` provides additional information needed for some properties. These constants are defined for `property`:

Constant	Property Affected
NSLeftAlignedParagraph	Text alignment. Aligns the text to the left margin. `value` is ignored.
NSRightAlignedParagraph	Text alignment. Aligns the text to the right margin. `value` is ignored.
NSCenterAlignedParagraph	Text alignment. Centers the text between the left and right margins. `value` is ignored.
NSJustificationAlignedParagraph	Not yet implemented.
NSFirstIndentParagraph	Indentation of the first line. `value` specifies the number of units (in the receiver's coordinate system) along the x axis to indent.
NSIndentParagraph	Indentation of lines other than the first line. `value` specifies the number of units (in the receiver's coordinate system) along the x axis to indent.
NSAddTabParagraph	Tab placement. `value` specifies the position on the x axis (in the receiver's coordinate system) to add the new tab.
NSRemoveTabParagraph	Tab placement. `value` identifies the tab to be removed by specifying its position on the x axis (in the receiver's coordinate system).
NSLeftMarginParagraph	Left margin width. `value` gives the new width as a number of units in the receiver's coordinate system.
NSRightMarginParagraph	Right margin width. `value` gives the new width as a number of units in the receiver's coordinate system.

`setSelProp:to:` sets the left and right margins by performing the `setMarginLeft:right:top:bottom:` method. For all other properties, it performs the `setFont:paragraphStyle:` method. After the paragraph property is set, the text is rewrapped and redrawn. See also `setParagraphStyle:`, `setMarginLeft:right:top:bottom:`.

setSelectionStart:end:

```
- (void)setSelectionStart:(int)start end:(int)end
```

Makes the text object the first responder and then selects and highlights a portion of the text from `start` to `end`. To create an empty selection, `start` must equal `end`. Use this method to select a portion of the text programmatically. See also `selectError`, `selectNull`.

setTag:

```
- (void)setTag:(int)anInt
```

Makes `anInt` the text object's tag. See also `tag`.

setTextFilter:

```
- (void)setTextFilter:(NSTextFilterFunc)aFunction
```

Sets the function that analyzes text the user enters. See the `NSTextFilterFunc` type definition in the Application Kit's "Types and Constants chapter" for a description of the text filter function. This filter is different from the character filter in that you're given where the text is to be inserted and the new text that will be inserted. This enables you to write a filter to do auto-indent, or a filter to allow only properly formatted floating point numbers. The character filter doesn't give enough context to determine exactly what the state of the text object is before and after the edit. See also `textFilter`.

showCaret

```
- (void)showCaret
```

Displays the previously hidden caret in the text display. The text object sends itself `showCaret` messages whenever it needs to redisplay the caret; you rarely need to send a `showCaret` message directly. If the text object is not in a window, or the window is not the key window, or the ext object is not editable, this method has no effect. See also `hideCaret`.

startReadingRichText

```
- (void)startReadingRichText
```

This message is sent to the text object just before it begins reading RTF data. The text object responds by sending its delegate a `textWillStartReadingRichText:` message, assuming there is a delegate and it responds to this message. The delegate can then perform any required initialization. Alternatively, a subclass could put these initialization routines in its own implementation of this method. See also `finishReadingRichText`.

tag

```
- (int)tag
```

Returns the text object's tag. See also `setTag:`.

textFilter

```
- (NSTextFilterFunc)textFilter
```

Returns the current text filter function. See also `setTextFilter:`.

trackMouse:inRect:ofView:untilMouseUp:

```
- (BOOL)trackMouse:(NSEvent *)theEvent inRect:(NSRect)cellFrame
    ofView:(NSView *)controlView untilMouseUp:(BOOL)untilMouseUp
```

The embedded object responds to this message from the text object by tracking the mouse while it's within the specified rectangle of the supplied view. `theEvent` is a pointer to the mouse-down event that caused the text object to send this message. `cellFrame` is the area within `controlView` (generally the text object) where the mouse will be tracked. See the `NSCell` class specification for one implementation of this method.

validRequestorForSendType:returnType:

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

Responds to a message that the NSApplication object sends to determine which items in the Services menu should be enabled or disabled at any particular time. Don't send this message directly, but you can override it in a subclass.

A text object registers for services during initialization (however, see excludeFromServicesMenu:). Thereafter, whenever the text object is the first responder, the application object can send it one or more validRequestorForSendType:ReturnType: messages during event processing to determine which Services menu items should be enabled. If the ext object can place data of type sendType on the pasteboard and receive data of type returnType back, it should return self; otherwise it should return nil. The application object checks the return value to determine whether to enable or disable commands in the Services menu.

Since an object can receive one or more of these messages per event, it's important that if you override this method in a subclass of text, the new implementation include no time-consuming calculations. See also validRequestorForSendType:returnType: (NSResponder).

writeSelectionToPasteboard:types:

```
- (BOOL)writeSelectionToPasteboard:(NSPasteboard *)pboard
    types:(NSArray *)types
```

Writes the current selection to the supplied pasteboard object, pboard. types lists the data types to be copied to the pasteboard. A return value of NO indicates that the data of the requested types could not be provided. When the user chooses a command in the Services menu, this message is sent to the first responder. This message is followed by a readSelectionFromPasteboard: message if the command requires the requesting application to replace its selection with data from the service provider. See also readSelectionFromPasteboard:.

Methods Implemented by the Delegate

textDidRead:paperSize:

```
- (void)textDidRead:(NSCStringText *)textObject
    paperSize:(NSSize)paperSize
```

Lets the delegate review paper size. This message is sent to the delegate after the text object reads RTF data, allowing the delegate to modify the paper size. `paperSize` is the dimensions of the paper size specified by the `\paperw` and `\paperh` RTF control words. See also `textWillWrite:`.

textDidResize:oldBounds:

```
- (NSRect)textDidResize:(NSCStringText *)textObject
    oldBounds:(NSRect)oldBounds
```

Responds to a message informing the delegate that the text object has changed its size. `oldBounds` is the text object's bounds rectangle before the change.

textWillConvert:fromFont:toFont:

```
- (NSFont *)textWillConvert:(NSCStringText *)textObject
    fromFont:(NSFont *)font toFont:(NSFont *)font
```

This message lets the delegate intercede in a selection's font change. The message is sent whenever the Font panel sends a `changeFont:` (`NSText` class) message to the text object. `fromFont` is the old font that's currently being changed; `toFont` is the font that's to replace `fromFont`.

textWillFinishReadingRichText:

```
- (void)textWillFinishReadingRichText:(NSCStringText *)textObject
```

Informs the delegate that the text object finished reading RTF data, either from the pasteboard or from a text file.

textWillResize:

```
- (void)textWillResize:(NSCStringText *)textObject
```

Informs delegate of impending size change. This method can specify the maximum dimensions of the text object by using the `resizeTextWithOldBounds:maxRect:` method. If the delegate doesn't implement this method, the change is allowed by default.

`textWillSetSel:toFont:`

```
- (void)textWillSetSel:(NSCStringText *)textObject
    toFont:(NSFont *)font
```

Lets delegate intercede in the updating of `font` in the Font panel.

`textWillStartReadingRichText:`

```
- (void)textWillStartReadingRichText:(NSCStringText *)textObject
```

Informs delegate that text object will read RTF data, either from the pasteboard or from a text file.

`textWillWrite:`

```
- (NSSize)textWillWrite:(NSCStringText *)textObject
```

Lets the delegate specify paper size.

NSCursor

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

Class Description

An `NSCursor` holds an image that the window system can display for the cursor. An `NSCursor` is initialized with an `NSImage` object (which can subsequently be replaced by sending the `NSCursor` a `setImage:` message). This `NSImage` object must contain an `NSBitmapImageRep` representation of the image otherwise an error will occur. To make the window system display a particular image as the current cursor, simply send a `set` message to the `NSCursor` instance associated with that image.

For automatic cursor management, an NSCursor can be assigned to a cursor rectangle within a window. When the window is key and the user moves the cursor into the rectangle, the NSCursor becomes the current cursor. It ceases to be the current cursor when the cursor leaves the rectangle. The assignment is made using NSView's addCursorRect:cursor: method, usually inside a resetCursorRects method:

```
- (void)resetCursorRects
{
    [self addCursorRect:someRect cursor:theNSCursorObject];
}
```

This is the recommended way of associating a cursor with a particular region inside a window. However, the NSCursor class provides two other ways of setting the cursor:

- NSCursor maintains its own stack of cursors. Pushing an NSCursor instance on the stack sets it to be the current cursor. Popping an NSCursor from the stack sets the next NSCursor in line, the one that's then at the top of the stack, to be the current cursor.
- An NSCursor can be made the owner of a tracking rectangle and told to set itself when it receives a mouse-entered or mouse-exited event.

The Application Kit provides two ready-made NSCursor instances: the standard arrow cursor, and the I-beam cursor that's displayed over editable or selectable text. These can be retrieved with the class methods arrowCursor and IBeamCursor, respectively. There's no NSCursor instance for the wait cursor. The wait cursor is displayed automatically by the system, without any required program intervention.

Method Types

Activity	Class Method
Initializing a new NSCursor object	– initWithImage:foregroundColor:backgroundColor: – initWithImage:foregroundColor:backgroundColor:hotSpot – initWithImage:hotSpot:
Defining the cursor	– getForeground:andBackground: – setForeground:andBackground: – hotSpot – image – setImage: – setImage:foregroundColor:backgroundColor:
Setting the cursor	+ hide + pop + setHiddenUntilMouseMoves: + unhide – isSetOnMouseEntered – isSetOnMouseExited – mouseEntered: – mouseExited: – pop – push – set – setOnMouseEntered: – setOnMouseExited:
Getting the cursor	+ arrowCursor + currentCursor + IBeamCursor

Class Methods

arrowCursor

```
+ (NSCursor *)arrowCursor
```

Returns a ready-made arrow cursor. See also `IBeamCursor`.

currentCursor

```
+ (NSCursor *)currentCursor
```

Returns the current cursor. The current cursor is the cursor currently being used by the application.

hide

```
+ (void)hide
```

Hides the cursor. See also `unhide`.

IBeamCursor

```
+ (NSCursor *)IBeamCursor
```

Returns a ready-made I-beam cursor. See also `arrowCursor`.

pop

```
+ (void)pop
```

Removes the cursor at the top of the cursor stack, and sets the cursor that was beneath it to the current cursor. See also `push`, `pop` (instance method).

setHiddenUntilMouseMoves:

```
+ (void)setHiddenUntilMouseMoves:(BOOL)flag
```

Hides the cursor when `flag` is YES; reveals it otherwise.

unhide

```
+ (void)unhide
```

Shows the cursor. See also `hide`.

Instance Methods

getForeground:andBackground:

```
- (void)getForeground:(NSColor *)fg andBackground:(NSColor *)bg
```

Returns the cursor foreground color in `fg` and the cursor background color in `bg`. See also `setForeground:andBackground:`.

hotSpot

```
- (NSPoint)hotSpot
```

Returns the point on the cursor image that is reported as the cursor location.

image

```
- (NSImage *)image
```

Returns the NSImage object that contains the cursor image. See also setImage:.

initWithImage:foregroundColor:backgroundColor:

```
- (id)initWithImage:(NSImage *)newImage
    foregroundColor:(NSColor *) fg
    backgroundColor: (NSColor *) bg
```

Initializes a new NSCursor object with newImage. newImage must be an NSBitmapImageRep. Its contents will be interpreted as follows.

- Only the bottom-left-most 16 by 16 square of the image will be used for the cursor.
- If the image contains alpha samples, then the shape of the cursor composed from the image will correspond to the shape of the portion of the image with nonzero alpha samples.
- If the image does not contain alpha samples, then the shape of the cursor will be a 16 by 16 square.
- If the image has a bitmap representation with 1 bit-per-sample, then the on bits will be colored with the foreground color in the resulting cursor, while the off bits will be colored with the background color.
- If the image has a gray scale bitmap representation (2 or fewer samples per pixel), then this representation will first be translated into a 1-bit-per-sample representation by thresholding the gray scale values at some level (note: this level can be either built in, or computed heuristically based on the values in the image). The resulting representation will then be treated as the 1 bit-per-sample representation described above.

- If the image has an RGB bitmap representation, then the respresentation will first be converted to gray scale, and then treated as the gray scale representation above.

Note that newImage should be an `NSBitmapImageRep` object, otherwise an error will occur. See also `initWithImage:hotSpot:`, `initWithImage:foregroundColor:backgroundColor:hotSpot`.

initWithImage:foregroundColor:backgroundColor:hotSpot

```
- (id)initWithImage:(NSImage *)newImage
    foregroundColor:(NSColor *)fg backgroundColor: (NSColor *) bg
    hotSpot: (NSPoint) hotSpot
```

Initializes a new `NSCursor` with the given foreground and background colors, and sets the hot spot of the new cursor to `hotSpot`. Note that newImage should be an `NSBitmapImageRep` object, otherwise an error will occur. See `initWithImage:foregroundColor:backgroundColor:` for more information.

initWithImage:hotSpot:

```
- (id)initWithImage:(NSImage *)newImage hotSpot:(NSPoint)hotSpot
```

Initializes a new `NSCursor` using the default foreground and background colors, and sets the hot spot of the new cursor to `hotSpot`. Note that newImage should be an `NSBitmapImageRep` object, otherwise an error will occur. See `initWithImage:foregroundColor:backgroundColor:` for more information.

isSetOnMouseEntered

```
- (BOOL)isSetOnMouseEntered
```

Returns YES if a `mouseEntered:` message will set the cursor. See also `isSetOnMouseExited`, `setOnMouseEntered:`, `mouseEntered:`, `mouseExited:`.

isSetOnMouseExited

```
- (BOOL)isSetOnMouseExited
```

Returns YES if a mouseExited: message will set the cursor. See also isSetOnMouseEntered, setOnMouseExited:, mouseEntered:, mouseExited:.

mouseEntered:

```
- (void)mouseEntered:(NSEvent *)theEvent
```

Responds to a mouse-entered event by setting the receiver to be the current cursor, but only if enabled to do so by a previous setOnMouseEntered: message. This method does not push the receiver on the cursor stack. See also setOnMouseEntered:, mouseExited:.

mouseExited:

```
- (void)mouseExited:(NSEvent *)theEvent
```

Responds to a mouse-exited event by setting the receiver to be the current cursor, but only if enabled to do so by a previous setOnMouseExited: message. This method does not push the receiver on the cursor stack. See also setOnMouseExited:, mouseEntered:.

pop

```
- (void)pop
```

Removes the topmost cursor object from the cursor stack, and makes the next cursor object the current cursor. This method is a cover for the class method of the same name. See also push, pop (class method).

push

```
- (void)push
```

Puts the receiving cursor on the cursor stack and sets it to be the current cursor. This method can be used in conjunction with the pop method to manage a group of cursors within a local context. Every push should be matched by a subsequent pop. See also pop.

set

```
- (void)set
```

Sets the NSCursor object to be the current cursor.

setForeground:andBackground:

```
- (void)setForeground:(NSColor *)fg andBackground:(NSColor *)bg
```

Changes the cursor foreground color to fg and the background color to bg. The image associated with the cursor is unchanged. See also getForeground:andBackground:.

setImage:

```
- (void)setImage:(NSImage *)newImage
```

Assigns a new cursor image to the receiving cursor. newImage should be an NSImage object for an image that's 16 pixels wide by 16 pixels high. If the image is smaller than 16-by-16, an error is generated when the application tries to use the cursor, and the previous cursor remains in use. If the image is larger than 16-by-16, only the lower-left 16-by-16 pixels of the image will be displayed. Resetting the image of an cursor while it is the current cursor may have unpredictable results. See also image, initWithImage:foregroundColor:backgroundColor:.

setImage:foregroundColor:backgroundColor:

```
- (void)setImage:(NSImage *)newImage
    foregroundColor: (NSColor *) fg
    backgroundColor: (NSColor *) bg
```

Makes newImage the NSImage object that supplies the cursor image, fg the new foreground cursor color, and bg the new background cursor color. See also initWithImage:foregroundColor:backgroundColor:, setForeground:andBackground:.

setOnMouseEntered:

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

Sets a flag that determines whether on not the mouseEntered: message sets the cursor. If flag is YES, then a mouseEntered: message will set the cursor. Otherwise, a mouseEntered: message does not set the cursor. See also isSetOnMouseEntered, mouseEntered:.

setOnMouseExited:

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

Sets a flag that determines whether on not the `mouseExited:` message sets the cursor. If `flag` is `YES`, then a `mouseExited:` message sets cursor; otherwise a `mouseExited:` message does not set the cursor. See also `isSetOnMouseExited`, `mouseExited:`.

NSCustomImageRep

Inherits From:	NSImageRep : NSObject
Conforms To:	NSCoding, NSCopying (NSImageRep) NSObject (NSObject)
Declared In:	AppKit/NSCustomImageRep.h

An `NSCustomImageRep` is an object that uses a delegated method to render an image. When called upon to produce the image, it sends a message to its delegate to have the method performed.

Like most other kinds of `NSImageRep`s, an `NSCustomImageRep` is generally used indirectly, through an `NSImage` object. An `NSImage` must be able to choose between various representations of a given image. It also needs to provide an off-screen cache of the appropriate depth for any image it uses. It determines this information by querying its `NSImageRep`s.

To work with an `NSImage`, an `NSCustomImageRep` must be able to provide some information about its image. Use the following methods, inherited from the `NSImageRep` class, to set these attributes of the `NSCustomImageRep`:

- `setSize:`
- `setColorSpaceName:`
- `setAlpha:`
- `setPixelsHigh:`
- `setPixelsWide:`
- `setBitsPerSample:`

Method Types

Activity	Class Method
Initializing a new NSCustomImageRep	– initWithDrawSelector:delegate:
Identifying the object	– delegate – drawSelector

Instance Methods

delegate

`– (id)delegate`

Returns the delegate. See also `drawSelector`, `initWithDrawSelector:delegate:`.

drawSelector

`– (SEL)drawSelector`

Returns the associated draw method selector. See also `delegate`, `initWithDrawSelector:delegate:`.

initWithDrawSelector:delegate:

`– (id)initWithDrawSelector:(SEL)aSelector delegate:(id)anObject`

Initializes a new instance so that it delegates the responsibility for drawing to `anObject`. When the `NSCustomImageRep` receives a `draw` message, it sends an `aSelector` message to `anObject`. See also `delegate`, `drawSelector`.

NSDataLink

Inherits From:	NSObject
Conforms To:	NSCoding, NSCopying NSObject (NSObject)
Declared In:	AppKit/NSDataLink.h

Class Description

An `NSDataLink` object (or *data link*) defines a single link between a selection in a source document and a dependent, dynamically updated selection in a destination document. A data link is typically created when linkable data is copied to the pasteboard. First, an `NSSelection` object describing the data is created. Then a link to that selection is created. The link can then be written to the pasteboard.

Once the data and link have been written to the pasteboard, they can be added to a destination document by an object that can respond to a message to Paste and Link. The object responding to this message will paste the data as usual. The destination application will then read the link from the pasteboard, create an `NSSelection` describing the linked data within the destination document, and add the link to the destination document's link manager (`NSDataLinkManager`).

When the link is added to the destination document's link manager, it becomes a *destination link*. At that time, the data link's object establishes a connection with the source document's link manager, which automatically creates a *source link* in the source application; the source link refers to the source selection.

A link that isn't managed by a link manager is a *broken link*. (Both source and destination links have link managers.) All links are broken links when they are created. This ensures that they cause no updates. The disposition of a link (destination, source, or broken) can be retrieved with the `disposition` method. Most of the messages defined by the `NSDataLink` class can be sent to a link of any disposition, but some only make sense when sent to a link with a specific disposition; these are so noted in their method descriptions.

Links of all dispositions except links to files maintain an `NSSelection` object referring to the link's selection in the source document; this selection is returned by the `sourceSelection` method. Source and destination links also

maintain an NSSelection describing the location of the data in the destination document; this selection is returned by the destinationSelection method.

Note – NSDataLink is not part of the OpenStep specification.

See the NSSelection class description for more information on NSSelection objects. See also NSDataLinkManager.

Method Types

Activity	Class Method
Information about the link	– disposition – linkNumber – manager
Information about the link's source	– currentSourceFilename – lastUpdateTime – sourceApplicationName – sourceFilename – sourceSelection – types
Information about the link's destination	– destinationApplicationName – destinationFilename – destinationSelection

Instance Methods

currentSourceFilename

```
- (NSString *)currentSourceFilename
```

Returns the links "best guess" at the current location of the source file. Returns nil if the location cannot be ascertained.

destinationApplicationName

```
– (NSString *)destinationApplicationName
```

Returns the name of the application that owns the destination document. See also `sourceApplicationName`, `destinationFilename`.

destinationFilename

```
- (NSString *)destinationFilename
```

Returns the destination document file name. See also `sourceFilename`, `destinationApplicationName`, `destinationSelection`.

destinationSelection

```
- (NSSelection *)destinationSelection
```

Returns the destination selection, which describes how the linked data is represented in the destination document. See also `sourceSelection`, `destinationApplicationName`, `destinationFilename`.

disposition

```
- (NSDataLinkDisposition)disposition
```

Identifies the link as a source link, a destination link, or a broken link by returning one of the following values:

- `NSLinkInDestination`
- `NSLinkInSource`
- `NSLinkBroken`

See also `linkNumber`, `manager`.

lastUpdateTime

```
- (NSDate *)lastUpdateTime
```

Returns the last time the link was updated. A link could be updated for many reasons; for example, a message could be sent to the source document's link manager telling it that its document was saved, or the link could be brought up to date with an `updateDestination` message.

linkNumber

```
- (NSDataLinkNumber)linkNumber
```

Returns a destination link's link number, which may be useful in identifying the link. This number is constant through the life of the document, and unique among the document's links; it is not meaningful in source links. See also `manager`.

manager

```
- (NSDataLinkManager *)manager
```

Returns the link's manager, or `nil` if it doesn't have a manager (for example, returns `nil` if a link is broken).

sourceApplicationName

```
- (NSString *)sourceApplicationName
```

Returns the name of the application that owns the source document. See also `sourceFilename`, `sourceSelection`, `types`, `lastUpdateTime`.

sourceFilename

```
- (NSString *)sourceFilename
```

Returns the source document's file name. See also `sourceApplicationName`, `sourceSelection`, `types`.

sourceSelection

```
- (NSSelection *)sourceSelection
```

Returns the source selection or `nil` if the link refers to an entire file, in which case the source file can be retrieved using `sourceFilename`. See also `sourceApplicationName`.

types

```
- (NSArray *)types
```

Returns the pasteboard types that the source document can provide. See also `NSPasteboard`.

NSDataLinkManager

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

Class Description

An `NSDataLinkManager` object (also known as a *data link manager* or simply *link manager*) manages data linked from and into a document through `NSDataLink` objects. `NSDataLink` objects (or *data links*) provide a link between a selection in a source document and a dependent, dynamically updated selection in a destination document. When a user does a Paste and Link command in the destination document, the link manager creates the link in response . When this link is added to the destination document, it makes a connection with the source document's link manager, which creates a source link in the source application.

Note – `NSDataLinkManager` is not part of the OpenStep specification.

For more information about `NSDataLink` objects, see the `NSDataLink` class description. See the `NSSelection` class description for more information on `NSSelection` objects.

Method Types

Activity	Class Method
Initializing and freeing a link manager	– initWithDelegate:
Getting and setting information about the manager's links	– destinationLinkEnumerator – sourceLinkEnumerator

Instance Methods

destinationLinkEnumerator

```
– (NSEnumerator *)destinationLinkEnumerator
```

Returns an enumerator of the destination's source links. See also `sourceLinkEnumerator`, `NSEnumerator` (Foundation Kit).

initWithDelegate:

```
– (id)initWithDelegate:(id)anObject
```

Initializes and returns a newly allocated `NSDataLinkManager` instance for a new document. The link manager's delegate, specified by `anObject`, will be expected to provide source data, paste destination data, and help the data link manager keep links up to date. Before data in the document can be linkable, the document will have to be saved and the link manager will have to be informed of the document's name by a `noteDocumentSavedAs:` message.

sourceLinkEnumerator

```
– (NSEnumerator *)sourceLinkEnumerator
```

Returns an `NSEnumerator` of the receiver's source links. See also `destinationLinkEnumerator`, `NSEnumerator` (Foundation Kit).

NSDataLinkPanel

Inherits From:	NSPanel : NSWindow : NSResponder : NSObject
Conforms To:	NSCoding (NSResponder) NSObject (NSObject)
Declared In:	AppKit / NSDataLinkPanel.h

Class Description

An `NSDataLinkPanel` is an `NSPanel` that allows the user to inspect data links. The `NSDataLinkPanel` sends messages to the current data link manager (representing the current document) and to the current link (representing the current selection if it's based on a data link). Thus, the panel should be informed, by a `setLink:manager:isMultiple:` message, any time the selection changes or a document is created or activated. Since the selection may need to be tracked even before the panel is created, this message can be sent to either the `NSDataLinkPanel` class or the shared instance.

The `NSDataLinkPanel` is generally displayed using `NSApplication`'s `orderFrontDataLinkPanel:` method. An application's sole instance of `NSDataLinkPanel` can be accessed with the `sharedDataLinkPanel` method.

Note – `NSDataLinkPanel` is not part of the OpenStep specification.

Method Types

Activity	Class Method
Initializing	+ sharedDataLinkPanel
Keeping the panel up-to-date	+ getLink:manager:isMultiple: + setLink:manager:isMultiple: – getLink:manager:isMultiple: – setLink:manager:isMultiple:
Customizing the panel:	– accessoryView – setAccessoryView:
Responding to user input	– pickedBreakAllLinks: – pickedBreakLink: – pickedOpenSource: – pickedUpdateDestination: – pickedUpdateMode:

Class Methods

getLink:manager:isMultiple:

```
+ (void)getLink:(NSDataLink **)link
    manager:(NSDataLinkManager **)linkManager
    isMultiple:(BOOL *)flag
```

Gets information about the `NSDataLinkPanel`'s currently selected link; returns the link in `link`, the link manager in `linkManager`, and the multiple selection status in `flag`. Whenever a link is selected or deselected, this information must be set using `setLink:manager:isMultiple:`.

setLink:manager:isMultiple:

```
+ (void)setLink:(NSDataLink *)link
    manager:(NSDataLinkManager *)linkManager isMultiple:(BOOL)flag
```

Informs the `NSDataLinkPanel` of the current document and selection. This message must be sent any time data, based on a data link, is selected or deselected, or when a document (and therefore a new link manager) is activated. Since the state of the selection always needs to be tracked, this message can be sent to either the `NSDataLinkPanel` class or instance. `link` is the currently selected link; it should be `nil` if no link is selected.

linkManager is the current link manager. flag should be YES if the panel is to indicate that more than one link is selected. See also setLink:manager:isMultiple: (instance method).

sharedDataLinkPanel

```
+ (NSDataLinkPanel *)sharedDataLinkPanel
```

Initializes and returns the shared NSDataLinkPanel object.

Instance Methods

accessoryView

```
- (NSView *)accessoryView
```

Returns the NSDataLinkPanel's custom accessory view. See also setAccessoryView:.

getLink:manager:isMultiple:

```
- (void)getLink:(NSDataLink **)link
    manager:(NSDataLinkManager **)linkManager
    isMultiple:(BOOL *)flag
```

Returns information about the NSDataLinkPanel's currently selected link. This method returns the link in link, the link manager in linkManager, and the multiple selection status in flag. This method functions identically to the class method of the same name. Whenever a link is selected or deselected, this information must be set using setLink:andManager:isMultiple:.

pickedBreakAllLinks:

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

Invoked when the user clicks the Break All Links button, this method puts up an attention panel to confirm the user's action, and then sends a breakAllLinks (NSDataLinkManager) message to the current link manager, as set by setLink:manager:isMultiple:. See also pickedBreakLink:, pickedOpenSource:, pickedUpdateDestination:, pickedUpdateMode:.

pickedBreakLink:

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

Invoked when the user clicks the Break Link button; puts up an attention panel to confirm the user's action, and then sends a `break` message (`NSDataLink`) to the current link, as set by `setLink:manager:isMultiple:`. See also `pickedBreakAllLinks:`.

pickedOpenSource:

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

Invoked when the user clicks the Open Source button, this method sends an `openSource` message (`NSDataLink`) to the current link, as set by `setLink:manager:isMultiple:`. See also `pickedBreakAllLinks:`.

pickedUpdateDestination:

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

Invoked when the user clicks the Update from Source button; sends a message to the current link to verify and update the data source and then update the destination data. See also `pickedBreakAllLinks:`.

pickedUpdateMode:

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

Invoked when the user selects the update mode; sends a `setUpdateMode:` message to the current link, as set by `setLink:andManager:isMultiple:`.

setAccessoryView:

```
- (void)setAccessoryView:(NSView *)aView
```

Adds `aView` to the `NSDataLinkPanel`'s view hierarchy. Applications can invoke this method to add an `NSView` that contains their own controls. The panel is automatically resized to accommodate `aView`. This method can be invoked repeatedly to change the accessory view depending on the situation. If `aView` is `nil`, then the panel's current accessory view, if any, is removed. See also `accessoryView`.

setLink:manager:isMultiple:

```
- (void)setLink:(NSDataLink *)link
    manager:(NSDataLinkManager *)linkManager isMultiple:(BOOL)flag
```

Informs the NSDataLinkPanel of the current document and selection. link is the currently selected link; it should be nil if no link is selected. linkManager is the current link manager. flag should be YES if the panel is to indicate that more than one link is selected. Returns the NSDataLinkPanel class. This message must be sent any time data based on a data link is selected or deselected, or when a document (and therefore a new link manager) is activated. This method functions identically to the class method of the same name; since the state of the selection always needs to be tracked, this message can be sent to either the NSDataLinkPanel class or instance.

NSEPSImageRep

Inherits From:	NSImageRep : NSObject
Conforms To:	NSCoding, NSCopying (NSImageRep) NSObject (NSObject)
Declared In:	AppKit/NSEPSImageRep.h

Class Description

An NSEPSImageRep is an object that can render an image from encapsulated PostScript code (EPS). Like most other kinds of NSImageReps, an NSEPSImageRep is generally used indirectly, through an NSImage object. An NSImage must be able to choose between various representations of a given image. It also needs to provide an off-screen cache of the appropriate depth for any image it uses. It determines this information by querying its NSImageReps.

To work with an NSImage, an NSEPSImageRep must be able to provide some information about its image. The size of the object is set from the bounding box specified in the EPS header comments. Use these methods, inherited from the NSImageRep class, to set the other attributes of the NSEPSImageRep:

- setColorSpaceName:
- setAlpha:

- setPixelsHigh:
- setPixelsWide:
- setBitsPerSample:

Method Types

Activity	Class Method
Initializing a new instance	+ imageRepWithData: – initWithData:
Getting image data	– boundingBox – EPSRepresentation
Drawing the image	– prepareGState

Class Methods

imageRepWithData:

```
+ (id)imageRepWithData:(NSData *)epsData
```

Invokes initWithData: to return an instance with data from epsData. See also initWithData:.

Instance Methods

boundingBox

```
- (NSRect)boundingBox
```

Returns the rectangle that bounds the image. See also EPSRepresentation.

EPSRepresentation

```
- (NSData *)EPSRepresentation
```

Returns the EPS representation of the image. See also boundingBox.

initWithData:

```
- (id)initWithData:(NSData *)epsData
```

Initialize an instance with data from `epsData`. See also `imageRepWithData:`.

prepareGState

```
- (void)prepareGState
```

Implemented by subclasses to initialize the graphics state before the image is drawn.

NSEvent

Inherits From:	NSObject
Conforms To:	NSCoding, NSCopying NSObject (NSObject)
Declared In:	AppKit/NSEvent.h

Class Description

An `NSEvent` object contains information about an event such as a mouse-click or a key-down. The window system associates each such user action with a window, reporting the event to the application that created the window. Pertinent information about each event—such as which character was typed and where the mouse was located—is collected in an `NSEvent` object and made available to the application. As events are received in the application, they're temporarily placed in storage called the event queue. When the application is ready to process an event, it takes an `NSEvent` from the queue.

`NSEvent`s are typically passed to the responder chain—a set of objects within the window that inherit from `NSResponder`. For example, `NSResponder`'s `mouseDown:` and `keyDown:` methods take an `NSEvent` as an argument. When an `NSApplication` retrieves an `NSEvent` from the event queue, it dispatches it to the appropriate `NSWindow` (which is itself an `NSResponder`) by invoking `keyDown:` or a similar message. The `NSWindow` passes the event to the first responder, and the event gets passed on down the responder chain until some

object handles it. In the case of a mouse-down, a `mouseDown:` message is sent to the `NSView` in which the user clicked the mouse, which relays the message to its next responder if it can't handle the message itself.

Most events follow this same path: from the window system to the application's event queue, and from there, to the appropriate objects of the application. However, the Application Kit can create an `NSEvent` from scratch and insert it into the event queue for distribution, or send it directly to its destination. It's rare for an *application* to create an event directly, but it's possible, using `NSEvent` class methods. The newly created events can be added to the event queue by invoking `NSWindow`'s (or `NSApplication`'s) `postEvent:atStart:` method.

Events are retrieved from the event queue by calling the `NSWindow` method `nextEventMatchingMask:untilDate:inMode:dequeue:` or a similar `NSApplication` method. These methods return an instance of `NSEvent`. The nature of the retrieved event can then be ascertained by invoking `NSEvent` instance methods—`type`, `window`, and so forth. All types of events are associated with a window. The corresponding `NSWindow` object can be gotten by invoking `window`. The location of the event within the window's coordinate system is obtained from `locationInWindow`, and the time of the event is gotten from `timestamp`. The `modifierFlags` method returns an indication of which modifier keys (Command, Control, Shift, and so forth) were held down while the event occurred.

The `type` method returns an `NSEventType`, a constant that identifies the sort of event. The different types of events fall into five groups:

- Keyboard events
- Mouse events
- Tracking-rectangle events
- Periodic events
- Cursor-update events

Some of these groups comprise several `NSEventType` constants; others only one. The following sections discuss the groups, along with the corresponding `NSEventType` constants.

Keyboard Events

Among the most common events sent to an application are direct reports of the user's keyboard actions, identified by these three `NSEventType` constants:

- `NSKeyDown`: The user generated a character by pressing a key.
- `NSKeyUp`: The key was released.
- `NSFlagsChanged`: The user pressed or released a modifier key, or turned Alpha Lock on or off.

Of these, key-down events are the most useful to the application. When the `type` method returns `NSKeyDown`, your next step is typically to determine the character or characters generated by the key-down, by sending the `NSEvent` a `characters` message.

Key-up events are less used since they follow almost automatically when there has been a key-down event. Because `NSEvent`'s `modifierFlags` method returns the state of the modifier keys regardless of the type of event, applications normally don't need to receive flags-changed events; they're useful only for applications that have to keep track continuously of the state of these keys.

Mouse Events

Mouse events are generated by changes in the state of the mouse buttons and by changes in the position of the mouse cursor on the screen. This category consists of:

- `NSLeftMouseDown`, `NSLeftMouseUp`, `NSRightMouseDown`, `NSRightMouseUp`: Two sets of mouse-down and mouse-up events, one for the left mouse button and one for the right. “Mouse-down” means the user pressed the button; “mouse-up” means the button was released. If the mouse has just one button, only left mouse events are generated. By sending a `clickCount` message to the `NSEvent`, you can determine whether the mouse event was a single-click, double-click, and so on.
- `NSLeftMouseDragged`, `NSRightMouseDragged`: Two types of mouse-dragged events—one for when the mouse is moved with its left mouse button down, or with both buttons down, and one for when it's moved with just the right button down. A mouse with a single button generates only left mouse-dragged events. As the mouse is moved with a button down, a series of mouse-dragged events is produced. The series is always preceded by a mouse-down event and followed by a mouse-up event.
- `NSMouseMoved`: The user moved the mouse without holding down either mouse button.

Mouse-dragged and mouse-moved events are generated repeatedly as long as the user keeps moving the mouse. If the user holds the mouse stationary, neither event is generated until it moves again.

Note – OpenStep doesn't specify facilities for the third button of a three-button mouse.

Tracking-Rectangle Events

`NSMouseEntered` and `NSMouseExited` events are like the "Mouse Events" listed previously, in that they're dependent on mouse movements. However, unlike the others, they're generated only if the application has asked the window system to set a tracking rectangle in a window. An `NSMouseEntered` or `NSMouseExited` event is created when the cursor has entered the tracking rectangle or left it. A window can have any number of tracking rectangles; the `NSEvent` method `trackingNumber` identifies which rectangle was entered or exited.

Periodic Events

An event of type `NSPeriodic` simply notifies an application that a certain time interval has elapsed. By using the `NSEvent` class method `startPeriodicEventsAfterDelay:withPeriod:`, an application can register that it wants periodic events and that they should be placed in its event queue at a certain frequency. When the application no longer needs them, the flow of periodic events can be turned off by invoking `stopPeriodicEvents`. An application can't have more than one stream of periodic events active at a time. Unlike keyboard and mouse events, periodic events aren't dispatched to an `NSWindow`.

Cursor-Update Events

Events of type `NSCursorUpdate` are used to implement `NSView`'s cursor-rectangle methods. An `NSCursorUpdate` event is generated when the cursor has crossed the boundary of a predefined rectangular area. The application can respond by updating the cursor's shape.

Method Types

Activity	Class Method
Creating NSEvent objects	+ enterExitEventWithType:location:modifierFlags: timestamp:windowNumber:context:eventNumber: trackingNumber:userData: + keyEventWithType:location:modifierFlags: timestamp:windowNumber:context:characters: charactersIgnoringModifiers:isARepeat:keyCode: + mouseEventWithType:location:modifierFlags: timestamp:windowNumber:context:eventNumber: clickCount:pressure: + otherEventWithType:location:modifierFlags: timestamp:windowNumber:context:subtype: data1:data2:
Getting general event information	– context – locationInWindow – modifierFlags – timestamp – type – window – windowNumber
Getting key event information	– characters – charactersIgnoringModifiers – isARepeat – keyCode
Getting mouse event information	– clickCount – eventNumber – pressure
Getting tracking event information	– trackingNumber – userData
Requesting periodic events	+ startPeriodicEventsAfterDelay:withPeriod: + stopPeriodicEvents
Getting information about specially defined events	– data1 – data2 – subtype

Class Methods

enterExitEventWithType:location:modifierFlags: timestamp:windowNumber:context:eventNumber: trackingNumber:userData:

```
+ (NSEvent *)enterExitEventWithType:(NSEventType)type
    location:(NSPoint)location modifierFlags:(unsigned int)flags
    timestamp:(NSTimeInterval)time windowNumber:(int)windowNum
    context:(NSDPSContext *)context eventNumber:(int)eventNum
    trackingNumber:(int)trackingNum userData:(void *)userData
```

Creates and returns an `NSEvent` object initialized with general event data and information specific to mouse tracking (`eventNum`, `trackingNum`, `userData`). Applications rarely create these objects.

keyEventWithType:location:modifierFlags: timestamp:windowNumber:context:characters: charactersIgnoringModifiers:isARepeat:keyCode:

```
+ (NSEvent *)keyEventWithType:(NSEventType)type
    location:(NSPoint)location modifierFlags:(unsigned int)flags
    timestamp:(NSTimeInterval)time windowNumber:(int)windowNum
    context:(NSDPSContext *)context characters:(NSString *)keys
    charactersIgnoringModifiers:(NSString *)ukeys
    isARepeat:(BOOL)repeatKey keyCode:(unsigned short)code
```

Creates and returns an `NSEvent` object initialized with general event data and information specific to keyboard events (`keys`, `repeatKey`, `code`, `ukeys`). `ukeys` sets the unmodified character string. Applications rarely create these objects.

mouseEventWithType:location:modifierFlags: timestamp:windowNumber:context:eventNumber: clickCount:pressure:

```
+ (NSEvent *)mouseEventWithType:(NSEventType)type
    location:(NSPoint)location modifierFlags:(unsigned int)flags
    timestamp:(NSTimeInterval)time windowNumber:(int)windowNum
    context:(NSDPSContext *)context eventNumber:(int)eventNum
    clickCount:(int)clickNum pressure:(float)pressureValue
```

Creates and returns an NSEvent object initialized with general event data and information specific to mouse events (eventNum, clickNum, pressureValue). Applications rarely create these objects.

otherEventWithType:location:modifierFlags: timestamp:windowNumber:context:subtype: data1:data2:

```
+ (NSEvent *)otherEventWithType:(NSEventType)type
    location:(NSPoint)location modifierFlags:(unsigned int)flags
    timestamp:(NSTimeInterval)time windowNumber:(int)windowNum
    context:(NSDPSContext *)context subtype:(short)subType
    data1:(int)data1 data2:(int)data2
```

Creates and returns an NSEvent object initialized with general event data and information specific to kit-defined events (subType, data1, data2). Applications rarely create these objects.

startPeriodicEventsAfterDelay:withPeriod:

```
+ (void)startPeriodicEventsAfterDelay:(NSTimeInterval)delaySeconds
    withPeriod:(NSTimeInterval)periodSeconds
```

Starts generating periodic events with frequency periodSeconds after delay delaySeconds for the current thread. Used for initial delay and periodic behavior in tracking loops. See also stopPeriodicEvents.

stopPeriodicEvents

```
+ (void)stopPeriodicEvents
```

Stops generating periodic events for the current thread, and discard any periodic events remaining in the queue. See also startPeriodicEventsAfterDelay:withPeriod:.

Instance Methods

characters

```
– (NSString *)characters
```

Returns the character code (a string of characters generated by the key event). See also `charactersIgnoringModifiers`, `isARepeat`, `keyCode`.

charactersIgnoringModifiers

```
- (NSString *)charactersIgnoringModifiers
```

Returns the string of characters generated by the key event as if no modifier key had been pressed (except for Shift). See also `characters`.

clickCount

```
- (int)clickCount
```

Returns the number of mouse clicks associated with the mouse event. See also `pressure`.

context

```
- (NSDPSContext *)context
```

Returns the Display PostScript context of the event. See also `modifierFlags`, `timestamp`, `locationInWindow`.

data1

```
- (int)data1
```

Returns special data associated with the event. Used for Application Kit, system, and application-defined events. See also `data2`, `subtype`.

data2

```
- (int)data2
```

Returns special data associated with the event. Used for Application Kit, system, and application-defined events. See also `data1`, `subtype`.

eventNumber

```
- (int)eventNumber
```

Returns the event number of the latest mouse-down event. This information is also useful for handling tracking events. See also `clickCount`.

isARepeat

`- (BOOL)isARepeat`

Returns whether the key event is being repeated (user is holding down the key). See also `characters`.

keyCode

`- (unsigned short)keyCode`

Returns the code that maps to a key on the keyboard. See also `characters`.

locationInWindow

`- (NSPoint)locationInWindow`

Returns the event's location in the base coordinate system of the event's window. See also `window`, `windowNumber`.

modifierFlags

`- (unsigned int)modifierFlags`

Returns an integer bitfield containing modifier-key flags. See also `context`, `type`.

pressure

`- (float)pressure`

Returns a value indicating the pressure applied to the input device (used for appropriate devices, not a mouse). See also `clickCount`.

subtype

`- (short)subtype`

Returns the identifier of the specially defined event.

timestamp

```
- (NSTimeInterval)timestamp
```

Returns the time the event occurred in seconds since system startup. See also `context`.

trackingNumber

```
- (int)trackingNumber
```

Returns the number that identifies the tracking rectangle. See also `userData`.

type

```
- (NSEventType)type
```

Returns the event type (left-mouse-up, right-mouse-dragged, key-down, etc.). The event types are

- `NSNoEvent`
- `NSLeftMouseDown`
- `NSLeftMouseUp`
- `NSRightMouseDown`
- `NSRightMouseUp`
- `NSMouseMoved`
- `NSLeftMouseDragged`
- `NSRightMouseDragged`
- `NSMouseEntered`
- `NSMouseExited`
- `NSKeyDown`
- `NSKeyUp`
- `NSFlagsChanged`
- `NSAppKitDefined`
- `NSSystemDefined`
- `NSApplicationDefined`
- `NSPeriodic`
- `NSCursorUpdate`

userData

```
- (void *)userData
```

Returns data arbitrarily associated with the event. See also `trackingNumber`.

`window`

```
- (NSWindow *)window
```

Returns the window object associated with the event.See also `context`, `windowNumber`.

`windowNumber`

```
- (int)windowNumber
```

Returns the number of the window associated with the event. See also `window`.

NSFont

Inherits From:	NSObject
Conforms To:	NSCoding, NSCopying NSObject (NSObject)
Declared In:	AppKit/NSFont.h

Class Description

The `NSFont` class declares the programmatic interface to objects that correspond to fonts. `NSFont` is in principle an abstract class that represents fonts in general, not just PostScript fonts. In practice, at this time, `NSFont` objects represent PostScript fonts. Each `NSFont` object records a font's name, size, style, and matrix. When an `NSFont` object receives a `set` message, it establishes its font as the current font in the PostScript Server's current graphics state.

For a given application, only one `NSFont` object is created for a particular PostScript font/size or font/matrix combination. That is—if you ask for 24-point Optima, a new font object is created for 24-point Optima if such an object doesn't exist already. When the `NSFont` class object receives a message to create a new object for a particular font, it first checks whether an object has already been created for that font. If so, the the `NSFont` class object returns the existing font object; otherwise, the the `NSFont` class object creates a new font object and returns it.

This sharing of NSFont objects minimizes the number of distinct font objects created. It also implies that no one object in your application can know whether it has the only reference to a particular NSFont object. Thus, NSFont objects shouldn't be deallocated, but should be treated like autoreleased Foundation class objects.

Where *matrix* is used, it refers to a PostScript-style six-element array of numbers that indicate transformations to be applied to a font. An NSFontIdentityMatrix identifies a font matrix used for fonts created by specifying a size.

The *size* of a font in the method definitions is defined in "points", which in currently accepted practice are actually PostScript units. A PostScript unit is 1/72 of an inch, or 0.0139 of an inch. In metric equivalents, a PostScript unit is 0.3528 millimetres. PostScript "points" are minimally different from "printer's points", so for all intents and purposes you can think of PostScript units and points as interchangeable.

In general, you instantiate an NSFont object by sending one of the methods in the "Creating a Font Object" list below to the NSFont class object. The methods with system and user in their names obtain special predetermined fonts defined at the system level and the application level. In general, you would use the fontWithName:size: and fontWithName:matrix: methods to obtain a named font.

A variety of methods are available for querying a font object. In particular, AFM (Adobe Font Metrics) data can be obtained by invoking afmDictionary or afmFileContents. Methods whose descriptions state "Returns...and matrix NSFontIdentityMatrix" actually return an NSFontIdentityMatrix whose first and fourth elements are multiplied by the current size of the font.

Exceptions

Methods listed in "Creating a Font Object" can all raise a NSFontUnavailableException if the requested font can't be constructed.

Method Types

Activity	Class Method
Creating a font object	+ boldSystemFontOfSize: + fontWithName:matrix: + fontWithName:size: + systemFontOfSize: + userFixedPitchFontOfSize: + userFontOfSize:
Setting the font	+ setUserFixedPitchFont: + setUserFont: + useFont: – set
Querying the font	– afmDictionary – afmFileContents – ascender – boundingRectForFont – capHeight – descender – displayName – encodingScheme – familyName – fontName – glyphWithName: – isBaseFont – isFixedPitch – italicAngle – matrix – pointSize – printerFont – screenFont – underlinePosition – underlineThickness – widthOfString: – widths – xHeight
Manipulating glyphs	– advancementForGlyph: – boundingRectForGlyph: – glyphIsEncoded: – positionOfGlyph:precededByGlyph:isNominal:

Class Methods

boldSystemFontOfSize:

```
+ (NSFont *)boldSystemFontOfSize:(float)fontSize
```

Returns a font object representing the bold system font of size `fontSize` using the identity matrix. The bold system font is used for text in attention panels and window titles. If `fontSize` is 0, then `NSUserDefaults` (Foundation Kit) supplies the default size. This method raises `NSFontUnavailableException` if a suitable font object cannot be found.

fontWithName:matrix:

```
+ (NSFont *)fontWithName:(NSString *)fontName
    matrix:(const float *)fontMatrix
```

Returns a font object for font `fontName` and matrix `fontMatrix`. If this font object already exists, it is returned. Otherwise a new font object is created and returned. If an error occurs, `nil` is returned. See also `fontWithName:size:`.

fontWithName:size:

```
+ (NSFont *)fontWithName:(NSString *)fontName size:(float)fontSize
```

Returns a font object for font `fontName` of size `fontSize`. If this font object already exists, it is returned. Otherwise a new font object is created and returned. If an error occurs, `nil` is returned. See also `fontWithName:matrix:`.

setUserFixedPitchFont:

```
+ (void)setUserFixedPitchFont:(NSFont *)aFont
```

Sets the default fixed-pitch font used in the application to `aFont`.This method is intended for an application that wants to override the default fixed-pitch font. See also `setUserFont:`, `NSUserDefaults` (Foundation Kit).

setUserFont:

```
+ (void)setUserFont:(NSFont *)aFont
```

Sets the default standard font used in the application to `aFont`. This method is intended for an application that wants to override the default standard font. See also `setUserFixedPitchFont:`, `NSUserDefaults` (Foundation Kit).

systemFontOfSize:

```
+ (NSFont *)systemFontOfSize:(float)fontSize
```

Returns the font object representing the system font of size `fontSize` and matrix `NSFontIdentityMatrix`. The system font is used for text in panels, menus, and similar objects. If `fontSize` is 0, then `NSUserDefaults` (Foundation Kit) supplies the default size.

useFont:

```
+ (void)useFont:(NSString *)fontName
```

Registers that `fontName` is used in the document. This information is used by the printing machinery. The font class object keeps track of the fonts that are being used in a document by registering the font whenever a font object receives a `set` message. When a document is being prepared for printing, the font class provides the list of fonts required for the `%%DocumentFonts` comment (see *Document Structuring Conventions* by Adobe Systems Inc.). `useFont:` augments this system by providing a way to register fonts that are included in the document but not set using the font's `set` method. For example, you might set a font by executing the `setfont` operator within a function created by `pswrap`. In such a case, make sure to pair the use of the font with a `useFont:` message to register the font with the Font class object. See also `set`.

userFixedPitchFontOfSize:

```
+ (NSFont *)userFixedPitchFontOfSize:(float)fontSize
```

Returns the font object representing the application's fixed-pitch font of size `fontSize` and matrix `NSFontIdentityMatrix`. If fontSize is 0, then `NSUserDefaults` (Foundation Kit) supplies the default fixed-pitch font size. This method raises `NSFontUnavailableException` if a suitable font object cannot be found. See also `userFontOfSize:`.

userFontOfSize:

```
+ (NSFont *)userFontOfSize:(float)fontSize
```

Returns the font object representing the application's standard font of size `fontSize` and matrix `NSFontIdentityMatrix`. If fontSize is 0, then `NSUserDefaults` (Foundation Kit) supplies the default fixed-pitch font size. This method provides an easy way of determining the user's font preference, which you can then use to initialize new documents. This method raises `NSFontUnavailableException` if a suitable font object cannot be found. See also `userFixedPitchFontOfSize:`.

Instance Methods

advancementForGlyph:

```
- (NSSize)advancementForGlyph:(NSGlyph)aGlyph
```

Returns the horizontal and vertical advancement for `aGlyph`. That is, this method returns the amount by which the current point would be displaced in both horizontal and vertical axes if the specified glyph were rendered in the current font and size. In general, the vertical displacement for "Western" fonts will be zero. See also `NSSize` (Foundation Kit).

afmDictionary

```
- (NSDictionary *)afmDictionary
```

Returns the font's AFM dictionary if the font has an AFM file. The return value can possibly be `nil`, so you must check to determine if a non-`nil` value was actually returned. See also `afmFileContents`.

afmFileContents

```
- (NSString *)afmFileContents
```

Returns the raw contents of the entire AFM file, in terms of strings, if the font has an AFM file. Returns `nil` otherwise. See also `afmDictionary`.

ascender

```
- (float)ascender
```

Returns the font's height above the base line. This value is used to determine interline spacing. See also `descender`.

boundingRectForFont

```
– (NSRect)boundingRectForFont
```

Returns the bounding rectangle for the font, scaled to the current size of the font. See also `boundingRectForGlyph:`, `NSRect` (Foundation Kit).

boundingRectForGlyph:

```
– (NSRect)boundingRectForGlyph:(NSGlyph)aGlyph
```

Returns a bounding rectangle for `aGlyph`, scaled to the font's actual size and matrix. See also advancementForGlyph:, `glyphIsEncoded:`, `positionOfGlyph:precededByGlyph:isNominal:`, `NSRect` (Foundation Kit).

capHeight

```
- (float)capHeight
```

Returns the nominal height of the font's capital letters. This is defined as the height of the Latin uppercase X letter, where applicable.

descender

```
- (float)descender
```

Returns the recommended typographic descent below the font baseline. Used for determining interline spacing. See also `ascender`.

displayName

```
– (NSString *)displayName
```

Returns the full name of the font as displayed in the font panel: for example, the font name “Futura-CondExtraBoldObl” returns the display name “Futura Condensed Extra Bold Oblique”. This is the localized version of the font’s name. It is not necessarily the `FullName` field of the font. See also `familyName`.

encodingScheme

```
- (NSString *)encodingScheme
```

Returns the name of the character set used to encode the font glyphs.

familyName

```
– (NSString *)familyName
```

Returns the font’s family name. For example, the font named “Futura-CondExtraBoldObl” returns the family name “Futura”. See also `fontName`, `displayName`.

fontName

```
– (NSString *)fontName
```

Returns the font name, as would be used in a PostScript language program. See also `displayName`, `familyName`.

glyphIsEncoded:

```
– (BOOL)glyphIsEncoded:(NSGlyph)aGlyph
```

Indicates whether `aGlyph` is encoded. That is, this method returns `YES` if `aGlyph` is present in the encoding for the font.

glyphWithName:

```
- (NSGlyph)glyphWithName:(NSString *)aName
```

Returns the font glyph with name `aName`.

isBaseFont

```
- (BOOL)isBaseFont
```

Returns YES if the font is a base font, as opposed to a composite font.

isFixedPitch

```
- (BOOL)isFixedPitch
```

Returns YES if the receiver is a fixed-pitch font, and returns NO otherwise.

italicAngle

```
- (float)italicAngle
```

Returns the posture angle of the typeface design, in 1/64 degrees, measured from the glyph origin counterclockwise from the three o'clock position.

matrix

```
- (const float *)matrix
```

Returns a pointer to an array of six floats representing the font's matrix. You should not alter the data pointed to by `matrix`. If you wish to change values for any reason you must make a copy of the matrix. See also `widths`.

pointSize

```
- (float)pointSize
```

Returns the font size, in points.

positionOfGlyph:precededByGlyph:isNominal:

```
- (NSPoint)positionOfGlyph:(NSGlyph)curGlyph
    precededByGlyph:(NSGlyph)prevGlyph isNominal:(BOOL *)nominal
```

Returns `curGlyph`'s position when it follows `prevGlyph`. `nominal` is a pointer to a `BOOL`. If not `nil`, this method fills in `nominal` with YES, to indicate that the position has been modified by kerning information, and NO to indicate that no kerning information was present.

printerFont

```
– (NSFont *)printerFont
```

Returns the printer font for the font object, if the receiving font object is a screen font. Otherwise this method returns `self`. See also `screenFont`.

screenFont

```
– (NSFont *)screenFont
```

Returns the screen font for the font object, if there is one. Otherwise this method returns `self`. See also `printerFont`.

set

```
– (void)set
```

Makes this font the graphic state's current font. When a font object receives a `set` message, it registers with the font class object that its PostScript font has been used. In this way, the Application Kit, when called upon to generate a conforming PostScript language document file, can list the fonts used within a document. (See *Document Structuring Conventions* by Adobe Systems Inc.) If the application uses fonts without sending set messages (say through including an EPS file), such fonts must be registered by sending the class object a `useFont:` message. See also `setUserFont:`, `setUserFixedPitchFont:`, `useFont:`.

underlinePosition

```
- (float)underlinePosition
```

Returns the distance from the font baseline to the top of the underline. See also `underlineThickness`.

underlineThickness

```
- (float)underlineThickness
```

Returns the underline thickness. See also `underlinePosition`.

widthOfString:

```
- (float)widthOfString:(NSString *)string
```

Returns the width of `string` using this font. Use this method with caution: it assumes that the characters in `string` can all actually be rendered in the font. It uses lossy encoding methods in `NSString` to get the character data.

widths

```
- (float *)widths
```

Returns a pointer to an array of 256 unscaled widths of the glyphs in the font.

xHeight

```
- (float)xHeight
```

Returns the nominal height above the baseline of the lowercase font glyphs. This is defined as the height of the Latin lowercase x where applicable.

NSFontManager

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

Class Description

`NSFontManager` declares the programmatic interface to objects that manage font conversion in an application. `NSFontManager` is the center of activity for font conversion. `NSFontManager` accepts messages from font conversion user-interface objects such as the Font menu or the Font panel (see `NSFontPanel` for more details) and appropriately converts the current font in the selection by sending a `changeFont:` message (see the `NSText` class) up the responder chain.

When an object receives a `changeFont:` message, it should message `NSFontManager` (by sending it a `convertFont:` message), asking it to convert the font in whatever way the user has specified. Thus, any object

containing a font that can be changed should respond to the `changeFont:` message by sending a `convertFont:` message back to the `NSFontManager` for each font in the selection.

To use `NSFontManager`, you simply insert a Font menu into your application's menu using the appropriate interface construction tools (such as Interface Builder). You can also obtain a Font menu by sending a `getFontMenu:` message to `NSFontManager` and then inserting the menu it returns into the application's main menu. Once the Font menu is installed, your application automatically gains the functionality of both the Font menu and the Font panel.

`NSFontManager`'s delegate can restrict which font names will appear in the Font panel. See "Methods Implemented by the Delegate" at the end of this class specification for more information.

`NSFontManager` can be used to convert a font or find out the attributes of a font. It can also be overridden to convert fonts in some application-specific manner. The default implementation of font conversion is very conservative: The font isn't converted unless all traits of the font can be maintained across the conversion.

Generally, you obtain an instance of `NSFontManager` by sending a `sharedFontManager` message to the `NSFontManager` class object. `NSFontManager` will return a font manager object that is shared within your application. `NSFontManager` normally returns a predefined font manager object, but the actual object which is returned can be changed by sending the `setFontManagerFactory:` message.

Font Traits

Fonts work mainly in terms of *traits*, or characteristics, such as bold, italic, condensed, and so on. Traits are described by a collection of constants such as `NSItalicFontMask`, `NSBoldFontMask`, and so on. The full complement of traits are defined in `AppKit/NSFontManager.h`. The values of traits are defined in bitwise form so they can be logically OR'ed together, although some traits, such as `NSBoldFontMask` and `NSUnboldFontMask` naturally conflict and have the effect of turning each other off. You use one of the `convertFont` methods to obtain a font of the desired characteristics from an existing font.

The `convertFont:toHaveTrait:` and the `convertFont:toNotHaveTrait:` methods deal with only one trait at a time. To convert a font to have (or not have) multiple traits, you must invoke these methods for each separate trait you wish to add to or remove from the font. Alternatively, use the `fontWithFamily:traits:weight:size:` method to specify multiple traits in one invocation.

The `size` of a font in the method definitions in the following is defined in "points", which are currently PostScript units. A PostScript unit is 1/72 of an inch, or 0.0139 of an inch. In metric equivalents, a PostScript unit is 0.3528 millimetres. PostScript "points" are minimally different from "printer's points", so for all intents and purposes you can think of PostScript units and points as interchangeable.

The `weight` of a font as used in these methods is simply a value representing a point in a continuum of font weights from lightest to heaviest. There's no simple one-to-one mapping of some integer value to, say, a **bold** weight. If you query the font for its weight value, increment the value, and use it as a new weight, you'll not necessarily obtain a different face (such as a transition from medium to bold) in a new instance of the font.

Method Types

Activity	Class Method
Managing the font manager	+ setFontManagerFactory: + setFontPanelFactory: + sharedFontManager
Converting fonts	– addFontTrait: – convertFont: – convertFont:toFamily: – convertFont:toFace: – convertFont:toHaveTrait: – convertFont:toNotHaveTrait: – convertFont:toSize: – convertWeight:ofFont: – fontWithFamily:traits:weight:size: – modifyFont: – modifyFontViaPanel: – removeFontTrait:
Setting and getting parameters	– action – availableFontNamesWithTraits: – availableFonts – fontNamed:hasTraits: – fontMenu: – fontPanel: – isEnabled – isMultiple – orderFrontFontPanel: – selectedFont – setAction: – setEnabled: – setFontMenu: – setSelectedFont:isMultiple: – traitsOfFont: – weightOfFont:
Target and action methods	– sendAction
Assigning a delegate	– delegate – setDelegate:
Methods Implemented by the Delegate	– fontManager:willIncludeFont:

Class Methods

setFontManagerFactory:

```
+ (void)setFontManagerFactory:(Class)classId
```

Sets the class object that will be used to create the font manager, allowing you to specify a class of your own. See also `setFontPanelFactory:`.

setFontPanelFactory:

```
+ (void)setFontPanelFactory:(Class)classId
```

Sets the class object that's used to create the `NSFontPanel` object when the user chooses the Font panel command from the Font menu and no such panel has yet been created. Unless you use this method to specify another class, the `NSFontPanel` class will be used. See also `setFontManagerFactory:`.

sharedFontManager

```
+ (NSFontManager *)sharedFontManager
```

Returns a shared `NSFontManager` object, and also creates a shared `NSFontPanel` object if necessary. See the "Class Description" for more information on the shared font manager. See also `setFontManagerFactory:`, `setFontPanelFactory:`.

Instance Methods

action

```
- (SEL)action
```

Returns the action message that's sent by the font manager to the first responder when the user selects a new font from the Font panel or from the Font menu. See also `setAction:`, `sendAction`.

addFontTrait:

```
- addFontTrait:(id)sender
```

Causes the font manager's action message (by default, `changeFont:`) to be sent up the responder chain. When the responder replies with a `convertFont:` message, the font is converted to add the trait specified by `sender`.

Before the action message is sent up the responder chain, the Font Manager ascertains the trait to be changed by send `sender` a `selectedTag` (`NSControl`) message. When the `convertFont:` message is received, the font manager converts the supplied font by sending itself a `convertFont:toHaveTrait:` message. See the Fonts section of the Application Kit's Types and Constants chapter for a list of Font Manager tags. See also `removeFontTrait:`, `convertFont:toHaveTrait:`, `changeFont:` (`NSText`).

availableFontNamesWithTraits:

```
- (NSArray *)availableFontNamesWithTraits:
    (NSFontTraitMask)fontTraits
```

Searches for fonts with the given font traits, and returns those font names found. The font traits are:

- `NSItalicFontMask`
- `NSBoldFontMask`
- `NSUnboldFontMask`
- `NSNonStandardCharacterSetFontMask`
- `NSNarrowFontMask`
- `NSExpandedFontMask`
- `NSCondensedFontMask`
- `NSSmallCapsFontMask`
- `NSPosterFontMask`
- `NSCompressedFontMask`
- `NSUnitalicFontMask`
- `NSUnitalicFontMask`
- `NSFixedPitchFontMask`

See also `fontNamed:hasTraits:`.

availableFonts

```
– (NSArray *)availableFonts
```

Returns array listing all the fonts available for use by the Window Server. The returned names are suitable for creating new NSFonts. The fonts are not in any guaranteed order, but no font name is repeated in the list. It's the sender's responsibility to free the list when finished with it.

convertFont:

```
- (NSFont *)convertFont:(NSFont *)fontObject
```

Converts fontObject according to the user's selections from the Font panel or the Font menu. Returns the converted font. See also convertFont:toFamily:, convertFont:toFace:, convertFont:toHaveTrait:, convertFont:toNotHaveTrait:, convertFont:toSize:.

convertFont:toFamily:

```
- (NSFont *)convertFont:(NSFont *)fontObject
    toFamily:(NSString *)family
```

Returns an NSFont object whose traits are the same as those of fontObject except as specified by family. If the conversion can't be made, the method returns fontObject itself. This method can be used to convert a font, or it can be overridden to convert fonts in a different manner. See also convertFont:.

convertFont:toFace:

```
- (NSFont *)convertFont:(NSFont *)fontObject
    toFace:(NSString *)typeface
```

Returns an NSFont object whose traits are the same as those of fontObject except as specified by typeface. If the conversion can't be made, the method returns fontObject itself. This method can be used to convert a font, or it can be overridden to convert fonts in a different manner. See also convertFont:.

convertFont:toHaveTrait:

```
- (NSFont *)convertFont:(NSFont *)fontObject
    toHaveTrait:(NSFontTraitMask)trait
```

Returns a NSFont object whose traits are the same as those of fontObject except as altered by the addition of the traits specified by trait. Of course, conflicting traits (such as NSCondensedFontMask and NSExpandedFontMask) have the effect of turning each other off. If the conversion can't be made, the method returns fontObject itself. This method can be overridden to convert fonts in a different manner. See the "Fonts" section of the Application Kit's "Types and Constants" chapter for a list of font trait masks. See also convertFont:toNotHaveTrait:.

convertFont:toNotHaveTrait:

```
- (NSFont *)convertFont:(NSFont *)fontObject
    toNotHaveTrait:(NSFontTraitMask)trait
```

Returns an NSFont object whose traits are the same as those of fontObject except as altered by the removal of the traits specified by trait. If the conversion can't be made, the method returns fontObject itself. This method can be overridden to convert fonts in a different manner. See also convertFont:toHaveTrait:.

convertFont:toSize:

```
(NSFont *)convertFont:(NSFont *)fontObject toSize:(float)size
```

Returns an NSFont object whose traits are the same as those of fontObject except as specified by size. If the conversion can't be made, the method returns fontObject itself. This method can be used to convert a font, or it can be overridden to convert fonts in a different manner. See also convertFont:.

convertWeight:ofFont:

```
- (NSFont *)convertWeight:(BOOL)upFlag ofFont:(NSFont *)fontObject
```

Attempts to increase (if upFlag is YES) or decrease (if upFlag is NO) the weight of the font specified by fontObject. If it can change the font weight, it returns a new font object with the higher (or lower) weight. If it can't, it returns fontObject itself. By default, this method converts the weight only if it can maintain all of the traits of the original fontObject. This method can be overridden to convert fonts in a different manner. See also convertFont:.

delegate

```
- (id)delegate
```

Returns the NSFontManager's delegate. See also setDelegate:.

fontMenu:

```
- (NSMenu *)fontMenu:(BOOL)create
```

Returns a menu suitable for insertion in an application's menu. The menu contains an item that brings up the Font panel as well as some common accelerators (such as Bold and Italic). If the create flag is YES, the menu is created if it doesn't already exist. See also setFontMenu:, fontPanel:.

fontNamed:hasTraits:

```
- (BOOL)fontNamed:(NSString *)name
    hasTraits:(NSFontTraitMask)traits
```

Retrieves font name's font traits. Returns NO if font name is not found. See also availableFontNamesWithTraits:.

fontPanel:

```
- (NSFontPanel *)fontPanel:(BOOL)create
```

Returns the NSFontPanel that will be used when the user chooses the Font Panel command from the Font menu. If the create flag is YES, the NSFontPanel is created if it doesn't already exist. Unless you've specified a different class by sending a setFontPanelFactory: message to the NSFontManager class before creating the NSFontManager object, an NSFontPanel object is returned. See also fontMenu:.

fontWithFamily:traits:weight:size:

```
- (NSFont *)fontWithFamily:(NSString *)family
    traits:(NSFontTraitMask)traits
    weight:(int)weight size:(float)size
```

If there's a font on the system with the specified `family`, `traits`, `weight`, and `size`, then it's returned; otherwise, `nil` is returned. If `NSBoldFontMask` or `NSUnboldFontMask` is one of the traits, `weight` is ignored. See the Fonts section of the Application Kit's Types and Constants chapter for a list of font masks.

isEnabled

```
- (BOOL)isEnabled
```

Returns `YES` if the Font panel and menu are enabled, and returns `NO` otherwise. See also `setEnabled:`.

isMultiple

```
- (BOOL)isMultiple
```

Returns `YES` if the currently selected text contains multiple fonts, and returns `NO` otherwise. See also `setSelectedFont:isMultiple:`.

modifyFont:

```
- modifyFont:(id)sender:
```

Causes the font manager's action message (by default `NSText`'s `changeFont:` method) to be sent up the responder chain. When the responder replies with a `convertFont:` message, the font is converted in a way specified by the `selectedTag` of the sender of this message. The Font menu items invoke this method. See also `addFontTrait:`, `removeFontTrait:`, `changeFont:` (`NSText`).

modifyFontViaPanel:

```
- modifyFontViaPanel:(id)sender:
```

Causes the font manager's action message (by default, `changeFont:`) to be sent up the responder chain. When the receiver replies with a `convertFont:` message, the font manager sends a `panelConvertFont:` message to the Font panel to complete the conversion.

This message is usually sent by a control object in the Font panel. The Font panel uses the font manager's convert routines to do the conversion based on the choices the user has made on the Font panel.

orderFrontFontPanel:

```
- orderFrontFontPanel:(id)sender:
```

Sends `orderFront:` (`NSWindow`) to the font panel. If there is no font panel, it is created, by the `NSFontPanel` class object, or by an object you specified with the font manager's `setFontPanelFactory:` class method. See also `orderFront:` (`NSWindow`).

removeFontTrait:

```
- removeFontTrait:(id)sender:
```

Sends the font manager's action message (by default `NSText`'s `changeFont:` method) up the responder chain. When the responder replies with a `convertFont:` message, the font is converted to remove the trait specified by `sender`. When the `convertFont:` message is received, the font manager converts the supplied font by sending itself a `convertFont:toNotHaveTrait:` message. See also `addFontTrait:`, `convertFont:toHaveTrait:`.

selectedFont

```
– (NSFont *)selectedFont
```

Returns the first font in the current selection. See also `setSelectedFont:isMultiple:`.

sendAction

```
– (BOOL)sendAction
```

Sends the `NSFontManager`'s action message up the responder chain. You rarely, if ever, need to send a `sendAction:` message or to override this method. The message is sent by the target-action messages sent by different user-interface objects that allow users to manipulate the font of the current text selection for example, the Font panel and the Font menu. See also `setAction:`, `action`.

setAction:

```
- (void)setAction:(SEL)aSelector
```

Sets the action to that specified by `aSelector` to be sent by the font manager when the user selects a new font from the Font panel or from the Font menu. See also `action`, `sendAction`.

setDelegate:

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

Sets the `NSFontManager`'s delegate to `anObject`. The delegate can restrict which font names appear in the Font panel. See also `delegate`.

setEnabled:

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

Sets whether the controls in the Font panel and the commands in the Font menu are enabled or disabled depending on `flag`. By default, these controls and commands are enabled. Even when disabled, the Font panel allows the user to preview fonts. However, when the Font panel is disabled, the user can't apply the selected font to text in the application's main window. You can use this method to disable the user interface to the font selection system when its actions would be inappropriate. For example, you might disable the font selection system when your application has no document window. See also `isEnabled`.

setFontMenu:

```
- (void)setFontMenu:(NSMenu *)newMenu
```

Sets the font menu to `newMenu`. See also `fontMenu:`.

setSelectedFont:isMultiple:

```
- (void)setSelectedFont:(NSFont *)fontObject isMultiple:(BOOL)flag
```

Notifies font manager of the selection's current font from `fontObject` with `flag` indicating whether the selection has multiple fonts. An object containing a document should send this message every time its selection changes. If the selection contains multiple fonts, `flag` should be YES. See also `selectedFont`.

`traitsOfFont:`

```
- (NSFontTraitMask)traitsOfFont:(NSFont *)fontObject
```

Returns the font traits of `fontObject`. See the Fonts section of the Application Kit's Types and Constants chapter for more information on font masks. See also `fontWithFamily:traits:weight:size:`.

`weightOfFont:`

```
- (int)weightOfFont:(NSFont *)fontObject
```

Returns the `fontObject` weight.

Methods Implemented by the Delegate

`fontManager:willIncludeFont:`

```
- (BOOL)fontManager:(id)sender willIncludeFont:(NSString *)fontName
```

Responds to a message informing the `NSFontManager`'s delegate that the `NSFontPanel` is about to include `fontName` in the list displayed to the user. `fontName` is the name of the font, for example "Helvetica-Narrow-Bold". If this method returns NO, the font isn't added; otherwise, it is.

A delegate that implements this method can receive multiple `fontObject` messages whenever the Font panel needs updating, such as when the user selects a different family name to determine which typefaces are available. For each typeface within that family, the delegate will receive notification. Consequently, your implementation of this method shouldn't take long to execute.

1

NSFontPanel

Inherits From:	NSPanel : NSWindow : NSResponder : NSObject
Conforms To:	NSCoding (NSResponder) NSObject (NSObject)
Declared In:	AppKit / NSFontPanel.h

Class Description

The `NSFontPanel` class declares the programmatic interface to a user-interface object that displays a list of available fonts, enabling users to preview them and change the typefaces in which text is displayed. Actual changes to text are effected through conversion messages sent to the `NSFontManager`. There is only one `NSFontPanel` object for each application.

In general, you add the facilities of the `NSFontPanel` and of the other components of the font conversion system, the `NSFontManager` and the Font menu to your application through interface construction tools such as Interface Builder. You do this by including a Font menu into one of your application's menus. At run time, when the user chooses the Font Panel command for the first time, the `NSFontPanel` object is created and hooked into the font conversion system. You can also create (or access) `NSFontPanel` through the `sharedFontPanel` method.

An `NSFontPanel` can be customized by adding an additional `NSView` object or hierarchy of `NSView` objects by using the `setAccessoryView:` method. If you want the `NSFontManager` to instantiate a panel object from some class other than `NSFontPanel`, use the `NSFontManager`'s `setFontPanelFactory:` method. See `NSFontManager` for details on the font manager object that performs font conversion tasks.

Method Types

Activity	Class Method
Creating an NSFontPanel	+ sharedFontPanel – panelConvertFont:
Setting the Font	– setPanelFont: isMultiple:
Configuring the NSFontPanel	– accessoryView – isEnabled – setAccessoryView: – setEnabled: – worksWhenModal
Displaying the NSFontPanel	– orderWindow:relativeTo:

Class Methods

sharedFontPanel

```
+ (NSFontPanel *)sharedFontPanel
```

Returns an `NSFontPanel` object. The panel is created if it doesn't already exist.

Instance Methods

accessoryView

```
- (NSView *)accessoryView
```

Returns the application-customized view set by `setAccessoryView:`.

isEnabled

```
- (BOOL)isEnabled
```

Returns `YES` if the `NSFontPanel`'s Set button is enabled, and returns `NO` otherwise. See also `setEnabled:`.

orderWindow:relativeTo:

```
- (void)orderWindow:(NSWindowOrderingMode)place
    relativeTo:(int)otherWindows
```

Repositions the NSFontPanel above or below the other windows otherWindows as indicated by place and updates the NSFontPanel if necessary. place can be one of:

- NSWindowAbove
- NSWindowBelow
- NSWindowOut

If it's NSWindowOut, the panel is removed from the screen list and otherWin is ignored. If it's NSWindowAbove or NSWindowBelow, otherWin is the window number of the window that the NSFontPanel is to be placed above or below. If otherWin is 0, the panel will be placed above or below all other windows. See also orderWindow:relativeTo: (NSWindow), makeKeyAndOrderFront: (NSWindow).

panelConvertFont:

```
- (NSFont *)panelConvertFont:(NSFont *)fontObject
```

Returns an NSFont object whose traits are the same as those of fontObject except as specified by the user's choices in the Font Panel. If the conversion can't be made, the method returns fontObject itself. The NSFontPanel makes the conversion by using the NSFontManager's methods that convert fonts. A panelConvertFont: message is sent by the NSFontManager whenever it needs to convert a font as a result of user actions in the Font panel. See also sharedFontPanel.

setAccessoryView:

```
- (void)setAccessoryView:(NSView *)aView
```

Customizes the Font panel by adding aView above the action buttons at the bottom of the panel. The NSFontPanel is automatically resized to accommodate aView. aView should be the top NSView in a view hierarchy. If aView is nil, any existing accessory view is removed. If aView is the same as the current accessory view, this method does nothing. See also accessoryView.

setEnabled:

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

Enables (the default state) or disables the FontPanel's Set button depending on *flag*. Even when disabled, the Font panel allows the user to preview fonts. However, when the Font panel is disabled, the user can't apply the selected font to text in the application's main window. You can use this method to disable the user interface to the font selection system when its actions would be inappropriate. For example, you might disable the font selection system when your application has no document window. See also `isEnabled`.

setPanelFont: isMultiple:

```
- (void)setPanelFont:(NSFont *)fontObject  isMultiple:(BOOL)flag
```

Sets the `NSFontPanel`'s current font from `fontObject` with `flag` indicating whether it contains multiple fonts. This message should *only* be sent by the `NSFontManager`.

worksWhenModal

```
- (BOOL)worksWhenModal
```

Returns whether the `NSFontPanel` will operate while a modal panel is displayed within the application. By default, this method returns `YES`. See also `worksWhenModal` (`NSPanel`).

NSForm

Inherits From:	NSMatrix : NSControl : NSView : NSResponder : NSObject
Conforms To:	NSCoding (NSResponder) NSObject (NSObject)
Declared In:	AppKit/NSForm.h

Class Description

An `NSForm` is an `NSMatrix` subclass that contains titled entries (text fields) into which a user can type data values. Entries are indexed from the top down (starting with zero). Each item in the `NSForm`, including the titles, is an

NSFormCell. A mouse click on an NSFormCell (that is, on the title or in the entry area) starts text editing in that entry. If the user presses the Return or Enter key while editing an entry, the action of the entry is sent to the target of the entry; or, if the entry doesn't have an action, the NSForm sends its action to its target. If the user presses the Tab key, the next entry in the NSForm is selected; if the user presses Shift-Tab, the previous entry is selected.

For more information, see the NSFormCell and NSMatrix class specifications.

Method Types

Activity	Class Method
Laying out the form	– addEntry: – insertEntry:atIndex: – removeEntryAtIndex: – setInterlineSpacing:
Finding indices	– indexOfCellWithTag: – indexOfSelectedItem
Modifying graphic attributes	– setBezeled: – setBordered: – setTextAlignment: – setTextFont: – setTitleAlignment: – setTitleFont:
Setting the cell class	+ cellClass + setCellClass:
Getting a cell	– cellAtIndex:
Displaying a cell	– drawCellAtIndex:
Editing a cell	– selectTextAtIndex:
Resizing the form	– setEntryWidth:

Class Methods

cellClass

```
+ (Class)cellClass
```

Returns the class last set in a `setCellClass:` message, or the `NSFormCell` class if `setCellClass:` has never been called. See also `setCellClass:`.

setCellClass:

```
+ (void)setCellClass:(Class)classId
```

Configures the `NSForm` class to use instances of `classId` for its cells. `classId` should be an `NSFormCell` subclass `id`, obtained by sending the `class` message (`NSObject`) to either the `NSFormCell` subclass object, or to an instance of that subclass. The default cell class is `NSFormCell`. "Creating New NSControls" in the `NSControl` class specification has more information on how to safely set the cell class used by a subclass of `NSControl`. See also `cellClass`, `initWithFrame:`.

Instance Methods

addEntry:

```
- (NSFormCell *)addEntry:(NSString *)title
```

Adds and returns a new entry, with `title` as its title, at the end of the form. See also `insertEntry:atIndex:`, `removeEntryAtIndex:`, `setInterlineSpacing:`.

cellAtIndex:

```
- (id)cellAtIndex:(int)index
```

Returns the cell at location `index` or `nil` if none exists. See also `cellAtRow:column:` (`NSMatrix`).

drawCellAtIndex:

```
- (void)drawCellAtIndex:(int)index
```

Displays the cell at the specified `index`. See also `drawCellAtRow:column:` (`NSMatrix`).

indexOfCellWithTag:

```
- (int)indexOfCellWithTag:(int)aTag
```

Returns the index for the cell with tag `aTag`, or -1 if none exists. See also `indexOfSelectedItem`.

indexOfSelectedItem

```
- (int)indexOfSelectedItem
```

Returns the index of the currently selected entry, or -1 if none is selected. See also `indexOfCellWithTag:`.

insertEntry:atIndex:

```
- (NSFormCell *)insertEntry:(NSString *)title atIndex:(int)index
```

Inserts a new entry, with the title `title`, at position `index` in the form. The entry at the top of the form has an index of 0. The new `NSFormCell` has no tag, target, or action. Returns the newly inserted `NSFormCell`. Does not redraw the form. See also `removeEntryAtIndex:`, `addEntry:`.

removeEntryAtIndex:

```
- (void)removeEntryAtIndex:(int)index
```

Removes the entry at location `index`. See also `insertEntry:atIndex:`, `addEntry:`.

selectTextAtIndex:

```
- (void)selectTextAtIndex:(int)index
```

If given a valid `index`, selects the text in the entry at `index`.

setBezeled:

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

If `flag` is `YES`, all cells in the form are set to show a bezel around their editable text and are redrawn; if `flag` is `NO`, cells in the form have no bezel. A bezel is mutually exclusive with a border, and invoking this method with `NO` as the argument will not remove a border. See also `setBordered:`.

setBordered:

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

If `flag` is `YES`, all cells in the form are set to show a one-pixel black border around their editable text and are redrawn; if `flag` is `NO`, cells in the form have no border. A border is mutually exclusive with a bezel, and invoking this method with `NO` as the argument will not remove a bezel. See also `setBezeled:`.

setEntryWidth:

```
- (void)setEntryWidth:(float)width
```

Sets the width of all the entries (including the title part) to `width`. Doesn't redraw the form. Invoke `sizeToCells` (`NSMatrix`) after using this method.

setInterlineSpacing:

```
- (void)setInterlineSpacing:(float)spacing
```

Sets the spacing between entries to `spacing`. Does not redraw the form. See also `addEntry:`.

setTextAlignment:

```
- (void)setTextAlignment:(NSTextAlignment)mode
```

Sets the alignment mode for the editable text in the form. `mode` can be one of three values:

- `NSLeftTextAlignment`
- `NSCenteredTextAlignment`
- `NSRightTextAlignment`

The default is left aligned. Redraws the form. See also `setTextFont:`, `setTitleAlignment:`, `setTitleFont:`.

setTextFont:

```
- (void)setTextFont:(NSFont *)fontObject
```

Sets the font used to draw entry text to `fontObject`. Marks the form as needing redrawing. See also `setTitleFont:`, `setTextAlignment:`.

setTitleAlignment:

```
- (void)setTitleAlignment:(NSTextAlignment)mode
```

Sets the alignment mode for titles in the form. `mode` can be one of three values:

- `NSLeftTextAlignment`
- `NSCenteredTextAlignment`
- `NSRightTextAlignment`

The default is right aligned. See also `setTextAlignment:`.

setTitleFont:

```
- (void)setTitleFont:(NSFont *)fontObject
```

Sets the font used to draw entry titles to `fontObject`. Redraws the form. See also `setTitleAlignment:`, `setTextFont:`.

NSFormCell

Inherits From:	NSActionCell : NSCell : NSObject
Conforms To:	NSCoding, NSCopying (NSCell) NSObject (NSObject)
Declared In:	AppKit/NSFormCell.h

Class Description

`NSFormCell` is used to implement entries in an `NSForm`. It displays a title within itself, on the left side of the cell. Editing is allowed only in the remaining (right) portion. See the `NSForm` class specification for more on the use of `NSFormCell`.

Method Types

Activity	Class Method
Initializing an NSFormCell	– initTextCell:
Determining an NSFormCell's size	– cellSizeForBounds:
Determining graphic attributes	– isOpaque
Modifying the title	– setTitle: – setTitleAlignment: – setTitleFont: – setTitleWidth: – title – titleAlignment – titleFont – titleWidth – titleWidth:
Displaying	– drawInteriorWithFrame:inView:

Instance Methods

cellSizeForBounds:

```
– (NSSize)cellSizeForBounds:(NSRect)aRect
```

Calculates the `NSFormCell`'s size, assuming it is constrained within `aRect`. Returns the size.

drawInteriorWithFrame:inView:

```
– (void)drawInteriorWithFrame:(NSRect)cellFrame
    inView:(NSView *)controlView
```

Draws only the text inside the `NSFormCell` (not the bezel or the title of the `NSFormCell`) within the given `cellFrame`, for the given `controlView`.

initTextCell:

```
– (id)initTextCell:(NSString *)aString
```

Initializes and returns the receiver, a new instance of NSFormCell, with its contents set to the empty string ("") and its title set to aString. The font for both title and text is the user's chosen system font in 12.0 point, and the text area is drawn with a bezel. This method is the designated initializer for NSFormCell.

isOpaque

```
- (BOOL)isOpaque
```

Returns YES if the NSFormCell is opaque, NO otherwise. If the NSFormCell has a title, then it's not opaque (since the title field is not opaque).

setTitle:

```
- (void)setTitle:(NSString *)aString
```

Sets the NSFormCell's title to aString. See also title, setTitleAlignment:, setTitleFont:, setTitleWidth:, titleAlignment, titleFont, titleWidth, titleWidth:.

setTitleAlignment:

```
- (void)setTitleAlignment:(NSTextAlignment)mode
```

Sets the alignment of the title to mode which can be one of the following:

- NSLeftTextAlignment
- NSCenterTextAlignment
- NSRightTextAlignment

See also setTitle:.

setTitleFont:

```
- (void)setTitleFont:(NSFont *)fontObject
```

Sets the font used to draw the title to fontObject. See also setTitle:.

setTitleWidth:

```
- (void)setTitleWidth:(float)width
```

Sets the `width` of the title field. If `width` is –1, the title field's width is always calculated when needed. Use this method only if the `NSFormCell`'s title isn't going to change, or if your code always resets the title width when it resets the title. See also `setTitle:`.

title

```
- (NSString *)title
```

Returns the `NSFormCell`'s title. See also `setTitle:`.

titleAlignment

```
-  (NSTextAlignment)titleAlignment
```

Returns the alignment of the title. See also `setTitleAlignment:`.

titleFont

```
- (NSFont *)titleFont
```

Returns the font used to draw the title. See also `setTitleFont:`, `setTitle:`.

titleWidth

```
- (float)titleWidth
```

Returns the title width. See also `setTitleWidth:`, `setTitle:`, `titleWidth:`.

titleWidth:

```
- (float)titleWidth:(NSSize)aSize
```

If the title width has been set, then it's returned. Otherwise, the width is calculated constrained to `aSize`. `aSize` may be `NULL`, in which case the width is calculated without constraint. See also `setTitleWidth:`, `setTitle:`, `titleWidth`.

1

NSHelpPanel

Inherits From:	NSPanel : NSWindow : NSResponder : NSObject
Conforms To:	NSCoding (NSResponder) NSObject (NSObject)
Declared In:	AppKit/NSHelpPanel.h

Class Description

The `NSHelpPanel` class is the central component of the OpenStep help system. It provides the Help panel that displays the text and illustrations that constitute your application's help information. The `NSHelpPanel` class object itself stores the table of associations between an application's user-interface objects and specific passages of the help text.

Users can display the Help panel by choosing the Help command from an application's Info menu. The panel employs the metaphor of a book: It displays a table of contents, body text, and an index. Users can browse through the text by clicking entries in the table of contents or index. The panel also supports hypertext-like help links, which appear as diamond-shaped images within the text and allow the user to easily follow cross references. By using the help cursor and clicking user-interface objects, the user can query the Help panel for information associated with those objects.

The Help Text

An `NSHelpPanel` object looks in a language-specific directory within the application's file package for the text that it will display. (Some implementations may employ more efficient means of storage than files and directories.) For example, if the user's language preference is English, the panel searches for a directory named `Help` within the `C.lproj` directory of the application's file package. It searches for two files: `TableOfContents.rtf` and `Index.rtfd`. There may also be one or more files containing the body text that the Help panel will display. The table of contents, index, and body files are interconnected by a system of *help links* and *help markers*.

A help marker is a named position holder in the stream of text—in most cases, it's invisible to users. A help link is a diamond-shaped button embedded in the text. Help links store a file name and, optionally, a help marker name. When a

user clicks a help link, the Help panel displays the named file. If the help link also stores a marker name, the displayed file is scrolled to the position of the marker, and the text is selected from the marker's position to the end of the line.

Table-of-Contents and Index Files

The table-of-contents and index files are specially designed documents in Rich Text Format (RTF). An `NSHelpPanel` object identifies these files by name (`TableOfContents.rtf` and `Index.rtfd`) and processes them differently than it does other help files.

The table of contents file should contain one entry for each help text file in the help directory. Each entry begins with a help link that stores the name of the destination file for that entry. Following the link is the text of the entry, which may wrap and span several lines. Although the table of contents in the Help panel looks like it's displayed by an `NSMatrix`, it's actually displayed by a modified `NSText` object. You can use the full generality of RTF to format your table of contents.

The index file is structured similarly although there is no enforced one-to-one mapping. Generally, the help link that begins an index entry stores both a file name and a marker name, since an index entry usually points to a specific word or phrase within a file.

Generic Help Files

An application's Help directory can contain only table-of-contents and index files, and yet the application may be able to display numerous help subjects, each of a general nature. This is because OpenStep applications have access to generic help files contained in a directory found in a system-specific location.

When a help link is being resolved, the `NSHelpPanel` first looks for the specified file within the appropriate *language*`.lproj/Help` directory of the application's file package. If the file isn't found, it then searches the directory of generic help files. This search path is used for all links, whether they are in the table of contents, index, or body text.

If one of these generic help files is inappropriate for your application, you have two remedies: You can remove the table-of-contents and index entries that refer to it, or you can override the file with one that's more appropriate. By placing a file of the same name and relative location within your application's `Help` directory, `NSHelpPanel` will display it rather than the generic file.

Associating Help Text with Objects

The `NSHelpPanel` class stores associations between user-interface objects and help text. When the user presses the Help modifier key, which varies depending on the hardware running the application), a question mark cursor appears. If the user clicks an object using this cursor, the Help panel displays the associated help text.

You can attach a help file to a user-interface object programmatically by sending an `attachHelpFile:markerName:to:` message to the `NSHelpPanel` class object. This method takes a file name, a marker name, and an object `id` as its arguments. The `detachHelpFrom:` message removes such an association.

Just as with help links, an `NSHelpPanel` searches both the application's file package and the generic help files in attempting to find the file associated with a particular user-interface object.

Hidden Files

Although in general there's a one-to-one relationship between table-of-contents entries and files in the `Help` directory, you can force a single table-of-contents entry to represent multiple "hidden" files. This can be useful in reducing the overall length of the table of contents.

Hidden files can't be accessed from the table of contents; rather, the user must find them by Help-clicking an object in the application's user interface, by using the Help panel's Find command, by using the index, or by following a help link from some other file. However, when a hidden file is displayed, the Help panel must select some entry in the table of contents.

Conversely, when the user selects such a table-of-contents entry, the Help panel must display one of the files in the directory of hidden files; by convention, this file must be named `Prolog.rtfd`. This prolog file typically informs users that they can get help on a particular user-interface object by Help-clicking that object.

The Help panel's Find button searches through all the files that are connected to table-of-contents entries, first looking in the application's `Help` directory and then in the generic help material. If you don't want some hidden file in the generic help material to appear in your application's Help panel as the result of a Find operation, override the file with an empty file of the same name. Since the file is empty, no search string will ever be found in it, and it will effectively block the generic file of the same name from being searched.

Searching the Help Text

By clicking the Help panel's Find button, users can search the help text for strings. `NSHelpPanel` uses two approaches to locate text containing a specific string. First, it attempts to find the string in the currently displayed help text by sending the object that displays the text (an instance of `NSCStringText`) a `findText:ignoreCase:backwards:wrap:` message. If the search is unsuccessful, or if the search is continued past the last occurrence of the string in the current file, the `NSHelpPanel` object scans for the string in other help files, both within the application's help files and within the generic help files. Some implementations of `NSHelpPanel` may make use of a previously built index of all the help text to speed this search.

Help Supplements

Since in OpenStep an application may load executable modules dynamically (for example, a drawing program could allow the user to load a new drawing tool), an `NSHelpPanel` object provides the ability to load supplemental help information. When the application loads the module, it sends the `NSHelpPanel` object an `addSupplement:inPath:` message to inform the object of the location of the new help supplement. The `NSHelpPanel` object appends the contents of the supplement's `TableOfContents.rtf` to the existing table of contents, so the supplement should have a title that clearly sets it off from the main part of the table of contents, for example:

```
—Pattern Tool Supplement—
    Pattern Options
        Brick
        Stucco
        Wood
        Tile
    Custom
    Resizing and Rotating
    Index to Supplement
```

The supplement's index is only accessible from the table of contents; the Help panel's Index button displays only the main index.

Method Types

Activity	Class Method
Accessing the help panel	+ sharedHelpPanel + sharedHelpPanelWithDirectory:
Managing the contents	+ setHelpDirectory: – addSupplement:inPath: – helpDirectory – helpFile
Attaching help to objects	+ attachHelpFile:markerName:to: + detachHelpFrom:
Showing help	– showFile:atMarker: – showHelpAttachedTo:
Printing	– print:

Class Methods

attachHelpFile:markerName:to:

```
+ (void)attachHelpFile:(NSString *)filename
    markerName:(NSString *)markerName to:(id)anObject
```

Associates the help file `filename` and `markerName` with `anObject`.

detachHelpFrom:

```
+ (void)detachHelpFrom:(id)anObject
```

Removes any help information associated with `anObject`.

setHelpDirectory:

```
+ (void)setHelpDirectory:(NSString *)helpDirectory
```

Initializes the panel to display the help text found in `helpDirectory`. By default, the receiver looks for a directory named `Help`.

sharedHelpPanel

```
+ (NSHelpPanel *)sharedHelpPanel
```

Creates, if necessary, and returns the `NSHelpPanel` object.

sharedHelpPanelWithDirectory:

```
+ (NSHelpPanel *)sharedHelpPanelWithDirectory:
    (NSString *)helpDirectory
```

Creates, if necessary, and returns the `NSHelpPanel` object. If the panel is created, it loads the help directory specified by `helpDirectory`. The help directory must reside in the main bundle. If a Help panel already exists but has loaded a help directory other than `helpDirectory`, a second panel will be created.

Instance Methods

addSupplement:inPath:

```
- (void)addSupplement:(NSString *)helpDirectory
    inPath:(NSString *)supplementPath
```

Append additional help entries to the Help panel's table of contents.

helpDirectory

```
- (NSString *)helpDirectory
```

Returns the absolute path of the help directory.

helpFile

```
- (NSString *)helpFile
```

Returns the path of the currently loaded help file.

print:

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

Prints the currently displayed help text.

showFile:atMarker:

```
- (void)showFile:(NSString *)filename atMarker:(NSString
*)markerName
```

Causes the panel to display the help contained in `filename` at `markerName`.

showHelpAttachedTo:

```
- (BOOL)showHelpAttachedTo:(id)anObject
```

Causes the panel to display help attached to `anObject`.

NSImage

Inherits From:	NSObject
Conforms To:	NSCoding, NSCopying NSObject (NSObject)
Declared In:	AppKit/NSImage.h

Class Description

An `NSImage` object contains an image that can be composited anywhere without first being drawn in any particular view. It manages the image by:

- Reading image data from the application bundle, from an `NSPasteboard`, or from an `NSData` object
- Keeping multiple representations of the same image

- Choosing the representation that's appropriate for a particular data type
- Choosing the representation that's appropriate for any given display device
- Caching the representations it uses by rendering them in off-screen windows
- Optionally retaining the data used to draw the representations, so that they can be reproduced when needed
- Compositing the image from the off-screen cache to where it's needed on-screen
- Reproducing the image for the printer so that it matches what's displayed on-screen, yet is the best representation possible for the printed page
- Automatically using any filtering services installed by the user to convert image data from unsupported formats to supported formats

Defining an Image

An image can be created from various types of data:

- Encapsulated PostScript code (EPS)
- Bitmap data in Tag Image File Format (TIFF)
- Untagged (raw) bitmap data
- Other image data supported by an `NSImageRep` subclass registered with the `NSImage` class
- Data that can be filtered to a supported type by a user-installed filter service

If data is placed in a file (for example, in an application bundle), the `NSImage` object can access the data whenever it's needed to create the image. If data is read from an `NSData` object, the `NSImage` object may need to store the data itself.

Images can also be defined by the program, in two ways:

- By drawing the image in an off-screen window maintained by the `NSImage` object. In this case, the `NSImage` maintains only the cached image.
- By defining a method that can be used to draw the image when needed. This allows the `NSImage` to delegate responsibility for producing the image to some other object.

Image Representations

An `NSImage` object can keep more than one representation of an image. Multiple representations permit the image to be customized for the display device. For example, different hand-tuned TIFF images can be provided for monochrome and color screens, and an EPS representation or a custom method might be used for printing. All representations are versions of the same image.

An `NSImage` returns an `NSArray` of its representations in response to a `representations` message. Each representation is a kind of `NSImageRep` object:

Table 1-13 Subclasses of NSImageRep

Subclass	What It Represents
NSEPSImageRep	An image that can be recreated from EPS data that's either stored by the object or at a known location in the file system.
NSBitmapImageRep	An image that can be recreated from bitmap or TIFF data.
NSCustomImageRep	An image that can be redrawn by a method defined in the application.
NSCachedImageRep	An image that has been rendered in an off-screen cache from data or instructions that are no longer available. The image in the cache provides the only data from which the image can be reproduced. Cached image copying is done by PostScript compositing. Unless it is copied, the cached image representation's backing store is the actual window it was initialized from. Any changes to this window will be reflected in the cached image representation.

You can define other `NSImageRep` subclasses for objects that render images from other types of source data. To make these new subclasses available to an `NSImage` object, they need to be added to the `NSImageRep` class registry by invoking the `registerImageRepClass:` class method. `NSImage` determines the data types that each subclass can support by invoking its `imageUnfilteredFileTypes` and `imageUnfilteredPasteboardTypes` methods.

Choosing Representations

The NSImage object will choose the representation that best matches the rendering device. By default, the choice is made according to the following set of ordered rules. Each rule is applied in turn until the choice of representation is narrowed to one.

1. Choose a color representation for a color device, and a gray-scale representation for a monochrome device.

2. Choose a representation with a resolution that matches the resolution of the device, or if no representation matches, choose the one with the highest resolution. By default, any image representation with a resolution that's an integer multiple of the device resolution is considered to match. If more than one representation matches, the NSImage will choose the one that's closest to the device resolution. However, you can force resolution matches to be exact by passing NO to the setMatchesOnMultipleResolution: method. Rule 2 prefers TIFF and bitmap representations, which have a defined resolution, over EPS representations, which don't. However, you can use the setUsesEPSOnResolutionMismatch: method to have the NSImage choose an EPS representation in case a resolution match isn't possible.

3. If all else fails, choose the representation with a specified bits per sample that matches the depth of the device. If no representation matches, choose the one with the highest bits per sample.

By passing NO to the setPrefersColorMatch: method, you can have the NSImage try for a resolution match before a color match. This essentially inverts the first and second rules above.

If these rules fail to narrow the choice to a single representation—for example, if the NSImage has two color TIFF representations with the same resolution and depth—the one that will be chosen is system dependent.

Caching Representations

When first asked to composite the image, the NSImage object chooses the representation that's best for the destination display device, as outlined above. It renders the representation in an off-screen window on the same device, then composites it from this cache to the desired location. Subsequent requests to composite the image use the same cache. Representations aren't cached until

they're needed for compositing. When printing, the NSImage tries not to use the cached image. Instead, it attempts to render on the printer the best version of the image that it can, using the appropriate image data, or a delegated method. Only as a last resort will it image the cached bitmap.

Image Size

Before an NSImage can be used, the size of the image must be set, in units of the base coordinate system. If a representation is smaller or larger than the specified size, it can be scaled to fit. If the size of the image hasn't already been set when the NSImage is provided with a representation, the size will be set from the data. The bounding box is used to determine the size of an NSEPSImageRep. The TIFF fields "ImageLength" and "ImageWidth" are used to determine the size of an NSBitmapImageRep.

Coordinate Systems

Images have the horizontal and vertical orientation of the base coordinate system; they can't be rotated or flipped. When composited, an image maintains this orientation, no matter what coordinate system it's composited to. (The destination coordinate system is used only to determine the location of a composited image, not its size or orientation.) It's possible to refer to portions of an image when compositing by specifying a rectangle in the image's coordinate system, which is identical to the base coordinate system, except that the origin is at the lower left corner of the image.

Named Images

An NSImage object can be identified either by its id or by a name. Assigning an NSImage a name adds it to a table kept by the class object; each name in the database identifies one and only one instance of the class. When you ask for an NSImage object by name (with the imageNamed: method), the class object returns the one from its database, which also includes all the system bitmaps provided by the Application Kit. If there's no object in the database for the specified name, the class object tries to create one by checking for a system bitmap of the same name, checking the name of the application's own image, and then checking for the image in the application's main bundle.

If a section or file matches the name, an `NSImage` is created from the data stored there. You can create `NSImage` objects simply by including EPS or TIFF data for them within the executable file, or in files inside the application's file package.

Image Filtering Services

`NSImage` is designed to automatically take advantage of user-installed filter services for converting unsupported image file types to supported image file types. The class method `imageFileTypes` returns an array of all file types from which `NSImage` can create an instance of itself. This list includes all file types supported by registered subclasses of `NSImageRep`, and those types that can be converted to supported file types through a user-installed filter service.

Method Types

Activity	Class Method
Initializing a new NSImage instance	– initByReferencingFile: – initWithContentsOfFile: – initWithData: – initWithPasteboard: – initWithSize:
Setting the size of the image	– setSize: – size
Referring to images by name	+ imageNamed: – setName: – name
Specifying the image	– addRepresentation: – addRepresentations: – lockFocus – lockFocusOnRepresentation: – unlockFocus
Using the image	– compositeToPoint:operation: – compositeToPoint:fromRect:operation: – dissolveToPoint:fraction: – dissolveToPoint:fromRect:fraction:
Choosing which image representation to use	– setPrefersColorMatch: – prefersColorMatch – setUsesEPSOnResolutionMismatch: – usesEPSOnResolutionMismatch – setMatchesOnMultipleResolution: – matchesOnMultipleResolution
Getting the representations	– bestRepresentationForDevice: – representations – removeRepresentation:
Determining how the image is stored	– setCachedSeparately: – isCachedSeparately – setDataRetained: – isDataRetained – setCacheDepthMatchesImageDepth: – cacheDepthMatchesImageDepth

Activity	Class Method
Determining how the image is drawn	– isFlipped – setFlipped: – isValid – setScalesWhenResized: – scalesWhenResized – backgroundColor – setBackgroundColor: – drawRepresentation:inRect: – recache
Assigning a delegate	– setDelegate: – delegate
Producing TIFF data for the image	– TIFFRepresentation – TIFFRepresentationUsingCompression:factor:
Managing NSImageRep subclasses	+ imageUnfilteredFileTypes + imageUnfilteredPasteboardTypes
Testing image data sources	+ canInitWithPasteboard: + imageFileTypes + imagePasteboardTypes
Methods Implemented by the Delegate	– imageDidNotDraw:inRect:

Class Methods

canInitWithPasteboard:

```
+ (BOOL)canInitWithPasteboard:(NSPasteboard *)pasteboard
```

Tests if the receiver can create a representation from the `pasteboard`. Returns `YES` if the `NSImage` class object can create an `NSImage` instance from the data represented by `pasteboard`. Returns `YES` if `NSImage`'s list of registered `NSImageRep`s includes a class that can handle the data represented by `pasteboard`. By default, this method returns `YES` if `pasteboard`'s type is `NSTIFFPboardType`, `NSPostScriptPboardType`, or `NSFilenamesPboardType` (for file names with extension `.tiff`, `.tif`, or `.eps`).

`NSImage` uses the `NSImageRep` class method `imageUnfilteredPasteboardTypes` to find the class that can handle the data in `pasteboard`. When creating a subclass of `NSImageRep` that accepts

image data from a nondefault pasteboard type, override the `imageUnfilteredPasteboardTypes` method to notify `NSImage` of the pasteboard types your class supports.

imageFileTypes

```
+ (NSArray *)imageFileTypes
```

Returns a null-terminated array of `NSString`s representing file types for which a registered `NSImageRep` exists. This list includes all file types supported by registered subclasses of `NSImageRep`, and those types that can be converted to supported file types through a user-installed filter service. The array returned by this method may be passed directly to the `NSOpenPanel`'s `runModalForTypes:` method. The returned array belongs to the system, and should not be freed by the application.

File types are identified by extension. By default, the list returned by this method contains "tiff", "tif", "eps". When creating a subclass of `NSImageRep` that accepts image data from non-default file types, override the `imageUnfilteredFileTypes` method to notify `NSImage` of the file types your class supports.

imageNamed:

```
+ (id)imageNamed:(NSString *)name
```

Creates and returns the `NSImage` object having `name` if found. Returns `nil` if an image with the given `name` isn't found. Searches the main bundle for the image if necessary. See also `NSString`.

imagePasteboardTypes

```
+ (NSArray *)imagePasteboardTypes
```

Returns an array of supported pasteboard types for which a registered `NSImageRep` exists. This list includes all pasteboard types supported by registered subclasses of `NSImageRep`, and those that can be converted to supported pasteboard types through a user-installed filter service. By default, the list returned by this method contains `NSPostScriptPboardType`, and `NSTIFFPboardType`. See also `NSImageRep`.

imageUnfilteredFileTypes

```
+ (NSArray *)imageUnfilteredFileTypes
```

Returns an array of file types recognized by the NSImage without filtering. This list comes from all registered NSImageReps. This array should not be freed or changed. See also NSArray.

imageUnfilteredPasteboardTypes

```
+ (NSArray *)imageUnfilteredPasteboardTypes
```

Returns an array of pasteboard types recognized by the NSImage. This array should not be freed or changed.

Instance Methods

addRepresentation:

```
- (void)addRepresentation:(NSImageRep *)imageRep
```

Adds imageRep to the receiver's list of representations. The image-representation object is retained by the image object. See also NSImageRep.

addRepresentations:

```
- (void)addRepresentations:(NSArray *)imageRepArray
```

Adds the image representations from imageRepArray to the receiver's list of representations. The image-representation objects are retained by the image object. See NSImageRep.

backgroundColor

```
- (NSColor *)backgroundColor
```

Returns the background color of the image. Returns the background color of the rectangle where the image is cached. If no background color has been specified, NS_COLORCLEAR is returned, indicating a totally transparent background. The background color will be visible when the image is composited only if the image doesn't completely cover all the pixels within the area specified for its size. See NSColor.

bestRepresentationForDevice:

```
- (NSImageRep *)bestRepresentationForDevice:
    (NSDictionary *)deviceDescription
```

Returns the best representation for the device described by `deviceDescription`. If `deviceDescription` is `nil`, the current device is assumed. See `NSGraphics.h` for appropriate dictionary keys and values. Do not send this message before the application object is running, and a DPS context created. See also `removeRepresentation:`, `representations`.

cacheDepthMatchesImageDepth

```
- (BOOL)cacheDepthMatchesImageDepth
```

Returns `YES` if the default depth limit applies to cached representation (in an off-screen window). Otherwise returns `NO`.

compositeToPoint:operation:

```
- (void)compositeToPoint:(NSPoint)aPoint
    operation:(NSCompositingOperation)op
```

Composites the image to the location specified by `aPoint`. `op` names the type of compositing operation requested, which must be one of of the following constants:

- `NSCompositeClear`
- `NSCompositeCopy`
- `NSCompositeSourceOver`
- `NSCompositeSourceIn`
- `NSCompositeSourceOut`
- `NSCompositeSourceAtop`
- `NSCompositeDestinationOver`
- `NSCompositeDestinationIn`
- `NSCompositeDestinationOut`
- `NSCompositeDestinationAtop`
- `NSCompositeXOR`
- `NSCompositePlusDarker`
- `NSCompositePlusLighter`

Note – The NSCompositeHighlight constant should not be used with this method.

aPoint is specified in the current coordinate system—the coordinate system of the currently focused view—and designates where the lower left corner of the image will appear. The image will have the orientation of the base coordinate system, regardless of the destination coordinates. The image is composited from its off-screen window cache. Since the cache isn't created until the image representation is first used, this method may need to render the image before compositing.

When printing, the compositing methods do not composite, but attempt to render the same image on the page that compositing would render on the screen, choosing the best available representation for the printer. The op argument is ignored. See also compositeToPoint:operation:, compositeToPoint:fromRect:operation:, and dissolveToPoint:fraction:.

compositeToPoint:fromRect:operation:

```
- (void)compositeToPoint:(NSPoint)aPoint fromRect:(NSRect)aRect
    operation:(NSCompositingOperation)op
```

Composites the aRect portion of the image to aPoint using the operation op, in the current coordinate system. The source rectangle is specified relative to a coordinate system that has its origin at the lower left corner of the image, but is otherwise the same as the base coordinate system. This method doesn't check to be sure that the rectangle encloses only portions of the image. Therefore it can conceivably composite areas that don't properly belong to the image, if the aRect rectangle happens to include them. If this turns out to be a problem, you can prevent it from happening by having the NSImage cache its representations in their own individual windows (with the setCachedSeparately: method). In this case, the window's clipping path will prevent anything but the image from being composited.

Compositing part of an image is as efficient as compositing the whole image, but printing part of an image is not. When printing, it's necessary to draw the whole image and rely on a clipping path to be sure that only the desired portion appears. See also compositeToPoint:operation:,

dissolveToPoint:fraction:, dissolveToPoint:fromRect:fraction:, and NSCompositingOperation.

delegate

```
- (id)delegate
```

Returns the NSImage delegate, or nil if no delegate is set.

dissolveToPoint:fraction:

```
- (void)dissolveToPoint:(NSPoint)aPoint fraction:(float)aFloat
```

Composites the image to the location specified by aPoint, just as composite:toPoint: does, but uses the dissolve operator rather than composite. aFloat is a fraction between 0.0 and 1.0 that specifies how much of the resulting composite will come from the NSImage.

To slowly dissolve one image into another, this method (or dissolveToPoint:fromRect:fraction:) needs to be invoked repeatedly with an ever-increasing aFloat value. Since aFloat refers to the fraction of the source image that's combined with the original destination, not the destination image after some of the source has been dissolved into it, the destination image should be replaced with the original destination before each invocation. This is best done in a buffered window before the results of the composite are flushed to the screen.

When printing, this method is identical to compositeToPoint:operation:, and aFloat is ignored. See also dissolveToPoint:fromRect:fraction:, compositeToPoint:operation:.

dissolveToPoint:fromRect:fraction:

```
- (void)dissolveToPoint:(NSPoint)aPoint fromRect:(NSRect)aRect
    fraction:(float)aFloat
```

Composites the aRect portion of the image to aPoint using the dissolve operator. aFloat is a value from 0.0 to 1.0 that determines how much of the resulting composite comes from the NSImage. When printing, this method is identical to compositeToPoint:fromRect:fraction:, and aFloat is ignored. See also dissolveToPoint:fraction:.

drawRepresentation:inRect:

```
- (BOOL)drawRepresentation:(NSImageRep *)imageRep
    inRect:(NSRect)aRect
```

Fills the specified rectangle with the background color, then sends the `imageRep` a `drawInRect:` (`NSImageRep`) message to draw itself inside the rectangle (if the `NSImage` is scalable), or a `drawAtPoint:` (`NSImageRep`) message to draw itself at the location of the rectangle (if the `NSImage` is not scalable). The rectangle is located in the current window and is specified in the current coordinate system.

This method shouldn't be called directly; the `NSImage` uses it to cache and print its representations. By overriding it in a subclass, you can change how representations appear in the cache, and thus how they'll appear when composited. For example, your version of the method could scale or rotate the coordinate system, then send a message to `super` to perform this version.

This method returns the value returned by the `drawInRect:` or `drawAtPoint:` (`NSImageRep`) method, which indicates whether or not the representation was successfully drawn. When `NO` is returned, the `NSImage` will ask another representation, if there is one, to draw the image. If the background color is fully transparent and the image is not being cached by the `NSImage`, the rectangle won't be filled before the representation draws. See also `NSImageRep`.

initByReferencingFile:

```
- (id)initByReferencingFile:(NSString *)filename
```

Initializes the new `NSImage` from the data in filename. The file is assumed to persist and may be reread later if the `NSImage` is resized or otherwise modified. This method initializes lazily: the `NSImage` doesn't actually open filename or create an image representation from its data until an application attempts to composite or requests information about the `NSImage`.

`filename` may be a full or relative path name, and should include an extension that identifies the data type in the file. The mechanism that actually creates the image representation for `filename` will look for an `NSImageRep` subclass that handles that data type from among those registered with `NSImage`. By default, the files handled are those with the extensions `tiff`, `tif`, and `eps`.

After finishing the initialization, this method returns `self`. However, if the new instance can't be initialized, it is freed and `nil` is returned. Since this method doesn't actually create an image representation for the data, your application should do error checking before attempting to use the image; one way to do so is by invoking the `lockFocus` method to check whether the image can be drawn.

initWithContentsOfFile:

```
- (id)initWithContentsOfFile:(NSString *)filename
```

Creates and registers an `NSImageRep` object for the receiver. Initializes the new `NSImageRep` object with the contents of filename, and with a size of 0. Returns `self` if successful, or `nil` otherwise.

initWithData:

```
- (id)initWithData:(NSData *)data
```

Creates and registers an `NSImageRep` object for the receiver. Initializes the new `NSImageRep` from `data`, and with a size of 0. Returns `self` if successful, or `nil` otherwise. See slso `NSImageRep`, `NSData`.

initWithPasteboard:

```
- (id)initWithPasteboard:(NSPasteboard *)pasteboard
```

Initializes the new `NSImage` with the data in `pasteboard`. `pasteboard` should be of a type returned by one of the registered `NSImageRep`'s `imageUnfilteredPasteboardTypes` methods; the default types supported are `NSPostscriptPboardType` (`NSEPSImageRep`) and `NSTIFFPboardType` (`NSBitmapImageRep`). If `pasteboard` is an `NSFilenamesPboardType`, the file name should have an extension returned by one of the registered `NSImageRep`'s `imageUnfilteredFileTypes` methods; the default types supported are `TIFF`, `TIF`, (`NSBitmapImageRep`) and `EPS` (`NSEPSImageRep`).

initWithSize:

```
- (id)initWithSize:(NSSize)aSize
```

Initializes the receiver, a newly allocated NSImage instance, to the size specified and returns self. The size should be specified in units of the base coordinate system. It must be set before the NSImage can be used. This method is the designated initializer for the class (the method that incorporates the initialization of classes higher in the hierarchy through a message to super). All other initialization methods defined in this class work through this method. See also NSSize, size, and setSize:.

isCachedSeparately

```
- (BOOL)isCachedSeparately
```

Returns YES if each representation of the image is cached alone in an off-screen window of its own, and NO if they can be cached in off-screen windows together with other images. A return of NO doesn't mean that the windows are, in fact, shared, just that they can be. The default is NO. See also setCachedSeparately:, compositeToPoint:fromRect:operation:.

isDataRetained

```
- (BOOL)isDataRetained
```

Returns YES if the image data is retained after the image is cached. Returns NO otherwise. The default is NO. If the data is available in a section of the application executable or in a file that won't be moved or deleted, or if responsibility for drawing the image is delegated to another object with a custom method, there's no reason for the NSImage to retain the data. However, if the NSImage reads image data from a stream, you may want to have it keep the data itself; for example, to render the same image on another device at a different resolution. See also setDataRetained:.

isFlipped

```
- (BOOL)isFlipped
```

Returns YES if a flipped coordinate system is used when locating the image, and NO if it isn't. The default is NO. See also setFlipped:.

isValid

```
- (BOOL)isValid
```

Returns YES to indicate that the receiver's image is valid, otherwise returns NO. An image can be invalid if the file from which it was initialized is non-existent, or the data in that file is invalid. The following code shows how to use `isValid`.

```
if ([myImage isValid]) {
    [myImage lockFocus];

    // ...

    [myImage unlockFocus];
> }
```

See also `lockFocus`.

lockFocus

```
- (void)lockFocus
```

Focuses on the best representation for the `NSImage` by

- Making the off-screen window where the representation will be cached the current window
- Making a coordinate system specific to the area where the image will be drawn the current coordinate system.

Use this method in preparation for drawing. The best representation is the one that best matches the deepest available frame buffer; it's the same object returned by the `bestRepresentationForDevice:` method. If the `NSImage` has no representations, `lockFocus` creates one with default depth.

A `lockFocus` message should first check for a valid image, and must be balanced by a subsequent `unlockFocus` message to the same receiver. For example:

```
if ([myImage isValid]) {
    [myImage lockFocus];

    // ...

    [myImage unlockFocus];
> }
```

See also `lockFocusOnRepresentation:`, `unlockFocus`, `bestRepresentationForDevice:`, `isValid`.

lockFocusOnRepresentation:

```
- (void)lockFocusOnRepresentation:(NSImageRep *)imageRep
```

Focuses on the `imageRep` representation by making the off-screen window where `imageRep` will be cached the current window, and by making a coordinate system specific to the area where the image will be drawn the current coordinate system. Use this method in preparation for drawing. A `lockFocusOnRepresentation:` message should first check for a valid image, and must be balanced by a subsequent `unlockFocus` message to the same receiver. For example:

```
if ([myImage isValid]) {
    [myImage lockFocusOnRepresentation:myRep];

    // ...

    [myImage unlockFocus];
> }
```

See also `lockFocus`, `unlockFocus`, `isValid`.

matchesOnMultipleResolution

```
- (BOOL)matchesOnMultipleResolution
```

Returns `YES` if the resolution of the device and the resolution specified for the image are considered to match if one is a multiple of the other, and `NO` if device and image resolutions are considered to match only if they are exactly the same. The default is `YES`. See also `setMatchesOnMultipleResolution:`, `setUsesEPSOnResolutionMismatch:`, `usesEPSOnResolutionMismatch`.

name

```
- (NSString *)name
```

Returns the name assigned to the `NSImage`, or `nil` if no name has been assigned See also `setName:`, `imageNamed:`.

prefersColorMatch

```
- (BOOL)prefersColorMatch
```

Returns YES if, when selecting the representation it will use, the NSImage first looks for a representation that matches the color capability of the rendering device; that is, choosing a gray-scale representation for a monochrome device and a color representation for a color device. Then if necessary the NSImage narrows the selection by looking for one that matches the resolution of the device. If NO is returned, the NSImage first looks for a representation that matches the resolution of the device, then tries to match the representation to the color capability of the device. The default is YES. See also setPrefersColorMatch:.

recache

```
- (void)recache
```

Invalidates the off-screen caches of all representations and frees them. The next time any representation is composited, it will first be asked to redraw itself in the cache. Cached image representations aren't destroyed by this method. If an image is not likely to be used again, it is a good idea to free its caches, since that will reduce that amount of memory consumed by your program and therefore improve performance.

removeRepresentation:

```
- (void)removeRepresentation:(NSImageRep *)imageRep
```

Frees imageRep after removing it from the NSImage's list of representations. See also representations. See also representations, bestRepresentationForDevice:.

representations

```
- (NSArray *)representations
```

Returns an array of all the image representations. The array belongs to the NSImage object, and there's no guarantee that the same array will be returned each time. Therefore, rather than saving the array that is returned, you should ask for it each time you need it. See also bestRepresentationForDevice:, removeRepresentation:.

scalesWhenResized

```
- (BOOL)scalesWhenResized
```

Returns YES if image representations are scaled to fit the size specified for the NSImage. If representations are not scalable, this method returns NO. The default is NO. Representations created from data that specifies a size (for example, the "ImageLength" and "ImageWidth" fields of a TIFF representation or the bounding box of an EPS representation) will have the size the data specifies, which may differ from the size of the NSImage. See also setScalesWhenResized:.

setBackgroundColor:

```
- (void)setBackgroundColor:(NSColor *)aColor
```

Sets the background color of the image to aColor. The default is a transparent background. The background color will be visible only for representations that don't completely cover all the pixels within the image when drawing. The background color is ignored for cached image representations; such caches are always created with a white background. This method doesn't cause the receiving NSImage to recache itself.

setCacheDepthMatchesImageDepth:

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

Determines whether the depth of the off-screen windows, where the NSImage's representations are cached, should be limited by the application's default depth limit. If flag is NO, window depths will be determined by the specifications of the representations, rather than by the current display devices. The default is YES. This method doesn't cause the receiving NSImage to recache itself.

setCachedSeparately:

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

Sets whether each image representation will be cached in its own off-screen window or in a window shared with other images. If flag is YES, each representation is guaranteed to be in a separate window. If flag is NO, a representation can be cached together with other images, though in practice it

might not be. The default is NO. If an NSImage is to be resized frequently, it's more efficient to cache its representations in unique windows. This method does not invalidate any existing caches. See also isCachedSeparately.

setDataRetained:

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

Determines whether the NSImage retains the data needed to render the image. The default is NO. If the data is available in a section of the application executable or in a file that won't be moved or deleted, or if responsibility for drawing the image is delegated to another object with a custom method, there's no reason for the NSImage to retain the data. However, if the NSImage reads image data from a stream, you may want to have it keep the data itself. Generally, this is useful to redraw the image to a device of different resolution. If an image representation is created lazily (through the initByReferencingFile: method, for example), the only data retained is the source name. See also isDataRetained.

setDelegate:

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

Makes anObject the delegate of the NSImage. See also delegate.

setFlipped:

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

Determines whether the polarity of the y-axis is inverted when drawing an image. If flag is YES, the image will have its coordinate origin in the upper left corner and the positive y-axis will extend downward. This method affects only the coordinate system used to draw the image, and doesn't cause the receiving NSImage to recache itself. See also isFlipped.

setMatchesOnMultipleResolution:

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

Determines whether image representations with resolutions that are exact multiples of the resolution of the device are considered to match the device. The default is YES. See also matchesOnMultipleResolution.

setName:

```
- (BOOL)setName:(NSString *)name
```

Assigns `name` to be the receiver's name, and registers the image under that name. Returns `NO` if `name` is already in use; otherwise, returns `YES`. See also `name`.

setPrefersColorMatch:

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

Determines how the `NSImage` will select which representation to use. If `flag` is `YES`, it first tries to match the representation to the color capability of the rendering device, choosing a color representation for a color device and a gray-scale representation for a monochrome device. Then if necessary the `NSImage` narrows the selection by trying to match the resolution of the representation to the resolution of the device. If `flag` is `NO`, the `NSImage` first tries to match the representation to the resolution of the device, and then tries to match it to the color capability of the device. The default is `YES`. See also `prefersColorMatch`.

setScalesWhenResized:

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

Determines whether representations with sizes that differ from the size of the `NSImage` will be scaled to fit. The default is `NO`. Generally, representations that are created through `NSImage` methods have the same size as the `NSImage`. However, a representation that's added with the `addRepresentation:` method may have a different size, and representations created from data that specifies a size (for example, the "ImageLength" and "ImageWidth" fields of a TIFF representation or the bounding box of an EPS representation) will have the size specified. This method doesn't cause the receiving `NSImage` to recache itself when it is next composited. See also `scalesWhenResized`.

setSize:

```
- (void)setSize:(NSSize)aSize
```

Sets the width and height of the image in base coordinates. The `NSImage` size must be set before it can be used. The `NSImage` size can be changed after it has been used, but changing it invalidates all its caches and frees them. When the image is next composited, the selected representation will draw itself in an off-screen window to recreate the cache. See also `size`.

setUsesEPSOnResolutionMismatch:

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

Determines whether EPS representations will be preferred when there are no image representations that match the resolution of the device. The default is `NO`. See also `usesEPSOnResolutionMismatch`.

size

```
- (NSSize)size
```

Returns the image size. If no size has been set, all values in the structure will be set to 0.0. See also `setSize:`.

TIFFRepresentation

```
- (NSData *)TIFFRepresentation
```

Returns a data object containing TIFF data for imagel representations, using their default compressions. See also `TIFFRepresentationUsingCompression:factor:`.

TIFFRepresentationUsingCompression:factor:

```
- (NSData *)TIFFRepresentationUsingCompression:
    (NSTIFFCompression)comp factor:(float)aFloat
```

Returns a data object containing TIFF data for all image representations. The compression arguments `comp` and `aFloat` specify the type of compression and the compression amount (factor). The compression factor provides a hint for those compression types that implement variable compression ratios; currently only JPEG compression uses the compression factor argument.

unlockFocus

```
- (void)unlockFocus
```

Balances a previous `lockFocus` or `lockFocusOnRepresentation:` message. All successful `lockFocus` and `lockFocusOnRepresentation:` messages must be followed by a subsequent `unlockFocus` message. Those that return `NO` should never be followed by `unlockFocus`.

usesEPSOnResolutionMismatch

```
- (BOOL)usesEPSOnResolutionMismatch
```

Returns `YES` if an EPS representation of the image should be used whenever it's impossible to match the resolution of the device to the resolution of another representation of the image (a TIFF representation, for example). By default, this method returns `NO` to indicate that EPS representations are not necessarily preferred. See also `setUsesEPSOnResolutionMismatch:`.

Methods Implemented by the Delegate

imageDidNotDraw:inRect:

```
- (NSImage *)imageDidNotDraw:(id)sender inRect:(NSRect)aRect
```

This method should be implemented in the delegate, and should respond to a message that `sender` couldn't be composited into `aRect`. If an instance of `NSImage` cannot composite or dissolve after having tried all its representations, it will call its error handler which will try to send a `imageDidNotDraw:inRect:` to the delegate. If there's no delegate or it doesn't understand this method, then the `NSImage` will clear the destination area with the background color. Otherwise, the delegate is assumed to return an instance of `NSImage` to which the same dissolve or composite message will be sent. The delegate can also return `nil`, in which case it is assumed to have taken care of things. See also `delegate`, `setDelegate:`.

1

NSImageRep

Inherits From:	NSObject
Conforms To:	NSCoding, NSCopying NSObject (NSObject)
Declared In:	AppKit/NSImageRep.h

Class Description

`NSImageRep` is an abstract superclass; each of its subclasses knows how to draw an image from a particular kind of source data. While an `NSImageRep` subclass can be used directly, it's typically used through an `NSImage` object. An `NSImage` manages a group of representations, choosing the best one for the current output device.

There are four subclasses defined in the Application Kit:

Table 1-14 NSImageRep Subclasses

Subclass	Source Data
NSBitmapImageRep	Tag Image File Format (TIFF) and other bitmap data
NSEPSImageRep	Encapsulated PostScript (EPS) code
NSCustomImageRep	A delegated method that can draw the image
NSCachedImageRep	A rendered image, usually in an off-screen window

You can define other `NSImageRep` subclasses for objects that render images from other types of source information. New subclasses must be added to the `NSImageRep` class registry by invoking the `registerImageRepClass:` class method. The `NSImageRep` subclass informs the registry of the data types it can support through its `imageUnfilteredFileTypes`, `imageUnfilteredPasteboardTypes`, and `canInitWithData:` class methods. Once an `NSImageRep` subclass is registered, an instance of that subclass is created any time `NSImage` encounters the type of data handled by that subclass.

Method Types

Activity	Class Method
Creating an NSImageRep	+ imageRepWithContentsOfFile: + imageRepsWithContentsOfFile: + imageRepWithPasteboard: + imageRepsWithPasteboard:
Checking data types	+ canInitWithData: + canInitWithPasteboard: + imageFileTypes + imagePasteboardTypes + imageUnfilteredFileTypes + imageUnfilteredPasteboardTypes
Setting the size of the image	– setSize: – size
Specifying information about the representation	– bitsPerSample – colorSpaceName – hasAlpha – isOpaque – pixelsHigh – pixelsWide – setAlpha: – setBitsPerSample: – setColorSpaceName: – setOpaque: – setPixelsHigh: – setPixelsWide:
Drawing the image	– draw – drawAtPoint: – drawInRect:
Managing NSImageRep subclasses	+ imageRepClassForData: + imageRepClassForFileType: + imageRepClassForPasteboardType: + registerImageRepClass: + registeredImageRepClasses + unregisterImageRepClass:

Class Methods

canInitWithData:

```
+ (BOOL)canInitWithData:(NSData *)data
```

Overridden in subclasses to return YES if the receiver can initialize itself from data. The default implementation returns NO. See also canInitWithPasteboard:, imageFileTypes, imagePasteboardTypes, imageUnfilteredFileTypes, imageUnfilteredPasteboardTypes.

canInitWithPasteboard:

```
+ (BOOL)canInitWithPasteboard:(NSPasteboard *)pasteboard
```

Returns YES if NSImageRep can handle the data represented by pasteboard. This method invokes the imageUnfilteredPasteboardTypes and imageFileTypes class methods to return arrays of directly supported pasteboard types and files, and checks these against the data in pasteboard. When creating a subclass of NSImageRep that accepts image data from a nondefault pasteboard type, you override the imageUnfilteredPasteboardTypes and imageFileTypes methods to assure that this method returns the correct response.

imageFileTypes

```
+ (NSArray *)imageFileTypes
```

Returns an array of strings representing all file types supported by NSImageRep. The list includes both those types returned by the NSGetFileTypes() function and those types that can be converted to a supported type by a user-installed filter service. Don't override this method when subclassing NSImageRep. It always returns a valid list for a subclass of NSImageRep. See also NSGetFileTypes() (in the Application Kit's "Functions" chapter).

imagePasteboardTypes

```
+ (NSArray *)imagePasteboardTypes
```

Returns an array representing all pasteboard types supported by NSImageRep or one of its subclasses. The list includes both those types returned by the imageUnfilteredPasteboardTypes class method and those that can be converted by a user-installed filter service to a supported type. Don't override this method when subclassing NSImageRep; it always returns a valid list for a subclass of NSImageRep that correctly overrides the imageUnfilteredPasteboardTypes method. See also imageFileTypes.

imageRepClassForData:

```
+ (Class)imageRepClassForData:(NSData *)data
```

Returns the NSImageRep subclass that handles data of type data. See also registerImageRepClass:, registeredImageRepClasses, unregisterImageRepClass:, imageRepClassForFileType:, imageRepClassForPasteboardType:.

imageRepClassForFileType:

```
+ (Class)imageRepClassForFileType:(NSString *)type
```

Returns the NSImageRep subclass that handles data of file type type. See also imageRepClassForData:, registerImageRepClass:.

imageRepClassForPasteboardType:

```
+ (Class)imageRepClassForPasteboardType:(NSString *)type
```

Returns the NSImageRep subclass that handles data of pasteboard type type. See also imageRepClassForData:, registerImageRepClass:.

imageRepWithContentsOfFile:

```
+ (id)imageRepWithContentsOfFile:(NSString *)filename
```

In subclasses that respond to imageFileTypes and imageRepWithData: (for example NSBitmapImageRep), returns an object that has been initialized with the data in filename. NSImageRep's implementation returns an instance of the appropriate registered subclass. See also imageRepsWithContentsOfFile:, imageRepWithPasteboard:.

imageRepsWithContentsOfFile:

```
+ (NSArray *)imageRepsWithContentsOfFile:(NSString *)filename
```

In subclasses that respond to `imageFileTypes`, `imageRepWithData:`,and `imageRepWithData:` (for example, `NSBitmapImageRep`), returns an array of objects that have been initialized with the data in `filename`. `NSImageRep`'s implementation returns an array of objects (each an instance of the appropriate registered subclass) that have been initialized with the data in `filename`. See also `imageRepWithContentsOfFile:`.

imageRepWithPasteboard:

```
+ (id)imageRepWithPasteboard:(NSPasteboard *)pasteboard
```

In subclasses that respond to `imagePasteboardTypes` and `imageRepWithData:` (for example, `NSBitmapImageRep`), returns an object that has been initialized with the data in `pasteboard`. `NSImageRep`'s implementation returns an instance of the appropriate registered subclass. See also `imageRepsWithPasteboard:`, `imageRepWithContentsOfFile:`.

imageRepsWithPasteboard:

```
+ (NSArray *)imageRepsWithPasteboard:(NSPasteboard *)pasteboard
```

In subclasses that respond to `imagePasteboardTypes` and `imageRepsWithData:` (or `imageRepWithData:`), returns an array of objects that have been initialized with the data in `pasteboard`. `NSImageRep`'s implementation returns an array of objects (each an instance of the appropriate registered subclass) that have been initialized with the data in `pasteboard`. See also `imageRepWithPasteboard:`, `imageRepWithContentsOfFile:`.

imageUnfilteredFileTypes

```
+ (NSArray *)imageUnfilteredFileTypes
```

Returns an array of strings representing all file types (extensions) supported by the `NSImageRep`. By default, the returned array is emtpy. When creating an `NSImageRep` subclass, override this method to return a list of strings representing the file types supported. For example, `NSBitmapImageRep` implements the following code for this method:

```
+ (NSArray *)imageUnfilteredFileTypes {
    static NSArray *types = nil;
    if (!types)  types = [[NSArray allocWithZone:_NXAppZone()]
                          initWithObjects:@"tiff", @"tif", nil];
    return types;
}
```

If your subclass supports the types supported by its superclass, you must explicitly get the array of types from the superclass and put them in the array returned by this method. See also `imageUnfilteredPasteboardTypes`.

imageUnfilteredPasteboardTypes

```
+ (NSArray *)imageUnfilteredPasteboardTypes
```

Returns an array of strings representing directly supported pasteboard types. By default, the returned array is empty. When creating a subclass of `NSImageRep`, override this method to return a list representing the pasteboard types supported. For example, `NSBitmapImageRep` implements the following code for this method:

```
+ (NSArray *)imageUnfilteredPasteboardTypes {
    static NSArray *types = nil;
    if (!types) types = [[NSArray allocWithZone:_NXAppZone()]
                           initWithObjects:NSTIFFPboardType, nil];
    return types;
}
```

If your subclass supports the types supported by its superclass, you must explicitly get the list of types from the superclass and add them to the array returned by this method. See also `imageUnfilteredFileTypes`.

registerImageRepClass:

```
+ (void)registerImageRepClass:(Class)imageRepClass
```

Adds `imageRepClass` to the registry of available `NSImageRep` classes. This method posts the `NSImageRepRegistryDidChangeNotification` notification with the receiving object to the default notification center. See also `registeredImageRepClasses`, `unregisterImageRepClass:`, `imageRepClassForData:`.

registeredImageRepClasses

```
+ (NSArray *)registeredImageRepClasses
```

Returns an array containing the names of the registered NSImageRep classes. See also registerImageRepClass:.

unregisterImageRepClass:

```
+ (void)unregisterImageRepClass:(Class)imageRepClass
```

Removes imageRepClass from the registry of available NSImageRep classes. This method posts the NSImageRepRegistryDidChangeNotification notification with the receiving object to the default notification center. See also registerImageRepClass:.

Instance Methods

bitsPerSample

```
- (int)bitsPerSample
```

Returns the number of bits used to specify a single pixel in each component of the data. If the image isn't specified by pixel values, but is device-independent, the return value will be NSImageRepMatchesDevice. See also setBitsPerSample:.

colorSpaceName

```
- (NSString *)colorSpaceName
```

Returns the name of the image's color space. The default is NSCalibratedRGBColorSpace. See also NSColor, setColorSpaceName:.

draw

```
- (BOOL)draw
```

Implemented by subclasses to draw the image at location (0.0, 0.0) in the current coordinate system. Subclass methods return YES if the image is successfully drawn, and NO if it isn't. This version of the method simply returns YES. See also drawAtPoint:, drawInRect:.

drawAtPoint:

```
- (BOOL)drawAtPoint:(NSPoint)aPoint
```

Translates the current coordinate system to the location specified by `point` and has the receiver's `draw` method draw the image at that point.This method returns `NO` without translating or drawing if the size of the image has not been set. Otherwise, it returns the value returned by the `draw` method, which indicates whether the image is successfully drawn. The coordinate system is not restored after it has been translated. See also `draw`, `drawInRect:`.

drawInRect:

```
- (BOOL)drawInRect:(NSRect)aRect
```

Draws the image so that it fits inside the rectangle referred to by `aRect`. The current coordinate system is first translated to the point specified in the rectangle and is then scaled so the image will fit within the rectangle. The receiver's `draw` method is then invoked to draw the image. See also `draw`, `drawAtPoint:`.

hasAlpha

```
- (BOOL)hasAlpha
```

Returns `YES` if the receiver has been informed that the image has a coverage component (alpha), and `NO` if not. See also `setAlpha:`.

isOpaque

```
- (BOOL)isOpaque
```

Returns `YES` if the receiver is opaque; `NO` otherwise. Use this method to test whether an `NSImageRep` completely covers the area within the rectangle returned by `getSize:`. See also `setOpaque:`.

pixelsHigh

```
- (int)pixelsHigh
```

Returns the height of the image in pixels, as specified in the image data. If the image isn't specified by pixel values, but is device-independent, the return value will be `NSImageRepMatchesDevice`. See also `setPixelsHigh:`, `pixelsWide`, `bitsPerSample`.

pixelsWide

```
- (int)pixelsWide
```

Returns the width of the image in pixels, as specified in the image data. If the image isn't specified by pixel values, but is device-independent, the return value will be `NSImageRepMatchesDevice`. See also `setPixelsWide:`, `pixelsHigh`, `bitsPerSample`.

setAlpha:

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

Informs the `NSImageRep` whether the image has an alpha component. `flag` should be `YES` if it does, and `NO` if it doesn't. See also `hasAlpha`.

setBitsPerSample:

```
- (void)setBitsPerSample:(int)anInt
```

Informs the `NSImageRep` that the image has `anInt` bits of data for each pixel in each component. If the image isn't specified by pixel values, but is device-independent, `anInt` should be `NSImageRepMatchesDevice`. See also `bitsPerSample`.

setColorSpaceName:

```
- (void)setColorSpaceName:(NSString *)aString
```

Tells the receiver of the image's color space. See also `NSColor`, `colorSpaceName`.

setOpaque:

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

Tells the receiver of the image's opacity. See also `isOpaque`.

setPixelsHigh:

```
- (void)setPixelsHigh:(int)anInt
```

Informs the NSImageRep that the data specifies an image anInt pixels high. If the image isn't specified by pixel values, but is device-independent, anInt should be NSImageRepMatchesDevice. See also pixelsHigh, setPixelsWide:.

setPixelsWide:

```
- (void)setPixelsWide:(int)anInt
```

Informs the NSImageRep that the data specifies an image anInt pixels wide. If the image isn't specified by pixel values, but is device-independent, anInt should be NSImageRepMatchesDevice. See also pixelsWide, setPixelsHigh:.

setSize:

```
- (void)setSize:(NSSize)aSize
```

Sets the size of the image in units of the base coordinate system. This determines the size of the image when it's rendered; it's not necessarily the same as the width and height of the image in pixels as specified in the image data. See also size.

size

```
- (NSSize)size
```

Returns the size of the image. The size is provided in units of the base coordinate system. See also size.

1

NSMatrix

Inherits From:	NSControl : NSView : NSResponder : NSObject
Conforms To:	NSCoding (NSResponder) NSObject (NSObject)
Declared In:	AppKit/NSMatrix.h

Class Description

`NSMatrix` is a class used for creating groups of `NSCell`s that work together in various ways. `NSMatrix` includes methods for arranging `NSCell`s in rows and columns, either with or without space between them. `NSCell`s in an `NSMatrix` are numbered by row and column, each starting with 0; for example, the top left `NSCell` would be at (0, 0), and the `NSCell` that's second down and third across would be at (1, 2).

The cell objects that an `NSMatrix` contains are usually of a single subclass of NSCell, but they can be of multiple subclasses of `NSCell`. The only restriction is that all cell objects must be the same size. An `NSMatrix` can be set up to create new `NSCell`s by copying a prototype object, or by allocating and initializing instances of a specific `NSCell` class.

An `NSMatrix` adds to `NSControl`'s target/action paradigm by allowing a separate target and action for each of its `NSCell`s in addition to its own target and action. It also allows for an action message that's sent when the user double-clicks an `NSCell`, and which is sent in addition to the single-click action message. If an `NSCell` doesn't have an action, the `NSMatrix` sends its own action to its own target. If an `NSCell` doesn't have a target, the `NSMatrix` sends the `NSCell`'s action to its own target. The double-click action of an `NSMatrix` is always sent to the target of the `NSMatrix`.

Since the user might press the mouse button while the cursor is within the `NSMatrix` and then drag the mouse around, `NSMatrix` offers four "selection modes" that determine how `NSCell`s behave when the `NSMatrix` is tracking the mouse:

- `NSTrackModeMatrix` is the most basic mode of operation. In this mode the `NSCell`s are asked to track the mouse with `trackMouse:inRect:ofView:untilMouseUp:` whenever the mouse is

inside their bounds. No highlighting is performed. An example of this mode might be a “graphic equalizer” NSMatrix of sliders, where moving the mouse around causes the sliders to move under the mouse.

- NSHighlightModeMatrix is a modification of NSTrackModeMatrix. In this mode, an NSCell is highlighted before it’s asked to track the mouse, then unhighlighted when it’s done tracking. This is useful for multiple unconnected NSCells that use highlighting to inform the user that they are being tracked (like push-buttons and switches).
- NSRadioModeMatrix is used when you want no more than one NSCell to be selected at a time. It can be used to create a set of buttons of which one and only one is selected (there’s the option of allowing no button to be selected). Any time an NSCell is selected, the previously selected NSCell is deselected. The canonical example of this mode is a set of radio buttons.
- NSListModeMatrix is the opposite of NSTrackModeMatrix. NSCells are highlighted, but they don’t track the mouse. This mode can be used to select a range of text values, for example. NSMatrix supports the standard multiple-selection paradigms of dragging to select, using the shift key to make discontinuous selections, and using the alternate key to extend selections.

Method Types

Activity	Class Method
Initializing the NSMatrix class	+ cellClass + setCellClass:
Initializing an NSMatrix object	– initWithFrame: – initWithFrame:mode:cellClass:numberOfRows: numberOfColumns: – initWithFrame:mode:prototype:numberOfRows: numberOfColumns:
Setting the selection mode	– mode – setMode:
Configuring the NSMatrix	– allowsEmptySelection – isSelectionByRect – setAllowsEmptySelection: – setSelectionByRect:
Setting the cell class	– cellClass – prototype – setCellClass: – setPrototype:

Activity	Class Method
Laying out the NSMatrix	– addColumn – addColumnWithCells: – addRow – addRowWithCells: – cellFrameAtRow:column: – cellSize – getNumberOfRows:columns: – insertColumn: – insertColumn:withCells: – insertRow: – insertRow:withCells: – intercellSpacing – makeCellAtRow:column: – numberOfColumns – numberOfRows – putCell:atRow:column: – removeColumn: – removeRow: – renewRows:columns: – setCellSize: – setIntercellSpacing: – sortUsingFunction:context:
Finding matrix coordintes	– getRow:column:forPoint: – getRow:column:ofCell:
Modifying individual cells	– setState:atRow:column:
Selecting cells	– deselectAllCells – deselectSelectedCell – selectAll: – selectCellAtRow:column: – selectCellWithTag: – selectedCell – selectedCells – selectedColumn – selectedRow – setSelectionFrom:to:anchor:highlight:
Finding cells	– cellAtRow:column: – cellWithTag: – cells

Activity	Class Method
Modifying graphic attributes	– backgroundColor – cellBackgroundColor – drawsBackground – drawsCellBackground – setBackgroundColor: – setCellBackgroundColor: – setDrawsBackground: – setDrawsCellBackground:
Editing text in cells	– selectText: – selectTextAtRow:column: – textDidBeginEditing: – textDidChange: – textDidEndEditing: – textShouldBeginEditing: – textShouldEndEditing:
Setting tab key behavior	– nextText – previousText – setNextText: – setPreviousText:
Assigning a delegate	– delegate – setDelegate:
Resizing the matrix and cells	– autosizesCells – setAutosizesCells: – setValidateSize: – sizeToCells
Scrolling	– isAutoscroll – scrollCellToVisibleAtRow:column: – setAutoscroll: – setScrollable:
Displaying	– drawCellAtRow:column: – highlightCell:atRow:column:

Activity	Class Method
Target and action	– doubleAction – setDoubleAction: – errorAction – sendAction – sendAction:to:forAllCells: – sendDoubleAction – setErrorAction:
Handling event and action messages	– acceptsFirstMouse: – mouseDown: – mouseDownFlags – performKeyEquivalent:
Managing the cursor	– resetCursorRects

Class Methods

cellClass

```
+ (Class)cellClass
```

Returns the default class object used to fill the matrix cells. The default class is `NSActionCell`. See also `setCellClass:`.

setCellClass:

```
+ (void)setCellClass:(Class)classId
```

Sets the default class used to fill the matrix cells. `classId` should be the `id` of an `NSCell` subclass (usually `NSActionCell`), obtained by sending the `class` message to either the `NSCell` subclass object or to an instance of that subclass. The default `NSCell` class is `NSActionCell`. Your code should rarely need to invoke this method, since each instance of `NSMatrix` can be configured to use its own `NSCell` class (or a prototype that gets copied). The `NSCell` class set with this method is simply a fallback for matrices initialized with `initWithFrame:`. “Creating New `NSControl`s” in the `NSControl` class specification has more information on how to safely set the `NSCell` class used by a subclass of `NSControl`. See also `cellClass`.

Instance Methods

acceptsFirstMouse:

```
- (BOOL)acceptsFirstMouse:(NSEvent *)theEvent
```

Returns NO only if receiver's mode is NSListModeMatrix. Returns YES if the matrix is in any other selection mode. NSMatrix does not accept first mouse in NSListModeMatrix mode to prevent the loss of multiple selections. See the NSMatrix "Class Description" for more information on selection modes. See also mouseDown:, mouseDownFlags, performKeyEquivalent:.

addColumn

```
- (void)addColumn
```

Adds a new column of cells to the right of the last column, creating new NCell objects if needed (with makeCellAtRow:column:). See also addColumnWithCells:, addRow, addRowWithCells:, makeCellAtRow:column:.

addColumnWithCells:

```
- (void)addColumnWithCells:(NSArray *)cellArray
```

Adds a new matrix column (using makeCellAtRow:column:), using the cells contained in cellArray. See also addColumn, makeCellAtRow:column:.

addRow

```
- (void)addRow
```

Adds a new row of cells below the last row, creating new cells if needed with makeCellAtRow:column:. See also addColumn, makeCellAtRow:column:.

addRowWithCells:

```
- (void)addRowWithCells:(NSArray *)cellArray
```

Adds a new matrix row, using the cells contained in cellArray. See also addColumn, makeCellAtRow:column:.

allowsEmptySelection

```
- (BOOL)allowsEmptySelection
```

Returns YES if it is possible to have no cells selected in a radio-mode NSMatrix, NO otherwise. See also setAllowsEmptySelection:.

autosizesCells

```
- (BOOL)autosizesCells
```

Returns YES if cells are resized proportionally to the NSMatrix when its size changes; the intercell spacing is kept constant. Returns NO if the inter-cell spacing changes when the NSMatrix is resized; the cell size remains constant. See also setAutosizesCells:, sizeToCells.

backgroundColor

```
- (NSColor *)backgroundColor
```

Returns the color of the background between cells. See also setBackgroundColor:, cellBackgroundColor, drawsBackground, setDrawsBackground:, drawsCellBackground, setDrawsCellBackground:, setCellBackgroundColor:.

cellAtRow:column:

```
- (id)cellAtRow:(int)row column:(int)column
```

Returns the cell at the given row and column, or nil if no cell exists at that location. See also cellWithTag:, cells.

cellBackgroundColor

```
- (NSColor *)cellBackgroundColor
```

Returns the background color within the cells. See also setCellBackgroundColor:, backgroundColor.

cellClass

```
- (Class)cellClass
```

Returns the subclass of NSCell that is used by NSMatrix to make new cells. See also setCellClass:, setPrototype:, prototype.

cellFrameAtRow:column:

```
- (NSRect)cellFrameAtRow:(int)row column:(int)column
```

Returns the frame rectangle of the cell at row and column. See also cellSize, makeCellAtRow:column:.

cellSize

```
- (NSSize)cellSize
```

Returns the width and height of cells in the matrix; all cells are the same size. See also cellFrameAtRow:column:, makeCellAtRow:column:.

cellWithTag:

```
- (id)cellWithTag:(int)anInt
```

Returns the cell having anInt as its tag, or nil if none exists. See also cellAtRow:column:, cells.

cells

```
- (NSArray *)cells
```

Returns the matrix's array of cells. See also cellAtRow:column:, cellWithTag:.

delegate

```
- (id)delegate
```

Returns the delegate object that receives messages passed on by the NSMatrix from the field editor. The field editor, as mentioned in the NSTextField class specification, is the NSText object used to draw text in all cells in a window. See also setDelegate:.

deselectAllCells

```
- (void)deselectAllCells
```

Clears the receiver's selection, assuming that the NSMatrix allows an empty selection and is not an NSRadioModeMatrix. See also deselectSelectedCell, selectAll:.

deselectSelectedCell

```
- (void)deselectSelectedCell
```

Deselects the selected cell. If the selection mode is NSRadioModeMatrix, and an empty selection is not allowed, this method won't deselect the selected NSCell. This method doesn't redisplay the NSMatrix. See also deselectAllCells, selectedCell, selectAll:.

doubleAction

```
- (SEL)doubleAction
```

Returns the action sent by the NSMatrix to its target when the user double-clicks an entry. Unlike NSBrowser, this method returns NULL if there is no double-click action. The double-click action of an NSMatrix is sent after the appropriate single-click action (for the NSCell clicked, or for the NSMatrix if the NSCell doesn't have its own action). If there is no double-click action and the NSMatrix doesn't ignore multiple clicks, the single-click action is sent twice. See also setDoubleAction:, sendDoubleAction, sendAction.

drawCellAtRow:column:

```
- (void)drawCellAtRow:(int)row column:(int)column
```

Displays the cell at row and col if it's within the NSMatrix. See also highlightCell:atRow:column:.

drawsBackground

```
- (BOOL)drawsBackground
```

Returns YES if the receiver draws the background between cells. Returns NO otherwise. See also drawsCellBackground, backgroundColor.

drawsCellBackground

```
- (BOOL)drawsCellBackground
```

Returns YES if the receiver draws the background within cells. Returns NO otherwise. See also setDrawsCellBackground:, cellBackgroundColor, backgroundColor.

errorAction

```
- (SEL)errorAction
```

Returns the action method sent to the target of the NSMatrix when the user enters an illegal value for an NSCell's type (that is, user input errors), as set by NSCell's setEntryType: method and checked by NSCell's isEntryAcceptable: method. See also setErrorAction:.

getNumberOfRows:columns:

```
- (void)getNumberOfRows:(int *)rowCount
    columns:(int *)columnCount
```

Returns the number of matrix rows and columns within rowCount and columnCount. See also numberOfColumns, numberOfRows, makeCellAtRow:column:, getRow:column:forPoint:, getRow:column:ofCell:.

getRow:column:forPoint:

```
- (BOOL)getRow:(int *)row column:(int *)column
    forPoint:(NSPoint)aPoint
```

Gets the row and column position corresponding to aPoint, or sets each to -1 if aPoint isn't within the matrix. Returns YES if aPoint is within the matrix; NO otherwise. See also getRow:column:ofCell:.

getRow:column:ofCell:

```
- (BOOL)getRow:(int *)row column:(int *)column
    ofCell:(NSCell *)aCell
```

Gets the row and column position of aCell, or sets each to -1 if aCell is not found in the matrix. Returns YES if aCell is in the matrix; NO otherwise. See also getRow:column:forPoint:.

highlightCell:atRow:column:

```
- (void)highlightCell:(BOOL)flag atRow:(int)row column:(int)column
```

Highlights (if flag is YES), or unhighlights (if flag is NO) the cell at row, column within the NSMatrix by sending highlight:withFrame:inView: (NSCell) to the cell. The PostScript focus must be locked on the NSMatrix when this message is sent. See also drawCellAtRow:column:.

initWithFrame:

```
- (id)initWithFrame:(NSRect)frameRect
```

Initializes and returns the receiver, a new instance of NSMatrix, with default parameters in the given frame rectangle. The default font is the user's chosen system font in 12.0 point, the default NSCell size is 100.0 by 17.0 points, the default inter-cell spacing is 1.0 point by 1.0 point The new NSMatrix contains no rows or columns. The default mode is NSRadioModeMatrix (see the Class Description). See also initWithFrame:mode:cellClass:numberOfRows: numberOfColumns:, initWithFrame:mode:prototype:numberOfRows: numberOfColumns:, intercellSpacing.

initWithFrame:mode:cellClass:numberOfRows: numberOfColumns:

```
- (id)initWithFrame:(NSRect)frameRect mode:(int)aMode
    cellClass:(Class)classId numberOfRows:(int)rowsHigh
    numberOfColumns:(int)colsWide
```

Initializes a new NSMatrix object in frameRect, with aMode as the selection mode, classId as the class used to make new cells, and containing rowsHigh rows and colsWide columns. aMode can be one of four values:

- NSTrackModeMatrix - only track mouse inside the cells.
- NSHighlightModeMatrix - highlight the cell, then track, then unhighlight.
- NSRadioModeMatrix - allow no more than one selected cell.
- NSListModeMatrix - allow multiple selected cells.

The behavior for these values is more fully described in the "Class Description". The new NSMatrix creates and uses NSCells of class classId, which should be the return value of a class message sent to a subclass of NSCell. This method is the designated initializer for any NSMatrix that adds NSCells by creating instances of an NSCell subclass. See also initWithFrame:, initWithFrame:mode:prototype:numberOfRows: numberOfColumns:, intercellSpacing, setMode:.

initWithFrame:mode:prototype:numberOfRows: numberOfColumns:

```
- (id)initWithFrame:(NSRect)frameRect mode:(int)aMode
    prototype:(NSCell *)aCell numberOfRows:(int)rowsHigh
    numberOfColumns:(int)colsWide
```

Initializes a new NSMatrix object with aMode as the selection mode, aCell as the prototype copied to make new cells, and having rowsHigh rows and colsWide columns. aMode can be one of four values:

- NSTrackModeMatrix - only track mouse inside the cells.
- NSHighlightModeMatrix - highlight the cell, then track, then unhighlight.
- NSRadioModeMatrix - allow no more than one selected cell.
- NSListModeMatrix - allow multiple selected cells.

The behavior for these constants is more fully described in the Class Description. The new NSMatrix creates cells by copying aCell, which should be a subclass instance NSCell. This method is the designated initializer for any NSMatrix that adds cells by copying an instance of an NSCell subclass. See also initWithFrame:, initWithFrame:mode:cellClass:numberOfRows: numberOfColumns:, setMode:.

insertColumn:

```
- (void)insertColumn:(int)column
```

Inserts a new column of cells before column, creating new cells with makeCellAtRow:column:. If column is greater than the number of columns in the matrix, then enough columns are created to expand the matrix to be column columns wide. This method doesn't redraw. Your code may need to use the sizeToCells method after sending this method to resize the matrix to fit the newly added cells.

If the number of rows or columns in the matrix has been changed with `renewRows:columns:`, then `makeCellAtRow:column:` is invoked only if new cells are needed; since `renewRows:columns:` doesn't free cells, it just rearranges them. This allows you to grow and shrink a matrix without repeatedly creating and freeing the cells. See also `insertColumn:withCells:`, `insertRow:`, `insertRow:withCells:`, `removeColumn:`, `removeRow:`, `intercellSpacing`, `makeCellAtRow:column:`.

insertColumn:withCells:

```
- (void)insertColumn:(int)column withCells:(NSArray *)cellArray
```

Inserts a new column of cells at `column`, using those cells contained in `cellArray`, and expanding the matrix as much as necessary to make the matrix `column` columns wide. See also `insertColumn:`, `makeCellAtRow:column:`.

insertRow:

```
- (void)insertRow:(int)row
```

Inserts a new row of cells before `row`, creating new cells with `makeCellAtRow:column:`. If `row` is greater than the number of rows in the matrix, enough rows are created to expand matrix to be `row` rows high. This method doesn't redraw. Your code may need to use the `sizeToCells` method after sending this method to resize the matrix to fit the newly added cells.

If the number of rows or columns in the matrix has been changed with `renewRows:columns:`, then `makeCellAtRow:column:` is invoked only if new cells are needed (since `renewRows:columns:` doesn't free cells, it just rearranges them). This allows you to grow and shrink a matrix without repeatedly creating and freeing the cells. See also `insertRow:withCells:`, `insertColumn:`, `makeCellAtRow:column:`.

insertRow:withCells:

```
- (void)insertRow:(int)row withCells:(NSArray *)cellArray
```

Inserts a new row of cells at `row`, using those cells contained in `cellArray`, and expanding the matrix as much as necessary to make the matrix `row` rows wide. See also `insertRow:`, `makeCellAtRow:column:`.

intercellSpacing

```
- (NSSize)intercellSpacing
```

Returns the vertical and horizontal spacing between cells. See also `setIntercellSpacing:`, `makeCellAtRow:column:`.

isAutoscroll

```
- (BOOL)isAutoscroll
```

Returns YES if the matrix automatically scrolls when mouse is dragged outside the matrix after a mouse-down event inside the matrix. See also `setAutoscroll:`, `setScrollable:`, `scrollCellToVisibleAtRow:column:`.

isSelectionByRect

```
- (BOOL)isSelectionByRect
```

Returns YES if a rectangle of cells in the matrix can be selected by dragging the cursor. Returns NO otherwise.

makeCellAtRow:column:

```
- (NSCell *)makeCellAtRow:(int)row column:(int)column
```

Creates a new matrix cell. If the matrix has a prototype cell, it's copied to create the new cell; if the matrix has a cell class set, it allocates and initializes (with `init`) an instance of that class; if the matrix has not had a cell class set, the default class, `NSActionCell`, is used. The new cell's font is set to the matrix font. This method returns the newly created cell. Your code should never invoke this method directly; it's used by the add and insert row and column methods of this class when a cell must be created. The default implementation ignores it's arguments, thereby providing the cell but not inserting it. It should be overridden by subclasses to provide more specific initialization of cells. See also `putCell:atRow:column:`, `addColumn`, `cellFrameAtRow:column:`, `getNumberOfRows:columns:`, `setCellSize:`, `sortUsingFunction:context:`, `sortUsingSelector:`.

mode

```
- (NSMatrixMode)mode
```

Returns the selection mode of the matrix. For a description of the matrix modes, see the NSMatrix "Class Description". See also setMode:.

mouseDown:

```
- (void)mouseDown:(NSEvent *)theEvent
```

Your code should never invoke this method, but you may override it to implement mouse tracking different than NSMatrix. The NSMatrix response depends on its selection mode, as explained in the "Class Description". In any selection mode, a mouse-down in an editable text cell immediately enters text editing mode. A double-click in any other kind of cell sends the double-click action of the NSMatrix (if there is one) in addition to the single-click action. See also acceptsFirstMouse:, mouseDownFlags.

mouseDownFlags

```
- (int)mouseDownFlags
```

Returns the modifier flags (for example, NSShiftKeyMask) that were in effect at the mouse-down event that started the current tracking session. Use this method if you want to access these flags, but don't want the overhead of having to use sendActionOn: (NSCell) to add mouse-down masks to every cell to get them. This method is valid only during tracking; it's not useful if the target of the matrix initiates another tracking loop as part of its action method. See the "Event Handling" section of the Application "Kit's Types and Constants" chapter for more information on event flags. See also mouseDown:.

numberOfColumns

```
- (int)numberOfColumns
```

Returns the number of matrix columns. See also numberOfRows, getNumberOfRows:columns:.

numberOfRows

```
- (int)numberOfRows
```

Returns the number of matrix rows. See also `numberOfColumns`, `getNumberOfRows:columns:`.

nextText

```
- (id)nextText
```

Returns the object to be selected when the user presses Tab while editing the last text cell. See also `setNextText:`, `previousText`, `setPreviousText:`.

performKeyEquivalent:

```
- (BOOL)performKeyEquivalent:(NSEvent *)theEvent
```

If there is a cell in the matrix that has a key equivalent equal to that in `theEvent`, that cell is made to react as if the user had clicked it by highlighting, changing its state as appropriate, sending its action if it has one, and then unhighlighting. Returns `YES` if a cell in the matrix responds to the key equivalent in `theEvent`, `NO` if no cell responds. Your code should never send this message; it is sent when the matrix or one of its superviews is the first responder and the user presses a key. You may want to override this method to change the way key equivalents are performed or displayed, or to disable them in your subclass. See also `acceptsFirstMouse:`.

previousText

```
- (id)previousText
```

Returns the object to be selected when the user presses Shift-Tab while editing the first text cell. See also `setPreviousText:`, `nextText`, `setNextText:`.

prototype

```
- (id)prototype
```

Returns the prototype cell copied to make new cells, or `nil` if there is none. See also `setPrototype:`, `initWithFrame:mode:prototype:numberOfRows: numberOfColumns:`, `makeCellAtRow:column:`.

putCell:atRow:column:

```
- (void)putCell:(NSCell *)newCell atRow:(int)row column:(int)column
```

Replaces the cell at `row` and `column` with `newCell`, and redraws. See also `makeCellAtRow:column:`.

removeColumn:

```
- (void)removeColumn:(int)column
```

Removes the column at position `column` and releases the cells. Doesn't redraw. Your code should normally send `sizeToCells` after invoking this method to resize the matrix so it fits the reduced cell count. See also `insertColumn:`, `removeRow:`, `makeCellAtRow:column:`.

removeRow:

```
- (void)removeRow:(int)row
```

Removes the row at position `row` and releases the cells. Doesn't redraw. Your code should normally send `sizeToCells` after invoking this method to resize the matrix so it fits the reduced cell count. See also `insertRow:`, `removeColumn:`, `makeCellAtRow:column:`.

renewRows:columns:

```
- (void)renewRows:(int)newRows columns:(int)newColumns
```

Rearranges the number of rows and columns in the matrix, using the existing cells. This method uses the same cells as before the message is sent, creating new cells only if the new size is larger; it never frees cells. This method doesn't display the matrix even if autodisplay is on. Your code should normally send `sizeToCells` after invoking this method to resize the matrix so it fits the changed cell arrangement. This method deselects all cells in the matrix. See also `makeCellAtRow:column:`.

resetCursorRects

```
- (void)resetCursorRects
```

Resets cursor rectangles so that the cursor becomes an I-beam over text cells. Sends `resetCursorRect:inView:` (`NSCell`) to each cell in the matrix. (Any cell that has a cursor rectangle to set up should send the message `addCursorRect:cursor:`, inherited from `NSView`, back to the matrix). See also `resetCursorRects` (`NSView`).

scrollCellToVisibleAtRow:column:

```
- (void)scrollCellToVisibleAtRow:(int)row column:(int)column
```

If the matrix is in a scrolling view, this method scrolls the matrix so that the cell at `row` and `column` is visible. See also `isAutoscroll`, `setAutoscroll:`, `setScrollable:`.

selectAll:

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

If the matrix mode is not `NSRadioModeMatrix`, then all the cells in the matrix are selected and highlighted, and the matrix is redisplayed. The currently selected cell is unaffected (it remains selected). Editable text cells are not affected. See also `deselectAllCells`, `deselectSelectedCell`, `selectCellAtRow:column:`, `selectCellWithTag:`, `selectedCell`, `selectedCells`, `selectedColumn`, `selectedRow`.

selectCellAtRow:column:

```
- (void)selectCellAtRow:(int)row column:(int)column
```

Selects the cell at position (`row`, `col`) in the matrix. An editable text cell's text is selected. If either `row` or `col` is –1, then the current selection is cleared unless the matrix is in `NSRadioModeMatrix` and does not allow empty selection. Redraws the affected cells. See also `selectAll:`.

selectCellWithTag:

```
- (BOOL)selectCellWithTag:(int)anInt
```

Selects the cell with the tag `anInt`. An text cell's text is selected. Returns `nil` if no cell with the given tag exists. See also `selectAll:`.

selectText:

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

If `sender` is the next `NSText` object of the matrix (as set with `setNextText:`), the text in the last selectable text cell (the one lowest and furthest to the right) is selected; otherwise, the text of the first selectable text cell is selected. See also `selectTextAtRow:column:`, `textDidChange:`.

selectedCell

```
- (id)selectedCell
```

Returns the currently selected cell, or `nil` if no cell is selected. See also `selectedCells`, `selectedColumn`, `selectedRow`, `selectAll:`.

selectedCells

```
- (NSArray *)selectedCells
```

Returns an array containing the selected cells. See also `selectedCell`, `selectedColumn`, `selectedRow`, `selectAll:`.

selectedColumn

```
- (int)selectedColumn
```

Returns the column of the selected cell or –1 if no column has been selected. See also selectedRow, `selectedCell`, `selectAll:`.

selectedRow

```
- (int)selectedRow
```

Returns the row of the selected cell or –1 if no row has been selected. See also `selectedColumn`, `selectedCell`, `selectAll:`.

selectTextAtRow:column:

```
- (id)selectTextAtRow:(int)row column:(int)column
```

Select the text of the cell at (row, col) in the matrix, if there is such a cell and its text is selectable. Returns the cell whose text was selected, the matrix if such a cell wasn't found, or nil if the cell was found but wasn't enabled or wasn't selectable. See also selectText:, textDidChange:.

sendAction

```
- (BOOL)sendAction
```

Sends the selected cell's action method, or the NSMatrix's action if the cell doesn't have one. Returns YES if a target receives the action; beeps and returns NO otherwise. If the matrix has no selected and enabled cell, NO is returned. If the selected cell has both an action method and a target, its action method is sent to its target. If the cell doesn't have a target (nil), the cell sends its action method to the matrix target. If the cell doesn't have an action method (nil), the matrix sends its action method to its target. See also sendAction:to:forAllCells:, sendDoubleAction, setDoubleAction:, errorAction, setErrorAction:.

sendAction:to:forAllCells:

```
- (void)sendAction:(SEL)aSelector to:(id)anObject
    forAllCells:(BOOL)flag
```

Sends aSelector to anObject for all matrix cells if flag is YES. aSelector must represent a method that takes a single argument: the id of the current cell in the iteration. aSelector's return value must be a BOOL. Iteration begins with the cell in the upper-left corner of the matrix, proceeding through all entries in the first row, then on to the next. If aSelector returns NO for any cell, this method terminates immediately without sending the message for other cells. If it returns YES, this method keeps sending the message.

This method is *not* invoked to send action messages to target objects in response to mouse-down events in the matrix. Instead, you can invoke it if you want to have multiple cells in a matrix interact with an object. For example you could use it to verify the titles in a list of items, or to enable a series of radio buttons based on their purpose in relation to anObject. Returns YES if a target receives the action; beeps and returns NO otherwise. See also sendAction.

sendDoubleAction

```
- (void)sendDoubleAction
```

Sends the action method corresponding to a double-click if it exists. If the selected cell is not enabled, the method returns. If the matrix has a double-click action, that message is sent to the matrix target. If not, then if the selected cell (as returned by `selectedCell`) has an action, that message is sent to the selected cell's target. If the selected cell also has no action, then the action of the matrix is sent to the target of the matrix. This method only sends an action if the selected cell is enabled. Your code shouldn't invoke this method; it's sent in response to a double-click event in the matrix. You may want to override it to change the search order for an action to send. See also `doubleAction`, `sendAction`.

setAllowsEmptySelection:

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

If `flag` is `YES`, then the matrix will allow zero cells to be selected. If `flag` is `NO`, then the matrix disallows zero selected cells. This setting effects `NSRadioModeMatrix` and `NSListModeMatrix` matrices only. See also `allowsEmptySelection`.

setAutoscroll:

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

If `flag` is `YES` and the matrix is in a scrolling view, it will be automatically scrolled whenever a the mouse is dragged outside the matrix after a mouse-down event within its bounds. See also `isAutoscroll`.

setAutosizesCells:

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

If `flag` is `YES`, then whenever the matrix is resized, the sizes of the cells change in proportion, keeping the inter-cell space constant; further, this method verifies that the cell sizes and inter-cell spacing add up to the exact size of the matrix, adjusting the size of the cells and updating the matrix if they

don't. If `flag` is `NO`, then the inter-cell space changes when the matrix is resized, with the cell size remaining constant. See also `autosizesCells`, `intercellSpacing`.

setBackgroundColor:

```
- (void)setBackgroundColor:(NSColor *)aColor
```

Sets the background color for the matrix to `aColor`. This color is used to fill the space between cells or the space behind any non-opaque cells. Marks the matrix as needing redrawing. See also `backgroundColor`.

setCellBackgroundColor:

```
- (void)setCellBackgroundColor:(NSColor *)aColor
```

Sets the background color for the matrix cells to `aColor`. This color is used to fill the space behind non-opaque cells. Marks the matrix as needing redrawing. See also `cellBackgroundColor`, `backgroundColor`.

setCellClass:

```
- (void)setCellClass:(Class)classId
```

Configures a matrix to use instances of `classId` when creating new cells. `classId` should be the `id` of a subclass of `NSCell`, obtained by sending the `class` message (`NSObject`) to either the cell subclass object or to an instance of that subclass. The cell class is the method set with the class method `setCellClass:`; the default cell class is `NSActionCell`. You only need to use this method with matrices initialized with `initFrame:`, since the other initializers allow you to specify an instance-specific cell class or cell prototype. See also `cellClass`, `initWithFrame:mode:cellClass:numberOfRows:numberOfColumns:`.

setCellSize:

```
- (void)setCellSize:(NSSize)aSize
```

Sets the width and the height of each cell in the matrix to `aSize`. This may change the size of the matrix. Does not redraw the matrix. See also `cellSize`, `makeCellAtRow:column:`.

setDelegate:

- (void)setDelegate:(id)anObject

Sets the object to which the matrix will forward messages from the field editor. These messages include textDidBeginEditing:, textDidEndEditing:, textDidChange:, textShouldBeginEditing:, and textShouldEndEditing:. See also delegate.

setDoubleAction:

- (void)setDoubleAction:(SEL)aSelector

Make aSelector the action method sent to the matrix target when the user double-clicks a cell. A double-click action is always sent after the appropriate single-click action: the cell's single-click action method if it has one, otherwise the single-click action method of the matrix. If a matrix has no double-click action set, then by default a double-click is treated as a single-click. See also doubleAction, sendAction.

setDrawsBackground:

- (void)setDrawsBackground:(BOOL)flag

Sets whether the receiver draws the background between cells. See also drawsCellBackground, backgroundColor.

setDrawsCellBackground:

- (void)setDrawsCellBackground:(BOOL)flag

Sets whether the receiver draws the background within the cells. See also cellBackgroundColor, backgroundColor.

setErrorAction:

- (void)setErrorAction:(SEL)aSelector

Sets the action method sent to the matrix target when the user enters an illegal value in a text cell for that cell's entry type as set by NSCell's setEntryType: method and checked by NSCell's isEntryAcceptable: method.

setIntercellSpacing:

```
- (void)setIntercellSpacing:(NSSize)aSize
```

Sets the vertical and horizontal spacing between cells to `aSize`. Doesn't redraw the matrix. See also `intercellSpacing`, `makeCellAtRow:column:`.

setMode:

```
- (void)setMode:(NSMatrixMode)aMode
```

Sets the selection mode of the matrix to one of the following values:

- `NSTrackModeMatrix` - Track mouse only inside the cells
- `NSHighlightModeMatrix` - Highlight the cell, then track, then unhighlight
- `NSRadioModeMatrix` - Allow no more than one selected cell
- `NSListModeMatrix` - Allow multiple selected cells

See the Class Description for more information on these modes. See also `mode`.

setNextText:

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

Sets `anObject` as the object whose text is selected when the user presses Tab while editing the last editable text cell. `anObject` should respond to the `selectText:` message. If `anObject` also responds to both `selectText:` and `setPreviousText:`, it is sent `setPreviousText:` with the receiving matrix as the argument; this builds a two-way connection, so that pressing Tab in the last text cell selects `anObject`'s text, and pressing Shift-Tab in `anObject` selects the last text cell of the matrix. See also `nextText`.

setPreviousText:

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

Sets `anObject` as the object whose text is selected when the user presses Shift-Tab while editing the first editable text cell. `anObject` should respond to the `selectText:` message. Your code shouldn't need to use this method directly, since it's invoked automatically by `setNextText:`. In deference to `setNextText:`, this method doesn't build a two-way connection. See also `previousText`.

setPrototype:

```
- (void)setPrototype:(NSCell *)aCell
```

Sets the prototype cell that is copied whenever a new cell needs to be made. `aCell` should be an instance of a subclass of cell. If a matrix has a prototype cell, it doesn't use its cell class object to create new cells; if you want your matrix to use its cell class, invoke this method with `nil` as the argument. The matrix is considered to own the prototype, and will free it when the matrix is itself freed; be sure to make a copy of an instance that your code may use elsewhere.

If you implement your own cell subclass for use as a prototype with a matrix, make sure your cell does the right thing when it receives a `copy` message. For example, `NSObject`'s `copy` copies only pointers, not what they point to—sometimes this is what it should do, sometimes not. The best way to implement `copy` when you subclass cell is send `copy` to `super`, then copy instance variable values (for example, title strings) into your subclass instance individually. Also, be careful that freeing the prototype will not damage any of the copies that were made and put into the matrix (due to shared pointers that are freed, for example). See also `prototype`, `cellClass`, `initWithFrame:mode:prototype:numberOfRows: numberOfColumns:`.

setScrollable:

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

If `flag` is `YES`, makes all the cells scrollable so that the text they contain scrolls to remain in view if the user types past the edge of the cell. See also `isAutoscroll`.

setSelectionByRect:

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

Sets whether a user can drag a rectangular selection (the default is `YES`). If `flag` is `NO`, selection is on a row-by-row basis. See also `isSelectionByRect`.

setSelectionFrom:to:anchor:highlight:

```
- (void)setSelectionFrom:(int)startPos to:
    (int)endPos anchor:(int)anchorPos highlight:(BOOL)flag
```

Programmatically selects a range of cells. `startPos`, `endPos`, and `anchorPos` are cell positions, counting from 0 in row order from the upper left cell of the matrix. For example, the third cell in the top row would be number 2. `startPos` and `endPos` are used to mark where the user would have pressed the mouse button and released it. `anchorPos` locates the "last selected cell" with regard to extending the selection by Shift- or Alternate-clicking. Finally, `flag` determines whether cells selected by this method are highlighted. See also `selectAll:`.

setState:atRow:column:

```
- (void)setState:(int)value atRow:(int)row column:(int)column
```

Sets the state of the cell at row `row` and column `col` to `value`. For radio-mode matrices, this is identical to `selectCellAtRow:column:` except that the state can be set to any arbitrary `value`. (If in radio-mode and empty selection is allowed, and value is 0, then the cells are cleared). Affected cells are redrawn.

setValidateSize:

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

Sets whether the cell size needs to be recalculated.

If `flag` is `YES`, then the size information in the matrix is assumed correct. If `flag` is `NO`, then cell size will be recalculated. See also `autosizesCells`.

sizeToCells

```
- (void)sizeToCells
```

Resizes the matrix to fit its cells exactly. Doesn't redraw the matrix. See also `autosizesCells`.

sortUsingFunction:context:

```
- (void)sortUsingFunction:(int (*)(id element1, id element2,
    void *userData))comparator context:(void *)context
```

Sorts the receiver's cells in ascending order as defined by the comparison function `comparator`. `context` is passed as the function's third argument. See also `sortUsingSelector:`, `sortUsingFunction:context:` (`NSMutableArray`).

sortUsingSelector:

```
- (void)sortUsingSelector:(SEL)comparator
```

Sorts the receiver's cells in ascending order as defined by the comparison method `comparator`. See also `sortUsingFunction:context:`, `sortUsingFunction:context:` (`NSMutableArray`).

textDidBeginEditing:

```
- (void)textDidBeginEditing:(NSNotification *)notification
```

Invoked when there's a change in the text after the receiver gains first responder status. Default behavior is to pass this message on to the text delegate. This method posts the `NSControlTextDidBeginEditingNotification` notification with the receiving object and, in the notification's dictionary, the text object (with the key `NSFieldEditor`) to the default notification center. See also `textDidEndEditing:`, `textDidChange:`, `setDelegate:`.

textDidChange:

```
- (void)textDidChange:(NSNotification *)notification
```

Invoked upon a key-down event or paste operation that changes the receiver's contents. Default behavior is to pass this message on to the text delegate. This method posts the `NSControlTextDidChangeNotification` notification with the receiving object and, in the notification's dictionary, the text object (key `NSFieldEditor`) to the default notification center. See also `textDidBeginEditing:`, `textDidEndEditing:`, `textShouldBeginEditing:`, `textShouldEndEditing:`, `setDelegate:`.

textDidEndEditing:

```
- (void)textDidEndEditing:(NSNotification *)notification
```

Invoked when text editing ends and then forwarded to the text delegate. This method posts the notification `NSControlTextDidEndEditingNotification` with the receiving object and, in the notification's dictionary, the text object (with the key `NSFieldEditor`) to the default notification center. See also `textDidBeginEditing:`, `textDidChange:`, `setDelegate:`.

`textShouldBeginEditing:`

```
- (BOOL)textShouldBeginEditing:(NSText *)textObject
```

Invoked to let the `NSTextField` respond to impending changes to its text and then forwarded to the text delegate. See also `textShouldEndEditing:`, `textDidChange:`, `setDelegate:`.

`textShouldEndEditing:`

```
- (BOOL)textShouldEndEditing:(NSText *)textObject
```

Invoked to let the `NSTextField` respond to impending loss of first responder status and then forwarded to the text delegate. See also `textShouldBeginEditing:`, `textDidChange:`, `setDelegate:`.

NSMenu

Inherits From:	NSPanel : NSWindow : NSResponder : NSObject
Conforms To:	NSCoding (NSResponder) NSObject (NSObject)
Declared In:	AppKit / NSMenu.h

Class Description

This class defines an object that manages an application's menus. An `NSMenu` object displays a list of items that a user can choose from. When an item is clicked, it may either issue a command directly or bring up another menu, a *submenu* that offers further choices. An `NSMenu` object's choices are implemented as a column of `NSMenuCell`s in an `NSMatrix`.

Each NSMenuCell can be configured to send its action message to a target, or to bring up a submenu. When the user clicks a submenu item, the submenu is displayed on the screen and attached to its supermenu so that if the user drags the supermenu, the submenu follows it. A submenu may also be torn away from its supermenu, in which case it displays a close button.

Exactly one NSMenu created by the application is designated as the main menu for the application (with NSApplication's setMainMenu: method). This menu is displayed on top of all other windows whenever the application is active, and should never display a close button (because the main menu doesn't have a supermenu). See the NSMenuCell and NSMatrix class specificiations for more details.

Method Types

Activity	Class Method
Controlling allocation zones	+ menuZone + setMenuZone:
Initializing a new NSMenu	– initWithTitle:
Setting up the menu commands	– addItemWithTitle:action:keyEquivalent: – insertItemWithTitle:action:keyEquivalent: atIndex: – itemArray – itemMatrix – setItemMatrix:
Finding and removing menu items	– itemWithTag: – itemWithTitle: – removeItem:
Building submenus	– setSubmenu:forItem: – submenuAction:
Managing NSMenu windows	– attachedMenu – isAttached – isTornOff – locationForSubmenu: – sizeToFit – supermenu
Displaying the menu	– autoenablesItems – setAutoenablesItems:

Class Methods

menuZone

```
+ (NSZone *)menuZone
```

Returns the zone from which new NSMenus should be allocated. If there isn't one, creates and returns a zone named "Menus." After invoking this method, you should allocate all new NSMenus from this zone.

setMenuZone:

```
+ (void)setMenuZone:(NSZone *)zone
```

Sets the zone from which NSMenus should be allocated. See also menuZone.

Instance Methods

addItemWithTitle:action:keyEquivalent:

```
- (id <NSMenuItem>)addItemWithTitle:(NSString *)aString
    action:(SEL)aSelector
    keyEquivalent:(NSString *)charCode
```

Adds a new menu item (command) named aString to the end of the receiving NSMenu and returns the the menu item created. The menu item's action method is set to aSelector. charCode is set as the menu item's key equivalent. The command name and key equivalent aren't checked for duplications within the same NSMenu (or any other NSMenu); be sure to assign them uniquely. See also insertItemWithTitle:action:keyEquivalent: atIndex:.

attachedMenu

```
- (NSMenu *)attachedMenu
```

Returns the NSMenu attached to the receiver or nil if there is no such object. See also isAttached, isTornOff, locationForSubmenu:, sizeToFit, supermenu, setSubmenu:forItem:, submenuAction:.

autoenablesItems

```
- (BOOL)autoenablesItems
```

Returns YES if the receiver enables and disables its NSMenuCells based on user actions, and NO otherwise. The default is YES. (See the NSMenuActionResponder informal protocol in the Application Kit's Protocols chapter). See also setAutoenablesItems:.

initWithTitle:

```
- (id)initWithTitle:(NSString *)aTitle
```

Initializes and returns the receiver, a new instance of NSMenu, with the title aTitle. The menu is positioned in the upper left corner of the screen, and has no command items. A new menu must receive an orderFront: message to be displayed on the screen; the NSApplication object takes care of this for standard NSMenus. The NSMenu is created as a buffered, menu-style window. All NSMenus have an event mask that excludes keyboard events, so they never become the key window or main window. See also addItemWithTitle:action:keyEquivalent:, insertItemWithTitle:action:keyEquivalent: atIndex:.

insertItemWithTitle:action:keyEquivalent: atIndex:

```
- (id <NSMenuItem>)insertItemWithTitle:(NSString *)aString
action:(SEL)aSelector
    keyEquivalent:(NSString *)charCode atIndex:(unsigned int)index
```

Adds a new item at index having the title aString, action method aSelector, and key equivalent charCode. Returns the new menu item. See also addItemWithTitle:action:keyEquivalent:, initWithTitle:.

isAttached

```
- (BOOL)isAttached
```

Returns YES if the receiving menu is attached to another menu and NO otherwise. See also attachedMenu.

isTornOff

```
- (BOOL)isTornOff
```

Returns NO if the receiver is attached to another menu (or if it's the main menu) and YES otherwise. See also attachedMenu.

itemArray

```
- (NSArray *)itemArray
```

Returns an array of the receiver's menu items. See also itemMatrix.

itemMatrix

```
- (NSMatrix *)itemMatrix
```

Returns the NSMatrix of NSMenuCell items, which your code can use to add or rearrange command items directly. Be sure to send sizeToFit after altering the NSMatrix, as the NSMenu won't know that the NSMatrix has been altered. Note that this method is not part of the OpenStep specification. See also setItemMatrix:.

itemWithTag:

```
- (id)itemWithTag:(int)aTag
```

Returns the menu item that has aTag as its tag. If you use menu item tags, each menu cell should have a unique tag. See also itemWithTitle:.

itemWithTitle:

```
- (id <NSMenuItem>)itemWithTitle:(NSString *)aTitle
```

Returns the the first menu item with title aTitle. See also itemWithTag:, removeItem:.

locationForSubmenu:

```
- (NSPoint)locationForSubmenu:(NSMenu *)aSubmenu
```

Determines where to display an attached submenu when it's brought up. The returned NSPoint specifies where the lower-left corner of the submenu should be drawn. NSMenu invokes this method whenever it brings up a submenu. By default, the submenu is to the right of its supermenu, with its title bar aligned with the supermenu's. Your code need never directly use this method, but may override it to cause the submenu to be attached at a different location. See also attachedMenu.

removeItem:

```
- (void)removeItem:(id <NSMenuItem>)item
```

Removes the given menu item from the menu.

setAutoenablesItems:

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

Sets whether the receiver enables and disables its NSMenuCells. (See the NSMenuActionResponder informal protocol in the Application Kit's Protocols chapter). See also autoenablesItems.

setItemMatrix:

```
- (void)setItemMatrix:(NSMatrix *)aMatrix
```

Replaces the current matrix of items within the menu with aMatrix. Note that this method is not part of the OpenStep specification. See also itemMatrix.

setSubmenu:forItem:

```
- (void)setSubmenu:(NSMenu *)aMenu
    forItem:(Id <NSMenuItem>)item
```

Sets aMenu as the submenu of the receiving NSMenu, controlled by the item. item's target is set to aMenu, its action method to submenuAction:, and its icon to the arrow indicating that it brings up a submenu. This method doesn't remove item's key equivalent. If aMenu was on screen, it won't be removed from the screen or moved until it's first brought up as a submenu. See also submenuAction:.

sizeToFit

```
- (void)sizeToFit
```

Resizes the receiver to exactly fit the command items. First this method sizes the menu's NSMatrix to its NSMenuCells, so that all items fit in as small a rectangle as possible, and then fits the NSMenu to the resized NSMatrix. Use this method after you've added or altered items by sending messages directly to the NSMatrix. When the NSMenu is resized, its upper left corner remains fixed. After performing any necessary resizing, this method redisplays the menu. See also attachedMenu.

submenuAction:

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

This is the action method sent to a submenu associated with an entry in an NSMenu. If sender's window is a visible NSMenu, the receiver attaches and displays itself as a submenu of the sender's NSMenu; otherwise, it does nothing. sender should be the NSMatrix containing the NSMenuCell that brings up the submenu. See also setSubmenu:forItem:.

supermenu

```
- (NSMenu *)supermenu
```

Returns the receiver's supermenu. See also attachedMenu.

NSMenuCell

Inherits From:	NSButtonCell : NSActionCell : NSCell : NSObject
Conforms To:	NSCoding, NSCopying (NSCell) NSObject (NSObject)
Declared In:	AppKit/NSMenuCell.h

Class Description

NSMenuCell is a NSButtonCell subclass that defines objects that are used in menus. NSMenuCells draw their text left-justified and show an optional key equivalent or submenu arrow on the right. See the NSMenu class specification for more information.

Method Types

Activity	Class Method
Checking for a submenu	– hasSubmenu
Managing user key equivalents	+ setUsesUserKeyEquivalents: + usesUserKeyEquivalents – userKeyEquivalent

Class Methods

setUsesUserKeyEquivalents:

```
+ (void)setUsesUserKeyEquivalents:(BOOL)flag
```

If `flag` is `YES`, `NSMenuCell`s conform to user preferences (user's defaults) for key equivalents; otherwise, the key equivalents originally assigned to the `NSMenuCell`s are used. See also `usesUserKeyEquivalents`.

usesUserKeyEquivalents

```
+ (BOOL)usesUserKeyEquivalents
```

Returns `YES` if `NSMenuCell`s conform to user preferences for key equivalents; otherwise, returns `NO`. See also `setUsesUserKeyEquivalents:`.

Instance Methods

hasSubmenu

```
- (BOOL)hasSubmenu
```

Returns `YES` if the receiving menu cell brings up a submenu, and `NO` otherwise.

userKeyEquivalent

```
- (NSString *)userKeyEquivalent
```

If the `NSMenuCell` class has been configured to use user key equivalents, returns the user-assigned key equivalent for the `NSMenuCell`.

NSOpenPanel

Inherits From:	NSSavePanel : NSPanel : NSWindow : NSResponder : NSObject
Conforms To:	NSCoding (NSResponder) NSObject (NSObject)
Declared In:	AppKit/NSOpenPanel.h

Class Description

`NSOpenPanel` provides the Open panel of the OpenStep user interface. Applications use the Open panel as a convenient way to query the user for the name of a file to open. The Open panel can only be run modally.

Most of this class's behavior is defined by its superclass, `NSSavePanel`. `NSOpenPanel` adds to this behavior by:

- Letting you specify the types (by file-name extension) of the items that will appear in the panel
- Letting the user select files, directories, or both
- Letting the user select multiple items at a time

Typically, you access an `NSOpenPanel` by invoking the `openPanel` method. When the class receives an `openPanel` message, it tries to reuse an existing panel rather than create a new one. If a panel is reused, its attributes are reset to the default values so that the effect is the same as receiving a new panel. Because Open panels may be reused, you shouldn't modify the instance returned by `openPanel` except through the methods listed below (and those inherited from its superclass, `NSSavePanel`). For example, you can set the panel's title and whether it allows multiple selection, but not the arrangement of the buttons within the panel. If you must modify the Open panel substantially, create and manage your own instance using the `alloc...` and `init...` methods rather than the `openPanel` method.

Note that `NSOpenPanel` (and `NSSavePanel`) automatically "remembers" the last directory the user traversed to. That is, anytime a Save or Open panel is shown to the user, the default directory it displays is the directory the user was at the last time they clicked "ok", or double-clicked a file. If no such previous directory exists, the panels will go to the user's home directory.

Method Types

Activity	Class Method
Accessing the NSOpen panel	+ openPanel
Filtering files	– allowsMultipleSelection – canChooseDirectories – canChooseFiles – setAllowsMultipleSelection: – setCanChooseDirectories: – setCanChooseFiles:
Querying the chosen files	– filenames
Running the NSOpenPanel	– runModalForTypes: – runModalForDirectory:file:types:

Class Methods

openPanel

```
+ (NSOpenPanel *)openPanel
```

Returns an `NSOpenPanel` object having default initialization.

Instance Methods

allowsMultipleSelection

```
- (BOOL)allowsMultipleSelection
```

Returns `YES` if the user can select more than one file in the browser. If multiple files are allowed, then the `filename` method—inherited from `NSSavePanel`—returns a non-`NULL` value only if one and only one file is selected. By contrast, `NSOpenPanel`'s `filenames` method always returns the selected files, even if only one file is selected. A further distinction between the two methods is that the inherited `filename` method always returns a fully specified path, while the `filenames` method doesn't; the names it returns are always relative to the path returned by `directory` (`NSSavePanel`). See also `setAllowsMultipleSelection:`.

canChooseDirectories

```
- (BOOL)canChooseDirectories
```

Returns YES if the panel allows the user to choose directories. See also `setCanChooseDirectories:`, `canChooseFiles`, `allowsMultipleSelection`.

canChooseFiles

```
- (BOOL)canChooseFiles
```

Returns YES if the panel allows the user to choose files. See also `setCanChooseFiles:`, `canChooseDirectories`, `allowsMultipleSelection`.

filenames

```
- (NSArray *)filenames
```

Returns an array containing the full path names of the selected files and directories. This list will be valid even if `allowMultipleSelections` is `NO`, in which case this method returns a single entry. This is the preferred method to get the name or names of any files that the user has chosen.

runModalForTypes:

```
- (int)runModalForTypes:(NSArray *)fileTypes
```

Invokes the `runModalForDirectory:file:types:` method, using the last directory from which a file was chosen as the path argument. Returns the value returned by that method. See also `runModalForDirectory:file:types:`.

runModalForDirectory:file:types:

```
- (int)runModalForDirectory:(NSString *)path
    file:(NSString *)filename types:(NSArray *)fileTypes
```

Displays the panel and begins its event loop. The panel displays the files in `path` that match the types in `fileTypes` (an array of `NSString` objects), with `filename` selected. Returns `NSOKButton` (if the user clicks the OK button) or `NSCancelButton` (if the user clicks the Cancel button). See also `runModalForTypes:`.

setAllowsMultipleSelection:

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

Sets multiple file (and directory) selection when flag is YES. See also `allowsMultipleSelection`.

setCanChooseDirectories:

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

Sets whether the user can choose directories. See also `canChooseDirectories`.

setCanChooseFiles:

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

Sets whether the user can choose files. See also `canChooseFiles`.

NSPageLayout

Inherits From:	NSPanel : NSWindow : NSResponder : NSObject
Conforms To:	NSCoding (NSResponder) NSObject (NSObject)
Declared In:	AppKit/NSPageLayout.h

Class Description

`NSPageLayout` is a type of `NSPanel` that queries the user for information such as paper type and orientation. This information is stored in an `NSPrintInfo` object, and is later used when printing. The `NSPageLayout` panel is created, displayed, and run (in a modal loop) when a `runPageLayout:` message is sent to the `NSApplication` object. By default, this message is sent up the responder chain when the user clicks the Page Layout menu item.

Typically, you access an `NSPageLayout` panel by invoking the `pageLayout` method. When the class receives a `pageLayout` message, it tries to reuse an existing panel rather than create a new one. If a panel is reused, its attributes

are reset to the default values so that the effect is the same as receiving a new panel. Because Page Layout panels may be reused, you shouldn't modify the instance returned by `pageLayout`, except through the methods listed below. If you must modify the Page Layout panel in other ways than those allowed by its methods, create and manage your own instance using the `alloc...` and `init...` methods rather than the `pageLayout` method.

You can add your own controls to the Page Layout panel through the `setAccessoryView:` method. The panel is automatically resized to accommodate the `NSView` that you've added. Note that you can't retrieve the `NSPageLayout`'s settings through messages to the page layout panel object—`NSPageLayout` does not have accessor methods to obtain the state of its controls. If controls you add through an accessory view require the values of the existing controls in the page layout panel (or vice versa), access `NSPageLayout`'s controls using the tags defined in `AppKit/NSPageLayout.h` as arguments to `viewWithTag:` messages to the page layout panel object. Controls thus returned can then be queried for their state.

Method Types

Activity	Class Method
Creating an NSPageLayout instance	+ pageLayout
Running the panel	– runModal – runModalWithPrintInfo:
Customizing the panel	– accessoryView – setAccessoryView:
Updating the panel's display	– convertOldFactor:newFactor: – pickedButton: – pickedOrientation: – pickedPaperSize: – pickedUnits:
Communicating with the NSPrintInfo object	– printInfo – readPrintInfo – writePrintInfo

Class Methods

pageLayout

```
+ (NSPageLayout *)pageLayout
```

Returns the default `NSPageLayout` object, creating it if necessary.

Instance Methods

accessoryView

```
- (NSView *)accessoryView
```

Returns the `NSPageLayout`'s accessory view. See also `setAccessoryView:`.

convertOldFactor:newFactor:

```
- (void)convertOldFactor:(float *)old newFactor:(float *)new
```

The standard unit used to measure a paper's dimensions is a point (for example, the PrintInfo object defines a paper's size in units of points). This method returns, by reference, a value that expresses the ratio between a point and the currently chosen unit of measurement. In general, both `old` and `new` are set to this value. The only time the values returned in the arguments differ is when the unit of measurement is being changed. Specifically, if you invoke this method from within `pickedUnits:`, `old` gives the old ratio and `new` gives the new ratio. Note that this method is implementation-dependent, and not part of the OpenStep specification.

pickedButton:

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

The action of the OK and Cancel buttons, this method ends the Page Layout panel's modal run. If the OK button inspired this method, the height, width, and scale entries must be acceptable (they must hold positive numbers), otherwise the unacceptable entry is selected and the panel isn't stopped. If the panel is being cancelled, then it's stopped regardless of the entries' acceptability. Note that this method is implementation-dependent, and not part of the OpenStep specification.

pickedOrientation:

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

Updates the panel with the selected orientation. This method is performed when the user selects a page orientation from the Portrait/Landscape matrix. This method updates the width and height, and redraws the paper view. You can get the new orientation by sending the message

```
int orientation = [sender selectedColumn]
```

and comparing the returned value to `NSLandscapeOrientation` and `NSPortraitOrientation`. Note that this method is implementation-dependent, and not part of the OpenStep specification.

pickedPaperSize:

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

Performed when the user selects a paper size from the Paper Size list. This method updates the page-layout width and height, redraws the paper view, and may switch the Portrait/Landscape orientation. Note that this method is implementation-dependent, and not part of the OpenStep specification.

pickedUnits:

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

Performed when the user selects a new unit of measurement from the Units list. The height and width are updated. Controls in the accessory view that express dimensions on the page must be converted to the new unit of measurement. The ratios returned by `convertOldFactor:newFactor:` method should be used to calculate the new values, as shown below. In the example, a hypothetical `NSPageLayout` subclass uses an `NSTextField` (`myField`) to display a value measured in the chosen units:

```
- pickedUnits:sender
{
    float old, new;

    /* At this point the units have been selected */
    /* but not set. Get the conversion factors. */
    [self convertOldFactor:&old newFactor:&new];

    /* Set myField based on the conversion factors. */
    [myField setFloatValue:([myField floatValue] * new / old)];

    /* Set the selected units. */
    return [super pickedUnits:sender];
}
```

The `NSTextField` object inherits `floatValue` from `NSControl`. Note that this method is implementation-dependent, and not part of the OpenStep specification. See also `convertOldFactor:newFactor:`.

printInfo

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

Returns the `NSPrintInfo` object used when the Print panel is run.

readPrintInfo

```
- (void)readPrintInfo
```

Reads the NSPageLayout's values from the NSPrintInfo object. This method is invoked from the runModal method; you shouldn't need to invoke it.

runModal

```
- (int)runModal
```

Reads the pertinent data from the associated NSPrintInfo object into the NSPageLayout object, and then runs the Page Layout panel in a modal loop. When the user clicks the Cancel or OK button the loop is broken from within the pickedButton: method, the panel is hidden, and, if the button was OK, the new NSPageLayout values are written to the NSPrintInfo object. This method returns the tag of the button that the user clicked to dismiss the panel (either NSOKButton or NSCancelButton).

This method is invoked by NSApplication's runPageLayout method; an application is best served by running the Page Layout panel from that method rather than invoking this one directly. See also runModalWithPrintInfo:.

runModalWithPrintInfo:

```
- (int)runModalWithPrintInfo:(NSPrintInfo *)pInfo
```

Displays the panel and begins its event loop. The panel's values are recorded in the pInfo. This method returns the tag of the button that the user clicked to dismiss the panel (either NSOKButton or NSCancelButton). See also runModal.

setAccessoryView:

```
- (void)setAccessoryView:(NSView *)aView
```

Adds aView to the page-layout's view hierarchy. Applications can invoke this method to add a view that contains their own controls. The panel is automatically resized to accommodate aView. This method can be invoked repeatedly to change the accessory view depending on the situation. If aView is nil, the panel's current accessory view, if any, is removed. See also accessoryView.

writePrintInfo

```
- (void)writePrintInfo
```

Writes the settings of the Page Layout panel to the `NSApplication` object's global `NSPrintInfo` object. This method is invoked when the user quits the Page Layout panel by clicking the OK button. See also `readPrintInfo`.

NSPanel

Inherits From:	NSWindow : NSResponder : NSObject
Conforms To:	NSCoding (NSResponder) NSObject (NSObject)
Declared In:	AppKit/NSPanel.h

Class Description

The `NSPanel` class defines objects that manage the panels of the OpenStep user interface. A panel is a window that serves an auxiliary function within an application. It generally displays controls that the user can act on to give instructions to the application or to modify the contents of a standard window.

Panels behave differently from standard windows in only a small number of ways, but the ways are important to the user interface:

- Panels can assume key window—but not main window—status. The key window receives keyboard events. The main window is the primary focus of user actions; it might contain the document the user is working on, for example.
- On-screen panels are normally removed from the screen list when the user begins to work in another application, and are restored to the screen when the user returns to the panel's application.

To aid in their auxiliary role, panels can be assigned special behaviors:

- A panel can be precluded from becoming the key window until the user makes a selection (makes some view in the panel the first responder) indicating an intention to begin typing. This prevents key window status from shifting to the panel unnecessarily.

- Palettes and similar panels can be made to float above standard windows and other panels. This prevents them from being covered and keeps them readily available to the user.
- A panel can be made to work—to receive mouse and keyboard events—even when there's an attention panel on-screen. This permits actions within the panel to affect the attention panel.

Method Types

Activity	Class Method
Determining the Panel Behavior	– becomesKeyOnlyIfNeeded – isFloatingPanel – setBecomesKeyOnlyIfNeeded: – setFloatingPanel: – setWorksWhenModal: – worksWhenModal

Instance Methods

`becomesKeyOnlyIfNeeded`

`- (BOOL)becomesKeyOnlyIfNeeded`

Returns whether the receiver waits until the user clicks within a view that can become the first responder to become the key window. See also `setBecomesKeyOnlyIfNeeded:`.

`isFloatingPanel`

`- (BOOL)isFloatingPanel`

Returns `YES` if the receiving panel floats above other windows, and `NO` otherwise. See also `setFloatingPanel:`.

`setBecomesKeyOnlyIfNeeded:`

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

Sets whether the NSPanel becomes the key window only when the user makes a selection (causing one of its NSViews to become the first responder). Since this requires the user to perform an extra action (clicking in the NSView) before being able to type within the window, it's appropriate only for NSPanels that don't normally require text entry. You should consider setting this attribute only if (1) most of the controls within the NSPanel are not text fields, and (2) the choices that can be made by entering text can also be made in another way, or are only incidental to the way the panel is normally used. See also becomesKeyOnlyIfNeeded.

setFloatingPanel:

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

Sets whether the receiving panel floats above other windows (that is, assigned to a window tier above standard windows). The default is NO. It's appropriate for an NSPanel to float above other windows only if:

- It's oriented to the mouse rather than the keyboard—that is, it doesn't become the key window or becomes the key window only if needed.
- It needs to remain visible while the user works in the application's standard windows—for example, if the user must frequently move the cursor back and forth between a standard window and the panel (such as a tool palette) or the panel gives information relevant to the user's actions within a standard window.
- It's small enough not to obscure much of what's behind it.
- It doesn't remain on-screen when the application is deactivated.

All four of these conditions should be true for flag to be set to YES. See also isFloatingPanel.

setWorksWhenModal:

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

Sets whether the NSPanel remains enabled to receive events, and possibly become the key window, even when a modal panel (attention panel) is on-screen. This is appropriate only for an NSPanel that needs to operate on attention panels. The default is NO. See worksWhenModal.

worksWhenModal

```
- (BOOL)worksWhenModal
```

Returns whether the `NSPanel` can receive keyboard and mouse events and possibly become the key window, even when a modal panel (attention panel) is on-screen. The default is `NO`. See also `setWorksWhenModal:`.

NSPasteboard

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

Class Description

`NSPasteboard` objects transfer data to and from the pasteboard server. The server is shared by all running applications. It contains data that the user has cut or copied and may paste, as well as other data that one application wants to transfer to another. `NSPasteboard` objects are an application's sole interface to the server and to all pasteboard operations.

Named Pasteboards

Data in the pasteboard server is associated with a name that indicates how it's to be used. Each set of data and its associated name is, in effect, a separate pasteboard, distinct from the others. An application keeps a separate `NSPasteboard` object for each named pasteboard that it uses. There are five standard pasteboards in common use:

Table 1-15 Standard Pasteboards

Pasteboard Type	Description
General pasteboard	Pasteboard that's used for ordinary cut, copy, and paste operations. It holds the contents of the last selection that's been cut or copied.
Font pasteboard	Pasteboard that holds font and character information and supports the Copy Font and Paste Font commands.
Ruler pasteboard	Pasteboard that holds information about paragraph formats in support of the Copy Ruler and Paste Ruler commands.
Find pasteboard	Pasteboard that holds information about the current state of the active application's Find panel. This information permits users to enter a search string into the Find panel, then switch to another application to conduct the search.
Drag pasteboard	Pasteboard that stores data to be manipulated as the result of a drag operation.

Each standard pasteboard is identified by a unique name (stored in global string objects):

- `NSGeneralPboard`
- `NSFontPboard`
- `NSRulerPboard`
- `NSFindPboard`
- `NSDragPboard`

You can create private pasteboards by asking for an `NSPasteboard` object with any name other than those listed above. The name of a private pasteboard can be passed to other applications to allow them to share the data it holds.

The `NSPasteboard` class makes sure there's never more than one object for each named pasteboard. If you ask for a new object when one has already been created for the pasteboard with that name, the existing object will be returned to you.

Data Types

Data can be placed in the pasteboard server in more than one representation. For example, an image might be provided both in Tag Image File Format (TIFF) and as encapsulated PostScript (EPS) code. Multiple representations give pasting applications the option of choosing which data type to use. In general, an application taking data from the pasteboard should choose the richest representation it can handle—rich text over plain ASCII, for example. An application putting data in the pasteboard should promise to supply it in as many data types as possible, so that as many applications as possible can make use of it.

Data types are identified by string objects containing the full type name. These global variables identify the string objects for the standard pasteboard types:

Table 1-16 Pasteboard Data Types

Data Type	Description
NSStringPboardType	NSString data
NSPostScriptPboardType	Encapsulated PostScript (EPS) code
NSTIFFPboardType	Tag Image File Format (TIFF)
NSRTFPboardType	Rich Text Format (RTF)
NSFilenamesPboardType	NSArray containing NSString filenames
NSTabularTextPboardType	Tab-separated fields of ASCII text
NSFontPboardType	Font and character information
NSRulerPboardType	Paragraph formatting information
NSFileContentsPboardType	A representation of a file's contents
NSColorPboardType	NSColor data
NSSelectionPboardType	Describes a selection
NSDataLinkPboardType	Defines a link between documents

Types other than those listed can also be used. For example, your application may keep data in a private format that's richer than any of the types listed above. That format can also be used as a pasteboard type.

Reading and Writing Data

Typically, data is written to the pasteboard using `setData:forType:` and read using `dataForType:`. However, data of the type `NSFileContentsPboardType`, representing the contents of a named file, must be written to the `NSPasteboard` object using `writeFileContents:` and copied from the object to a file using `readFileContentsType:toFile:`.

Errors

Except where errors are specifically mentioned in the method descriptions, any communications error with the pasteboard server raises `NSPasteboardCommunicationException`.

Method Types

Activity	Class Method
Creating and releasing an NSPasteboard object	+ generalPasteboard + pasteboardWithName: + pasteboardWithUniqueName – releaseGlobally
Getting data in different formats	+ pasteboardByFilteringData:ofType: + pasteboardByFilteringFile: + pasteboardByFilteringTypesInPasteboard: + typesFilterableTo:
Referring to a pasteboard by name	– name
Writing data	– addTypes:owner: – declareTypes:owner: – setData:forType: – setPropertyList:forType: – setString:forType: – writeFileContents:
Determining types	– availableTypeFromArray: – types
Reading data	– changeCount – dataForType: – propertyListForType: – readFileContentsType:toFile: – stringForType:
Methods implemented by the owner	– pasteboard:provideDataForType: – pasteboardChangedOwner:

Class Methods

generalPasteboard

```
+ (NSPasteboard *)generalPasteboard
```

Returns the general `NSPasteboard`. See also `pasteboardWithName:`, `pasteboardWithUniqueName`, `releaseGlobally`.

pasteboardByFilteringData:ofType:

```
+ (NSPasteboard *)pasteboardByFilteringData:(NSData *)data
    ofType:(NSString *)type
```

Creates and returns a new `NSPasteboard` with a unique name that has, declared within it, data of every type that can be provided by the available filter services from `data`. The returned pasteboard also declares data of the supplied `type`. No filter service is invoked until the data is actually requested, so invoking this method is reasonably inexpensive. See also `pasteboardByFilteringFile:`, `pasteboardByFilteringTypesInPasteboard:`.

pasteboardByFilteringFile:

```
+ (NSPasteboard *)pasteboardByFilteringFile:(NSString *)filename
```

Creates and returns a new `NSPasteboard` with a unique name that has, declared within it, data of every type that can be provided by the available filter services from the file `filename`. No filter service is invoked until the data is actually requested, so invoking this method is reasonably inexpensive. See also `pasteboardByFilteringData:ofType:`, `pasteboardByFilteringTypesInPasteboard:`.

pasteboardByFilteringTypesInPasteboard:

```
+ (NSPasteboard *)pasteboardByFilteringTypesInPasteboard:
    (NSPasteboard *)pboard
```

Creates and returns a new `NSPasteboard` with a unique name that has, declared within it, data of every type that can be provided by the available filter services from the data on pasteboard `pboard`. This process can be thought of as expanding the pasteboard, since the new pasteboard generally will contain more representations of the data on `pboard`.

This method returns `pboard` if it is a pasteboard returned by one of the `pasteboardByFiltering...` methods, so a pasteboard can't be expanded multiple times. This method only returns the original types and the types that can be created as a result of a single filter; the pasteboard will not have defined types that are the result of translation by multiple filters. No filter service is

invoked until the data is actually requested, so invoking this method is reasonably inexpensive. See also `pasteboardByFilteringData:ofType:`, `pasteboardByFilteringFile:`.

pasteboardWithName:

```
+ (NSPasteboard *)pasteboardWithName:(NSString *)name
```

Returns the `NSPasteboard` object for the `name` pasteboard. A new object is created only if the application doesn't yet have an `NSPasteboard` object for the specified `name`; otherwise, the existing `NSPasteboard` is returned. To get a standard pasteboard, `name` should be one of the following variables:

- `NSGeneralPboard`
- `NSFontPboard`
- `NSRulerPboard`
- `NSFindPboard`
- `NSDragPboard`

Other names can be assigned to create private pasteboards. See also the "Pasteboard" section of the "Types and Constants" chapter.

pasteboardWithUniqueName

```
+ (NSPasteboard *)pasteboardWithUniqueName
```

Creates and returns a new `NSPasteboard` with a name that is guaranteed to be unique with respect to other `NSPasteboard`s on the system. This method is useful for applications that implement their own interprocess communication using pasteboards. See also `generalPasteboard`, `pasteboardWithName:`.

typesFilterableTo:

```
+ (NSArray *)typesFilterableTo:(NSString *)type
```

Returns an array indicating the types that `type` can be converted to by available filters. The array contains the original type. The caller is responsible for freeing the returned array.

Instance Methods

addTypes:owner:

```
- (int)addTypes:(NSArray *)newTypes owner:(id)newOwner
```

Adds `newTypes` to the `NSPasteboard` and declares a `newOwner`. Returns the new change count or 0 in case of error. This method can be useful when multiple entities (such as a combination of application and library methods) contribute data for a single copy command. It should only be invoked after a `declareTypes:owner:` message has been sent for the same data. The owner for the new types may be different from the owner(s) of the previously declared data.

availableTypeFromArray:

```
- (NSString *)availableTypeFromArray:(NSArray *)types
```

Scans the array of `types` and returns the first type that matches a type declared on the pasteboard. A `types` or `availableTypeFromArray:` message should be sent before reading any data from the pasteboard.

changeCount

```
- (int)changeCount
```

Returns the `NSPasteboard`'s current change count. The change count is a system-wide variable that increments every time the contents of the pasteboard changes (a new owner is declared). By examining the change count, an application can determine whether the current data in the pasteboard is the same as the data the application last received. An independent change count is maintained for each named pasteboard.

dataForType:

```
- (NSData *)dataForType:(NSString *)dataType
```

Returns `NSPasteboard` data using the type specified by `dataType`. `dataType` should be one of the types returned by the types method. This method returns `nil` if the contents of the pasteboard have changed since last

checked with the types method, or the pasteboard server cannot supply the data. If `nil` is returned, the application should display a panel informing the user that it was unable to carry out a paste operation.

declareTypes:owner:

```
- (int)declareTypes:(NSArray *)newTypes owner:(id)newOwner
```

Prepares the pasteboard for a change in its contents by declaring the `newTypes` of data it will contain, and a `newOwner`. This is the first step in responding to a user's copy or cut command, and must precede the messages that actually write the data. A `declareTypes:owner:` message is tantamount to changing the contents of the pasteboard. This method invalidates the current contents of the pasteboard, and increments and returns the pasteboard's change count.

`newTypes` contains an array of strings that name the new types any new contents of the pasteboard may assume. The types should be ordered according to the preference of the source application, with the most preferred type coming first (typically, the richest representation is first).

The `newOwner` is the object responsible for writing data to the pasteboard in all the types listed in `newTypes`. Data is written using the `setData:forType:` method. You can write the data immediately after declaring the types, or wait until it's required for a paste operation. If you wait, the owner will receive a `pasteboard:provideDataForType:` message requesting the data in a particular type when it's needed. You might choose to write data immediately for the most preferred type, but wait for the others to see whether they'll be requested.

The `newOwner` can be `NULL` if data is provided for all types immediately. Otherwise, the owner should be an object that won't be freed. It should not, for example, be the `NSView` that displays the data if that `NSView` is in a window that might be closed.

name

```
- (NSString *)name
```

Returns the pasteboard's name.

pasteboard:provideDataForType:

```
- (void)pasteboard:(NSPasteboard *)sender
    provideDataForType:(NSString *)type
```

Implemented by the owner (previously declared in a `declareTypes:owner:` message) to provide promised data. The owner receives a `pasteboard:provideDataForType:` message from the `sender` pasteboard when the data is required for a paste operation. `type` gives the type of data being requested. The requested data should be written to `sender` using the `setData:forType:` method.

`pasteboard:provideDataForType:` messages may also be sent to the owner when the application is shut down through `NSApplication`'s `terminate:` method. This method is invoked in response to a Quit command. Thus the user can copy something to the pasteboard, quit the application, and still paste the data that was copied.

A `pasteboard:provideDataForType:` message is sent only if `type` data hasn't already been supplied. Instead of writing all data types when the cut or copy operation is done, an application can choose to implement this method to provide the data for certain types only when they're requested.

If an application writes data to the pasteboard in the richest, and therefore most preferred, type at the time of a cut or copy operation, its `pasteboard:provideDataForType:` method can read the pasteboard data , convert it to the requested type, and write it back to the pasteboard as the new type.

pasteboardChangedOwner:

```
- (void)pasteboardChangedOwner:(NSPasteboard *)sender
```

Notifies a prior owner of the `sender` pasteboard (and owners of representations on the pasteboard) that the pasteboard has changed owners. This method is optional and need only be implemented by pasteboard owners that need to know when they have lost ownership. The owner is not able to read the contents of the pasteboard when responding to this method. The owner should be prepared to receive this method at any time, even from within the `declareTypes:owner:` used to declare ownership.

propertyListForType:

```
- (id)propertyListForType:(NSString *)dataType
```

Returns a property list object using the type specified by `dataType`. See also `setPropertyList:forType:`.

readFileContentsType:toFile:

```
- (NSString *)readFileContentsType:(NSString *)type
    toFile:(NSString *)filename
```

Reads data representing a file's contents from the pasteboard, and writes it to the file `filename`. Data of any file contents type should only be read using this method. `type` should generally be specified; if `type` is `NULL`, a type based on `filename`'s extension (as returned by `NSCreateFileContentsPboardType()`) is substituted. If data matching `type` isn't found on the pasteboard, data of type `NSFileContentsPboardType` is requested. Returns an allocated string with the name of the file that the data was actually written to. You should send the `types` or `availableTypeFromArray:` message before reading any data from the pasteboard. See also `writeFileContents:`.

releaseGlobally

```
- (void)releaseGlobally
```

Causes all server resources to be freed when the receiving pasteboard object is deallocated.

setData:forType:

```
- (BOOL)setData:(NSData *)data forType:(NSString *)dataType
```

Writes data of type `dataType` to the pasteboard server from `data`. Returns `YES` if the data is successfully written; otherwise returns `NO`.

setPropertyList:forType:

```
- (BOOL)setPropertyList:(id)propertyList
    forType:(NSString *)dataType
```

Writes data of type dataType to the pasteboard server from propertyList (see NSSerializer in the Foundation Kit). Returns YES if the data is successfully written; otherwise returns NO.

setString:forType:

```
- (BOOL)setString:(NSString *)string forType:(NSString *)dataType
```

Writes data of type dataType to the pasteboard server from string. Returns YES if the data is successfully written; otherwise returns NO. See also stringForType:.

stringForType:

```
- (NSString *)stringForType:(NSString *)dataType
```

Returns an NSString using the type specified by dataType. See also setString:forType:.

types

```
- (NSArray *)types
```

Returns an array containing the NSPasteboard's data types. Types are listed in the same order that they were declared. A types or availableTypeFromArray: message should be sent before reading any data from the pasteboard. See the Class Description for a list of pasteboard types.

writeFileContents:

```
- (BOOL)writeFileContents:(NSString *)filename
```

Writes data from filename to the pasteboard server, and declares the data to be of type NSFileContentsPboardType and also of a type appropriate for the file's extension (as returned by NSCreateFileContentsPboardType() when passed the files extention) if it has an extension. Returns YES if the data from filename was successfully written to the pasteboard, and NO otherwise.

NSPopUpButton

Inherits From:	NSButton : NSControl : NSView : NSResponder : NSObject
Conforms To:	NSCoding (NSResponder) NSObject (NSObject)
Declared In:	AppKit/NSPopUpButton.h

Class Description

The `NSPopUpButton` class defines objects that implement the pop-up and pull-down lists of the OpenStep graphical user interface. When configured to display a pop-up list, an `NSPopUpButton` contains a number of options and displays as its title the option that was last selected. A pop-up list is often used for selecting items from a small to medium-sized set of options (like the zoom factor for a document window). It's a useful alternative to a matrix of radio buttons or an `NSBrowser` when screen space is at a premium; a zoom factor pop-up can easily fit next to a scroll bar at the bottom of a window, for example.

When configured to display a pull-down list, an `NSPopUpButton` is generally used for selecting commands in a very specific context. You can think of a pull-down list as a compact form of menu. A pull-down list's title isn't affected by the user's actions, and a pull-down list always displays a title that identifies the type of commands it contains. When the commands only make sense in the context of a particular display, a pull-down list can be used in that display to keep the related actions nearby, and to keep them out of the way when that display isn't visible.

Method Types

Activity	Class Method
Initializing an NSPopUpButton	– initWithFrame:pullsDown:
Target and action	– action – setAction: – setTarget: – target
Adding items	– addItemWithTitle: – addItemsWithTitles: – insertItemWithTitle:atIndex:
Removing items	– removeAllItems – removeItemWithTitle: – removeItemAtIndex:
Querying the NSPopUpButton about its items	– indexOfItemWithTitle: – indexOfSelectedItem – numberOfItems – itemArray – itemAtIndex: – itemMatrix – itemTitleAtIndex: – itemTitles – itemWithTitle: – lastItem – selectedItem – titleOfSelectedItem
Manipulating the NSPopUpButton	– font – pullsDown – selectItemAtIndex: – selectItemWithTitle: – setFont: – setPullsDown: – setTitle: – stringValue – synchronizeTitleAndSelectedItem
Displaying the NSPopUpButton's items	– autoenablesItems – setAutoenablesItems:

Instance Methods

action

```
- (SEL)action
```

Returns the NSPopUpButton's action method. See also `setAction:`.

addItemWithTitle:

```
- (void)addItemWithTitle:(NSString *)title
```

Adds an item with the name `title` to the bottom of the item list. See also `addItemsWithTitles:`, `insertItemWithTitle:atIndex:`, `removeItemWithTitle:`.

addItemsWithTitles:

```
- (void)addItemsWithTitles:(NSArray *)itemTitles
```

Adds multiple items to the end of the item list. The titles for the new items are taken from the `itemTitles` array. See also `addItemWithTitle:`.

autoenablesItems

```
- (BOOL)autoenablesItems
```

Returns whether the `NSPopUpButton` enables and disables its items. See the `NSMenuActionResponder` informal protocol for more information. See also `setAutoenablesItems:`.

font

```
- (NSFont *)font
```

Returns the font used to draw the items. See also `setFont:`.

indexOfItemWithTitle:

```
- (int)indexOfItemWithTitle:(NSString *)title
```

Returns the index of the item whose title matches `title`, or –1 if no match is found. See also `indexOfSelectedItem`, `itemAtIndex:`, `itemTitles`, `itemWithTitle:`, `titleOfSelectedItem`.

indexOfSelectedItem

```
- (int)indexOfSelectedItem
```

Returns the index of the item last selected by the user, or –1 if there's no selected item. See also `indexOfItemWithTitle:`.

initWithFrame:pullsDown:

```
- (id)initWithFrame:(NSRect)frameRect pullsDown:(BOOL)flag
```

Initializes and returns a newly allocated `NSPopUpButton`, giving it the frame specified by `frameRect`. If `flag` is `YES`, the receiver is initialized to operate as a pull-down list; otherwise, it operates as a pop-up list. This method is the designated initializer for `NSPopUpButton`. If you create an `NSPopUpButton` subclass that performs its own initialization, you must override this method.

insertItemWithTitle:atIndex:

```
- (void)insertItemWithTitle:(NSString *)title
    atIndex:(unsigned int)index
```

Inserts an item, with `title` as its title, at position `index`. The item at the top has an index of 0. If an item with a title of `title` already exists in the item list, it is removed and the new one is added at `index`. This essentially moves `title` to a new position, though if the item removed was at a position before `index`, the new item will actually be inserted at `index - 1`. If you want to move an item, it's better to invoke `removeItemWithTitle:` or `removeItemAtIndex:` explicitly, and then send this message.

itemArray

```
- (NSArray *)itemArray
```

Returns the `NSArray` that holds the receiver's menu items. See also `itemTitles`, `itemAtIndex:`.

itemAtIndex:

```
- (id <NSMenuItem>)itemAtIndex:(int)index
```

Returns the NSMenuItem for the item at index, or nil if no such item exists. See also itemTitleAtIndex:, indexOfItemWithTitle:.

itemMatrix

```
- (NSMatrix *)itemMatrix
```

Returns the NSMatrix that holds the receiver's items. Note that this method is not part of the OpenStep specification. See also itemArray.

itemTitleAtIndex:

```
- (NSString *)itemTitleAtIndex:(int)index
```

Returns the title of the item at index, or the empty string if no such item exists. See also itemTitles, indexOfItemWithTitle:.

itemTitles

```
- (NSArray *)itemTitles
```

Returns an NSArray that holds the titles of the receiver's items. See also numberOfItems, lastItem, selectedItem.

itemWithTitle:

```
- (id <NSMenuItem>)itemWithTitle:(NSString *)title
```

Returns the NSMenuItem for the item whose title is title, or nil if no such item exists. See also indexOfItemWithTitle:.

lastItem

```
- (id <NSMenuItem>)lastItem
```

Returns the NSMenuItem corresponding to the last item in the list. See also numberOfItems, selectedItem.

numberOfItems

```
- (int)numberOfItems
```

Returns the number of items in the receiver's item list. See also `lastItem`, `selectedItem`, `indexOfItemWithTitle:`.

pullsDown

```
- (BOOL)pullsDown
```

Returns `YES` if the receiver is configured as a pull-down list, and `NO` if it's configured as a pop-up list. See also `setPullsDown:`.

removeAllItems

```
- (void)removeAllItems
```

Removes all items in the receiver's item list. See also `removeItemWithTitle:`, `removeItemAtIndex:`.

removeItemWithTitle:

```
- (void)removeItemWithTitle:(NSString *)title
```

Removes the item whose title matches `title`. See also `removeAllItems`, `removeItemAtIndex:`.

removeItemAtIndex:

```
- (void)removeItemAtIndex:(int)index
```

Removes the item at the specified index. See also `removeAllItems`, `removeItemWithTitle:`.

selectItemAtIndex:

```
- (void)selectItemAtIndex:(int)index
```

Selects the item at `index` and invokes `synchronizeTitleAndSelectedItem`. See also `selectItemWithTitle:`.

selectItemWithTitle:

```
- (void)selectItemWithTitle:(NSString *)title
```

Selects the item whose title is `title` and invokes `synchronizeTitleAndSelectedItem`. See also `selectItemAtIndex:`.

selectedItem

```
- (id <NSMenuItem>)selectedItem
```

Returns the `NSMenuItem` for the selected item. See also `selectItemAtIndex:`, `selectItemWithTitle:`, `titleOfSelectedItem`, `indexOfSelectedItem`.

setAction:

```
- (void)setAction:(SEL)aSelector
```

Sets the `NSPopUpButton`'s action method to `aSelector`. The action message is actually sent by the `NSMatrix` containing the `NSMenuCell`s that make up the list items. See also `action`.

setAutoenablesItems:

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

Sets whether the `NSPopUpButton` enables and disables its items. See the `NSMenuActionResponder` informal protocol for more information. See also `autoenablesItems`.

setFont:

```
- (void)setFont:(NSFont *)fontObject
```

Sets the font used to draw the items. See also `font`.

setPullsDown:

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

If `flag` is `YES`, the receiver is configured as a pull-down list. If `flag` is `NO`, the receiver is configured as a pop-up list. See also `pullsDown`.

setTarget:

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

Sets the object to which an action will be sent when an item is selected from the NSPopUpButton's item list. The action is actually sent by the NSMatrix containing the NSMenuCells that make up the PopUpList. See also target.

setTitle:

```
- (void)setTitle:(NSString *)aString
```

Adds a new item (if the receiver doesn't already have an item titled aString), makes it the selected item, and invokes synchronizeTitleAndSelectedItem.

stringValue

```
- (NSString *)stringValue
```

Returns the title of the selected item. See also indexOfItemWithTitle:.

synchronizeTitleAndSelectedItem

```
- (void)synchronizeTitleAndSelectedItem
```

Ensures that the receiver's title agrees with the title of the selected item (see indexOfSelectedItem). If there's no selected item, this method selects the first item in the item list and sets the receiver's title to match. This method is useful in subclasses that directly select items in the item matrix or that override setTitle:.

target

```
- (id)target
```

Returns the object to which the action will be sent when an item is selected from the item list. The default value is nil, which causes the action message to be sent up the responder chain. The target is actually sent the action by the list's NSMatrix. See also setTarget:, action.

titleOfSelectedItem

```
- (NSString *)titleOfSelectedItem
```

Returns the title of the item last selected by the user, or the empty string if there's no such item. See also `selectedItem`.

NSPrinter

Inherits From:	NSObject
Conforms To:	NSCoding, NSCopying NSObject (NSObject)
Declared In:	AppKit/NSPrinter.h

Class Description

An `NSPrinter` object describes a printer's capabilities, such as whether the printer can print in color and whether it provides a particular font. An `NSPrinter` object represents either a particular make or type of printer, or an actual printer available to the computer.

There are two ways to create an `NSPrinter`:

- To create an abstract object that provides information about a type of printer rather than an object that represents an actual printer device, use the `printerWithType:` class method, passing a printer type (an `NSString`) as the argument. The `printerTypes` class method provides a list of the printer types recognized by the computer. Printer types are described in files written in PostScript Printer Description (PPD) format. The location of these files is platform dependent.
- To create or find an `NSPrinter` that corresponds to an actual printer device, use the `printerWithName:` class method, passing the name of a printer. The way you find out what the available printer names are depends on the platforms you are using.

Once you have an `NSPrinter`, there's only one thing you can do with it: Retrieve information regarding the type of printer or regarding the actual printer the object represents. You can't change the information in an `NSPrinter`, nor can you use an `NSPrinter` to initiate or control a printing job.

When you create an `NSPrinter` object, the object reads the file that corresponds to the type of printer you specified and stores the data it finds there in named tables. Printer types are described in files written in the PostScript Printer Description (PPD) format. Any piece of information in the PPD tables can be retrieved through the methods `stringForKey:inTable:` and `stringListForKey:inTable:`, as explained later. Commonly needed items, such as whether a printer uses color or what is the size of the page on which it prints, are available through more direct methods (methods such as `isColor` and `pageSizeForPaper:`).

Note – To understand what the `NSPrinter` tables contain, you need to be acquainted with the PPD file format. This is described in *PostScript Printer Description File Format Specification, version 4.0*, available from Adobe Systems Incorporated. The rest of this class description assumes a familiarity with the concepts and terminology presented in the Adobe manual. A brief summary of the PPD format is given in the following section; PPD terms defined in the Adobe manual are shown in italic.

PPD Format

A PPD file statement, or *entry*, associates a *value* with a *main keyword*:

```
*mainKeyword: value
```

The asterisk is literal; it indicates the beginning of a new entry.

For example:

```
*ModelName: "MMimeo Machine"
*3dDevice: False
```

A main keyword can be qualified by an *option keyword*:

```
*mainKeyword optionKeyword: value
```

For example:

```
*PaperDensity Letter: "0.1"
*PaperDensity Legal: "0.2"
*PaperDensity A4: "0.3"
*PaperDensity B5: "0.4"
```

In addition, any number of entries may have the same main keyword with no option keyword yet give different values:

```
*InkName: ProcessBlack/Process Black
*InkName: CustomColor/Custom Color
*InkName: ProcessCyan/Process Cyan
*InkName: ProcessMagenta/Process Magenta
*InkName: ProcessYellow/Process Yellow
```

Option keywords and values can sport *translation strings*. A translation string is a textual description, appropriate for display in a user interface, of the option or value. An option or value is separated from its translation string by a slash:

```
*Resolution 300dpi/300 dpi: " ... "
*InkName: ProcessBlack/Process Black
```

In the first example, the `300dpi` option would be presented in a user interface as "300 dpi." The second example assigns the string "Process Black" as the translation string for the `ProcessBlack` value.

`NSPrinter` treats entries that have an `*OrderDependency` or `*UIConstraint` main keyword specially. Such entries take the following forms (the bracketed elements are optional):

```
*OrderDependency: real section mainKeyword [optionKeyword]
*UIConstraint: mainKeyword1 [optionKeyword1]
               mainKeyword2 [optionKeyword2]
```

There may be more than one `UIConstraint` entry with the same `mainKeyword1` or `mainKeyword1`/`optionKeyword1` value. Following are some examples of `*OrderDependency` and `*UIConstraint` entries:

```
*OrderDependency: 10 AnySetup *Resolution
*UIConstraint: *Option3 None *PageSize Legal
*UIConstraint: *Option3 None *PageRegion Legal
```

Explaining these entries is beyond the scope of this documentation; however, it's important to note their forms in order to understand how they're represented in the `NSPrinter` tables.

NSPrinter Tables

NSPrinter defines five key-value tables to store PPD information. The tables are identified by the names given below:

Table 1-17 NSPrinter Key-Value Tables

Table Name	Contents
PPD	General information about a printer type. This table contains the values for all entries in a PPD file except those with the *OrderDependency and *UIConstraint main keywords. The values in this table don't include the translation strings.
PPDOptionTranslation	Option keyword translation strings.
PPDArgumentTranslation	Value translation strings.
PPDOrderDependency	*OrderDependency values.
PPDUIConstraints	*UIConstraint values.

There are two principle methods for retrieving data from the `NSPrinter` tables:

- `stringForKey:inTable:` returns the value for the first occurrence of a given key in the given table.
- `stringListForKey:inTable:` returns an array of values, one for each occurrence of the key.

For both methods, the first argument is an `NSString` that names a key. Which part of a PPD file entry the key corresponds to depends on the table as explained in the following sections. The second argument names the table that you want to look in. The values that are returned by these methods, whether singular or in an array, are always `NSString`s, even if the value wasn't a quoted string in the PPD file.

The `NSPrinter` tables store data as ASCII text; thus the two methods described above are sufficient for retrieving any value from any table. `NSPrinter` provides a number of other methods, such as `booleanForKey:inTable:` and `intForKey:inTable:`, that retrieve single values and coerce them, if possible, into particular data types. The coercion doesn't affect the data that's stored in the table (it remains in ASCII format).

To check the integrity of a table, use the `isKey:forTable:` and `statusForTable:` methods. The former returns a Boolean that indicates whether the given key is valid for the given table; the latter returns an error code that describes the general state of a table (in particular, whether it actually exists).

Retrieving Values from the PPD Table

Keys for the PPD table are strings that name a main keyword or main keyword/option keyword pairing (formatted as "*mainKeyword*/*optionKeyword*"). In both cases, you exclude the main keyword asterisk. The following example creates an `NSPrinter` and invokes `stringForKey:inTable:` to retrieve the value for an un-optioned main keyword:

```
/* Create an NSPrinter object for a printer type. */
NSPrinter *prType = [NSPrinter
                     printerWithType:@"My_Mimeo_Machine"]

 NSString *sValue = [prType stringForKey:@"3dDevice"
                       inTable:@"PPD"];
/* sValue is "False". */
```

To retrieve the value for a main keyword/option keyword pair, pass the keywords formatted as "*mainKeyword*/*optionKeyword*":

```
NSString *sValue = [prType stringForKey:@"PaperDensity/A4"
                           inTable:@"PPD"];
/* sValue is "0.3". */
```

`stringForKey:inTable:` can determine if a main keyword has options. If you pass a main keyword (only) as the first argument to the method, and if that keyword has options in the PPD file, the method returns the empty string. If it doesn't have options, it returns the value of the first occurrence of the main keyword:

```
NSString *sValue = [prType stringForKey:@"PaperDensity"
inTable:@"PPD"];
/* sValue is empty string*/

NSString *sValue = [prType stringForKey:@"InkName" inTable:@"PPD"];
/* sValue is "ProcessBlack" */
```

To retrieve the values for all occurrences of an un-optioned main keyword, use the `stringListForKey:inTable:` method:

```
NSArray *sList = [prType stringListForKey:@"InkName"
                            inTable:@"PPD"];
/* [slist objectAtIndex:0] is "ProcessBlack",
   [slist objectAtIndex:1] is "CustomColor",
   [slist objectAtIndex:2] is "ProcessCyan", and so on. */
```

In addition, `stringListForKey:inTable:` can be used to retrieve all the options for a main keyword given that the main keyword has options:

```
NSArray *sList = [prType stringListForKey:@"PaperDensity"
                            inTable:@"PPD"];
/* [slist objectAtIndex:0] is "Letter",
   [slist objectAtIndex:1] is "Legal",
   [slist objectAtIndex:2] is "A4", and so on. */
```

Retrieving Values from the Option and Argument Translation Tables

A key to a translation table is like that to the PPD table: It's a main keyword or main/option keyword pair (again excluding the asterisk). However, the values that are returned from the translation tables are the translation strings for the option or argument (value) portions of the PPD file entry. For example:

```
NSString *sValue = [prType stringForKey:@"Resolution/300dpi"
                            inTable:@"PPDOptionTranslation"];
/* sValue is "300 dpi". */

NSArray *sList = [prType stringListForKey:@"InkName"
                            inTable:@"PPDArgumentTranslation"];
/* [slist objectAtIndex:0] is "Process Black",
   [slist objectAtIndex:1] is "Custom Color",
   [slist objectAtIndex:2] is "Process Cyan", and so on. */
```

As with the PPD table, requesting an `NSArray` of `NSString`s for an un-optioned main keyword returns the keyword's options (if it has any).

Retrieving Values from the Order Dependency Table

As mentioned earlier, an order dependency entry takes this form:

```
*OrderDependency: real section mainKeyword [optionKeyword]
```

These entries are stored in the `PPDOrderDependency` table. To retrieve a value from this table, always use `stringListForKey:inTable:`. The value passed as the key is, again, a main keyword or main keyword/option keyword

pair; however, these values correspond to the mainKeyword and optionKeyword parts of an order dependency entry's value. As with the other tables, the main keyword's asterisk is excluded. The method returns an NSArray of two NSStrings that correspond to the real and section values for the entry. For example:

```
NSArray *sList = [prType stringListForKey:@"Resolution"
                            inTable:@"PPDOrderDependency"]
/* [slist objectAtIndex:0] = "10",
   [slist objectAtIndex:1] = "AnySetup" */
```

Retrieving Values from the UIConstraints Table

Retrieving a value from the PPDUIConstraints table is similar to retrieving a value from the PPDOrderDependency table: always use stringListForKey:inTable: and the key corresponds to elements in the entry's value. Given the following form (as described earlier), the key corresponds to mainKeyword1/optionKeyword1:

```
*UIConstraint: mainKeyword1 [optionKeyword1] mainKeyword2
[optionKeyword2]
```

The NSArray that's returned by stringListForKey:inTable: contains the mainKeyword2 and optionKeyword2 values (with the keywords stored as separate elements in the NSArray) for every *UIConstraints entry that has the given mainKeyword1/optionKeyword1 value. For example:

```
NSArray *sList = [prType stringListForKey:@"Option3/None"
                            inTable:@"PPDUIConstraints"]
/* [slist objectAtIndex:0] = "PageSize",
   [slist objectAtIndex:1] = "Legal",
   [slist objectAtIndex:2] = "PageRegion",
   [slist objectAtIndex:3] = "Legal" */
```

Note that the main keywords that are returned in the NSArray don't have asterisks. Also, the NSArray that's returned always alternates main and option keywords. If a particular main keyword doesn't have an option associated with it, the string for the option will be empty but the entry in the NSArray for the option *will* exist.

Method Types

Activity	Class Method
Finding an NSPrinter	+ printerWithName: + printerWithType: + printerNames + printerTypes
Printer attributes	– host – name – note – type
Retrieving specific information	– acceptsBinary – imageRectForPaper: – pageSizeForPaper: – isColor – isFontAvailable: – languageLevel – isOutputStackInReverseOrder
Querying the NSPrinter tables	– booleanForKey:inTable: – deviceDescription – floatForKey:inTable: – intForKey:inTable: – rectForKey:inTable: – sizeForKey:inTable: – stringForKey:inTable: – stringListForKey:inTable: – statusForTable: – isKey:inTable:

Class Methods

printerNames

```
+ (NSArray *)printerNames
```

Returns the printer names (configured printers) that are available. See also `printerWithName:`, `printerTypes`.

printerTypes

```
+ (NSArray *)printerTypes
```

Returns strings containing the names of the recognized printer types. A printer type is represented by a PPD file (extension `.ppd`). This method searches for normal PPD files directly, or in bundles, in the following directories:

/SunLibrary/PrinterTypes
~/Library/PrinterTypes
/HostLibrary/PrinterTypes
/LocalLibrary/PrinterTypes

Custom PPD files are searched for in the `CustomPrinters` subdirectory (or bundles therein) in each of the above. See also `printerWithType:`, `printerNames`.

printerWithName:

```
+ (NSPrinter *)printerWithName:(NSString *)name
```

Returns the printer with the given name. See also `printerNames`, `printerWithType:`.

printerWithType:

```
+ (NSPrinter *)printerWithType:(NSString *)type
```

Returns an `NSPrinter` object for the given printer `type`; the returned object doesn't correspond to an actual printer. The `type` argument should be an element in the array returned by `printerTypes:`. See also `printerWithName:`, `printerTypes`.

Instance Methods

acceptsBinary

```
- (BOOL)acceptsBinary
```

Returns `YES` if the printer accepts binary PostScript data, otherwise returns `NO`.

booleanForKey:inTable:

```
- (BOOL)booleanForKey:(NSString *)key inTable:(NSString *)table
```

Returns a boolean value for the given `key` in the given `table`: `YES` is returned if the value, which is stored as ASCII text, is "YES", "TRUE", or names a non-negative integer. Otherwise, this method returns `NO`. `key` should be formed as described in the `NSPrinter` class description, given previously. See also `stringForKey:inTable:`.

deviceDescription

```
- (NSDictionary *)deviceDescription
```

Returns a dictionary of keys and values describing the device. See `NSGraphics.h` for possible keys. See also `NSDictionary`.

floatForKey:inTable:

```
- (float)floatForKey:(NSString *)key inTable:(NSString *)table
```

Returns a floating-point value for the given `key` in the given `table`. Returns 0.0 if the value, which is stored as ASCII text, can't be coerced to a `float`. `key` should be formed as described in the `NSPrinter` Class Description, given previously. See also `sizeForKey:inTable:`, `floatForKey:inTable:`.

host

```
- (NSString *)host
```

Returns the name of the printer's host computer.

imageRectForPaper:

```
- (NSRect)imageRectForPaper:(NSString *)paperName
```

Returns the printing rectangle (the area of the page that's available for printing) for the named paper type. Possible values for `paperName` are contained in the printer's PPD file. Typical values are Letter and Legal.

intForKey:inTable:

```
- (int)intForKey:(NSString *)key inTable:(NSString *)table
```

Returns an integer value for the given `key` in the given `table`. Returns 0 if the value, which is stored as ASCII text, and can't be coerced to an `int`. `key` should be formed as described in the `NSPrinter` Class Description, above. See also `floatForKey:inTable:`, `stringForKey:inTable:`.

isColor

```
- (BOOL)isColor
```

Returns `YES` if this `NSPrinter` can print in color. Otherwise returns `NO`.

isFontAvailable:

```
- (BOOL)isFontAvailable:(NSString *)fontName
```

Returns `YES` if the named font is available to the `NSPrinter`. Otherwise returns `NO`. Font names are formed as in an invocation of `NSFont`'s `useFont:` method; examples include "Helvetica-Bold", "Times-Roman", and "Courier-BoldOblique".

isKey:inTable:

```
- (BOOL)isKey:(NSString *)key inTable:(NSString *)table
```

Returns `YES` if `key` is a key in `table` which must name one of the printer tables listed in the `NSPrinter` "Class Description".

isOutputStackInReverseOrder

```
- (BOOL)isOutputStackInReverseOrder
```

Returns `YES` if the printer prints pages in reverse page order, otherwise returns `NO`. By being printed in reverse order, the pages in the resulting output stack will be in the correct (first-to-last) order (assuming that the printer produces pages face-up).

languageLevel

```
- (int)languageLevel
```

Returns the PostScript language level (either 1 or 2) recognized by the printer.

name

```
- (NSString *)name
```

Returns the NSPrinter's name. If an actual printer isn't represented, a pointer to NULL is returned. See also printerWithName:.

note

```
- (NSString *)note
```

Returns the comment that's associated with the NSPrinter. If the object doesn't represent an actual printer, this method returns a pointer to NULL.

pageSizeForPaper:

```
- (NSSize)pageSizeForPaper:(NSString *)paperName
```

Returns the size of the page for the named paper type. The selection of paper type names depends on the NSPrinter's type; typical names include "Legal", "Letter", "A4", and "B5". See also imageRectForPaper:.

rectForKey:inTable:

```
- (NSRect)rectForKey:(NSString *)key inTable:(NSString *)table
```

Returns an NSRect for the given key in the given table. The individual fields are set to 0.0 if the value, which is stored as ASCII text, can't be fit into an NSRect structure. key should be formed as described in the NSPrinter "Class Description". See also stringForKey:inTable:, floatForKey:inTable:, intForKey:inTable:, booleanForKey:inTable:, sizeForKey:inTable:.

sizeForKey:inTable:

```
- (NSSize)sizeForKey:(NSString *)key inTable:(NSString *)table
```

Returns an NSSize for the given key in the given table. The individual fields are set to 0.0 if the value, which is stored as ASCII text, can't be fit into an NSSize structure. key should be formed as described in the NSPrinter "Class Description". See also stringForKey:inTable:, floatForKey:inTable:, intForKey:inTable:, booleanForKey:inTable:, rectForKey:inTable:.

stringForKey:inTable:

```
- (NSString *)stringForKey:(NSString *)key inTable:(NSString
*)table
```

Returns a string associated with key in table. If the table contains more than one entry with this key, the value of the first entry is returned. A pointer to NULL is returned if the table doesn't contain a key that precisely matches key. See the NSPrinter "Class Description" for more information on this method. See also stringListForKey:inTable:, floatForKey:inTable:, intForKey:inTable:, booleanForKey:inTable:, rectForKey:inTable:, sizeForKey:inTable:.

stringListForKey:inTable:

```
- (NSArray *)stringListForKey:(NSString *)key inTable:(NSString
*)table
```

Returns an array of strings that contain the ASCII text that corresponds to an entry that has the given key in the given table. If key names a main keyword for which there are (in the table) option keywords, the returned array contains the option keywords. See the NSPrinter Class Description, above, for more information on this method. See also stringForKey:inTable:, floatForKey:inTable:, intForKey:inTable:, booleanForKey:inTable:, rectForKey:inTable:, sizeForKey:inTable:.

statusForTable:

```
- (NSPrinterTableStatus)statusForTable:(NSString *)table
```

Returns the status (NSPrinterTableOK, NSPrinterTableNotFound, NSPrinterTableError) of the given table.

type

```
- (NSString *)type
```

Returns the string that names the printer's type. See also `printerWithType:`.

NSPrintInfo

Inherits From:	NSObject
Conforms To:	NSCoding, NSCopying NSObject (NSObject)
Declared In:	AppKit/NSPrintInfo.h

Class Description

An `NSPrintInfo` object stores information that's used during printing. A shared `NSPrintInfo` object is automatically created for an application and is used by default for all printing jobs for that application. You can create any number of additional `NSPrintInfo` objects; however, only one can be "active" at a time, as set through the `setSharedPrintInfo:` class method. The shared `NSPrintInfo` object is returned through the `sharedPrintInfo` class method.

An `NSPrintInfo` object is used by the `NSPrintOperation` class to control printing. If you create special instances of `NSPrintInfo` objects for a specific printing task, you must ensure that either the application's shared `NSPrintInfo` object is current, or you must instantiate an `NSPrintOperation` object using one of its methods that explicitly designate an `NSPrintInfo` object.

Although you can set an `NSPrintInfo`'s attributes through the methods it provides, this is usually the task of other objects, notably `NSPageLayout` objects. The `NSView` or `NSWindow` that's being printed may also supercede some `NSPrintInfo` settings. In particular, a `NSView` or `NSWindow` can supply the range of pages in the document and can provide its own pagination mechanism through the `knowsPagesFirst:last:` and `rect:forPage:` methods (see the documentation of these methods in the `NSView` class for details).

If the printed NSView or NSWindow doesn't supply pagination, the NSPrintInfo's vertical and horizontal pagination constants are used to trigger built-in pagination mechanisms:

Table 1-18 Pagination Constants

Constant	Meaning
NSAutoPagination	Image is diced into equal-sized rectangles and placed in one column of pages.
NSFitPagination	Image is scaled to produce one column or one row of pages.
NSClipPagination	Image is clipped to produce one column or row of pages.

Vertical and horizontal pagination needn't be the same. However, if either dimension is scaled (NSFitPagination), the other dimension is scaled by the same amount to avoid stretching the image. If both dimensions are scaled, the scaling factor that produces the smallest image is used. Note that NSPrintInfo's scaling factor is independent of the scaling that's imposed by pagination and is applied after the document has been paginated.

NSPrintInfo uses points as the unit of measurement for paper size and margin width in the methods below. See the NSFont specification for a discussion of points.

Method Types

Activity	Class Method
Creating and initializing an NSPrintInfo instance	– initWithDictionary:
Managing the shared NSPrintInfo object	+ setSharedPrintInfo: + sharedPrintInfo
Managing the printing rectangle	+ sizeForPaperName: – bottomMargin – leftMargin – orientation – paperName – paperSize – rightMargin – setBottomMargin: – setLeftMargin: – setOrientation: – setPaperName: – setPaperSize: – setRightMargin: – setTopMargin: – topMargin
Pagination	– horizontalPagination – setHorizontalPagination: – setVerticalPagination: – verticalPagination
Positioning the image on the page	– isHorizontallyCentered – isVerticallyCentered – setHorizontallyCentered: – setVerticallyCentered:
Specifying the printer	+ defaultPrinter + setDefaultPrinter: – printer – setPrinter:
Controlling printing	– jobDisposition – setJobDisposition: – setUpPrintOperationDefaultValues
Accessing the NSPrintInfo object's dictionary	– dictionary

Class Methods

defaultPrinter

```
+ (NSPrinter *)defaultPrinter
```

Returns an `NSPrinter` object that corresponds to the user's default printer, as declared in the defaults database. If the printer can't be found, `nil` is returned. See also `setDefaultPrinter:`, `printer`, `setPrinter:`, `NSUserDefaults` (Foundation Kit).

setDefaultPrinter:

```
+ (void)setDefaultPrinter:(NSPrinter *)printer
```

Sets the user's default printer by writing the name and host of `printer` to the defaults database. Unless a `NSPrintInfo`'s printer is otherwise set (through `setPrinter:`) the default printer is used for printing. See also `defaultPrinter`, `printer`.

setSharedPrintInfo:

```
+ (void)setSharedPrintInfo:(NSPrintInfo *)printInfo
```

Sets the shared `NSPrintInfo` object to `printInfo`.

sharedPrintInfo

```
+ (NSPrintInfo *)sharedPrintInfo
```

Returns the shared `NSPrintInfo` object, creating it if necessary. See also `setSharedPrintInfo:`.

sizeForPaperName:

```
+ (NSSize)sizeForPaperName:(NSString *)name
```

Returns the size for the specified type of paper. `name` identifies the type of paper. Paper names are implementation specific. Default page names are Letter, Tabloid, Ledger, Legal, Executive, A3, A4, A5, B4, B5.

Instance Methods

bottomMargin

```
- (float)bottomMargin
```

Returns the height of the bottom margin. See also `setBottomMargin:`, `leftMargin`, `rightMargin`, `topMargin`.

dictionary

```
(NSMutableDictionary *)dictionary
```

Returns the `NSPrintInfo` object's dictionary. See the "Printing" section of the Application Kit's "Types and Constants" chapter for a list of printing information dictionary keys. See also `initWithDictionary:`.

horizontalPagination

```
- (NSPrintingPaginationMode)horizontalPagination
```

Returns the horizontal pagination mode, which can be one of the following values:

- `NSAutoPagination`
- `NSFitPagination`
- `NSClipPagination`

See the "Class Description" for more information on pagination modes. See also `setHorizontalPagination:`, `verticalPagination`.

initWithDictionary:

```
- (id)initWithDictionary:(NSDictionary *)aDict
```

Initializes a newly allocated `NSPrintInfo` object by assigning it the parameters specified in `aDict`. This is the designated initializer for the class. See also `dictionary`.

isHorizontallyCentered

```
- (BOOL)isHorizontallyCentered
```

Returns YES if the image is centered horizontally on a page; if this returns NO, the image is flush against the left margin. If the image spills over more than one page horizontally, the image is always set against the left margin. See also `setHorizontallyCentered:`, `isVerticallyCentered`.

isVerticallyCentered

```
- (BOOL)isVerticallyCentered
```

Returns YES if the image is centered vertically on a page; if this returns NO, the image is flush against the top margin. If the image spills over more than one page vertically, the image is always set against the top margin. See also `setVerticallyCentered:`, `isHorizontallyCentered`.

jobDisposition

```
- (NSString *)jobDisposition
```

Returns the action specified for the job: printing, faxing, previewing, and so on. See also `setJobDisposition:`.

leftMargin

```
- (float)leftMargin
```

Returns the width of the left margin. See also `setLeftMargin:`, `rightMargin`.

orientation

```
- (NSPrintingOrientation)orientation
```

Returns the print job orientation, which can be one of the following values:

- `NSPortraitOrientation`
- `NSLandscapeOrientation`

See also `setOrientation:`.

paperName

```
- (NSString *)paperName
```

Returns the paper type, such as “Letter” or “Legal”. Paper names are implementation specific. See also `setPaperName:`, `paperSize`, `sizeForPaperName:`.

paperSize

```
- (NSSize)paperSize
```

Returns the size of the paper. See also `setPaperSize:`, `paperName`, `sizeForPaperName:`.

printer

```
- (NSPrinter *)printer
```

Returns the `NSPrinter` that’s used for printing. See also `setPrinter:`, `defaultPrinter`.

rightMargin

```
- (float)rightMargin
```

Returns the width of the right margin. See also `setRightMargin:`, `leftMargin`, `bottomMargin`, `topMargin`.

setBottomMargin:

```
- (void)setBottomMargin:(float)value
```

Sets the bottom margin to `value`. See also `bottomMargin`, `setLeftMargin:`, `setTopMargin:`.

setHorizontalPagination:

```
- (void)setHorizontalPagination:(NSPrintingPaginationMode)mode
```

Sets the horizontal pagination mode, which can be one of the following values:

- `NSAutoPagination`
- `NSFitPagination`
- `NSClipPagination`

See the Class Description for more information on pagination modes. See also `horizontalPagination`, `setVerticalPagination:`.

setHorizontallyCentered:

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

Sets whether the image is centered horizontally on a page; if `flag` is `NO`, the image is flush against the left margin. If the image spills over more than one page horizontally, then `flag` is ignored and the image is always against the left margin. See also `isHorizontallyCentered`, `setVerticallyCentered:`.

setJobDisposition:

`- (void)setJobDisposition:(NSString *)disposition`

Sets the action specified for the job. `disposition` can be one of the following values:

- `NSPrintSpoolJob`
- `NSPrintFaxJob`
- `NSPrintPreviewJob`
- `NSPrintSaveJob`
- `NSPrintCancelJob`

See also `jobDisposition`.

setLeftMargin:

`- (void)setLeftMargin:(float)value`

Sets the left margin to `value`. See also `leftMargin`, `setRightMargin:`, `setBottomMargin:`, `setTopMargin:`.

setOrientation:

`- (void)setOrientation:(NSPrintingOrientation)mode`

Sets the orientation as Portrait or Landscape. `mode` can be one of the following values:

- `NSPortraitOrientation`
- `NSLandscapeOrientation`

See also orientation.

setPaperName:

```
- (void)setPaperName:(NSString *)name
```

Sets the paper type. name identifies the type of paper, such as “Letter” or “Legal”. Paper names are implementation specific. See also paperName, paperSize, sizeForPaperName:.

setPaperSize:

```
- (void)setPaperSize:(NSSize)size
```

Sets the width and height of the paper. See also paperSize, setPaperName:, sizeForPaperName:.

setPrinter:

```
- (void)setPrinter:(NSPrinter *)aPrinter
```

Sets the printer that’s used in subsequent printing jobs. See also printer, defaultPrinter.

setRightMargin:

```
- (void)setRightMargin:(float)value
```

Sets the right margin to value. See also rightMargin, setLeftMargin:, setTopMargin:, setBottomMargin:.

setTopMargin:

```
- (void)setTopMargin:(float)value
```

Sets the top margin to value. See also topMargin, setBottomMargin:, setLeftMargin:.

setUpPrintOperationDefaultValues

```
- (void)setUpPrintOperationDefaultValues
```

Sets any likely to change attributes to default values before a print job. All information that's likely to change between operations is set to a default value in the `NSPrintInfo` before the operation begins. In this way, even though an `NSPrintOperation` updates the `NSPrintInfo` with information from the Print panel for print jobs, that information is reset back to the default values for each print job. The default values set are as follows:

Attribute	Value
First page	INT_MIN
Last page	INT_MAX
Copies	1
Page order	First-to-last
Printer	The user's default printer
Paper feed	The default paper feed slot

setVerticalPagination:

```
- (void)setVerticalPagination:(NSPrintingPaginationMode)mode
```

Sets the vertical pagination mode, which can be one of the following values:

- `NSAutoPagination`
- `NSFitPagination`
- `NSClipPagination`

See the "Class Description" for more information on pagination modes. See also `verticalPagination`, `setHorizontalPagination:`.

setVerticallyCentered:

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

Sets whether the image is centered vertically on a page; if `flag` is `NO`, the image is flush against the top margin. If the image spills over more than one page vertically, then `flag` is ignored and the image is always against the top margin. See also `isVerticallyCentered`, `setHorizontallyCentered:`.

topMargin

```
- (float)topMargin
```

Returns the height of the top margin. See also `setTopMargin:`, `bottomMargin`, `leftMargin`, `rightMargin`.

verticalPagination

```
- (NSPrintingPaginationMode)verticalPagination
```

Returns the vertical pagination mode, which can be one of the following values:

- `NSAutoPagination`
- `NSFitPagination`
- `NSClipPagination`

See the Class Description for more information on pagination modes. See also `setVerticalPagination:`, `horizontalPagination`.

NSPrintOperation

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

Class Description

`NSPrintOperation` controls operations that generate Encapsulated PostScript (EPS) code or PostScript print jobs. Generally, EPS code is used to transfer images between applications, which happens when the user copies and pastes graphics, uses a Service, or uses ObjectLinks. PostScript print jobs are generated when the user prints and faxes documents. An `NSPrintOperation` does not generate PostScript code itself; it just controls the overall process, relying on an `NSView` object to generate the actual code.

`NSPrintOperation` relies mainly on two other objects: an `NSPrintInfo` object, which specifies how the code should be generated, and an `NSView` object, which performs the actual code generation. You specify these two objects in the method you use to create the `NSPrintOperation`. If no

NSPrintInfo is specified, NSPrintOperation uses the shared NSPrintInfo, which contains default values. The shared NSPrintInfo works well for applications that are not document-based. Document-based applications should create an NSPrintInfo for each document that might be printed or copied and use that object instead.

You should create an NSPrintOperation in any method that is invoked when a user executes a Print command or a Copy command. That method also must send NSPrintOperation a runOperation message to start the operation. A print: method for a document-based application might look like this:

```
- (void)print:sender {
[[NSPrintOperation printOperationWithView:[self myView]
    printInfo:[document docPrintInfo]] runOperation];
}
```

This method creates an NSPrintOperation for a print job that uses the document's NSPrintInfo. Because this is a print job, a Print panel is displayed to allow the user to select printing options. The NSPrintOperation copies the NSPrintInfo, updates this copy with information from the Print panel, and uses the specified NSView to perform the operation.

The information stored in an NSPrintInfo that's retained between operations is information that's likely to remain constant for a document, such as its page size. All information that's likely to change between operations is set to a default value in the NSPrintInfo before the operation begins. In this way, even though NSPrintOperation updates the NSPrintInfo with information from the Print panel for print jobs, that information is reset back to the default values for each print job. Because NSPrintOperation keeps a copy of the NSPrintInfo it uses, you could duplicate a specific print job by storing that copy and reusing it.

You can augment a Print panel display by adding a custom NSView through the setAccessoryView: method. The panel is automatically resized to accommodate the NSView that you add. Note, however, that you don't have to create controls for special printer features. If a printer includes features in the "OpenUI" field of its PostScript Printer Description (PPD) table, these features will be displayed in a separate panel that's brought up when the user clicks the Print panel's Options button. For more information on a printer's PPD table, see the NSPrinter "Class Description".

Method Types

Activity	Class Method
Creating and initializing an NSPrintOperation object	+ EPSOperationWithView:insideRect:toData: + EPSOperationWithView:insideRect:toData: printInfo: + EPSOperationWithView:insideRect:toPath: printInfo: + printOperationWithView: + printOperationWithView:printInfo: – initEPSOperationWithView:insideRect:toData: printInfo:
Setting the print operation	+ currentOperation + setCurrentOperation:
Determining the type of operation	– isEPSOperation
Controlling the user interface	– showPanels – setAccessoryView: – setShowPanels:
Managing the DPS context	– createContext – context – destroyContext
Page information	– currentPage – pageOrder – setPageOrder:
Running a print operation	– cleanUpOperation – deliverResult – runOperation
Getting the NSPrintInfo object	– printInfo – setPrintInfo:
Getting the NSView object	– view
Methods Implemented by the Delegate	– finalWritePrintInfo – updateFromPrintInfo

Class Methods

currentOperation

```
+ (NSPrintOperation *)currentOperation
```

Returns the NSPrintOperation that represents the current operation or nil if there is no such operation. See also setCurrentOperation:.

EPSOperationWithView:insideRect:toData:

```
+ (NSPrintOperation *)EPSOperationWithView:(NSView *)aView
    insideRect:(NSRect)rect toData:(NSMutableData *)data
```

Returns a new NSPrintOperation that controls the copying of EPS graphics from the area specified by rect in aView, using the parameters in the default NSPrintInfo. The code is written to data. This method raises NSPrintOperationExistsException if there is already a print operation in progress. See also EPSOperationWithView:insideRect:toData: printInfo:, EPSOperationWithView:insideRect:toPath: printInfo:, printOperationWithView:.

EPSOperationWithView:insideRect:toData: printInfo:

```
+ (NSPrintOperation *)EPSOperationWithView:(NSView *)aView
    insideRect:(NSRect)rect toData:(NSMutableData *)data
    printInfo:(NSPrintInfo *)aPrintInfo
```

Returns a new NSPrintOperation that controls the copying of EPS graphics from the area specified by rect in aView, using the parameters in aPrintInfo. The code is written to data. Raises NSPrintOperationExistsException if there is already a print operation in progress. See also EPSOperationWithView:insideRect:toData:.

EPSOperationWithView:insideRect:toPath: printInfo:

```
+ (NSPrintOperation *)EPSOperationWithView:(NSView *)aView
    insideRect:(NSRect)rect toPath:(NSString *)path
    printInfo:(NSPrintInfo *)aPrintInfo
```

Returns a new NSPrintOperation that controls the copying of EPS graphics from the area specified by rect in aView, using the parameters in aPrintInfo. The code is written to path. Raises NSPrintOperationExistsException if there is already a print operation in progress. See also EPSOperationWithView:insideRect:toData:.

printOperationWithView:

```
+ (NSPrintOperation *)printOperationWithView:(NSView *)aView
```

Returns a new NSPrintOperation that controls the printing of aView, using the parameters in the shared NSPrintInfo object. Raises NSPrintOperationExistsException if there is already a print operation in progress. See also printOperationWithView:printInfo:.

printOperationWithView:printInfo:

```
+ (NSPrintOperation *)printOperationWithView:(NSView *)aView
    printInfo:(NSPrintInfo *)aPrintInfo
```

Returns a new NSPrintOperation that controls the printing of aView, using the parameters in aPrintInfo. Raises NSPrintOperationExistsException if there is already a print operation in progress. See also printOperationWithView:.

setCurrentOperation:

```
+ (void)setCurrentOperation:(NSPrintOperation *)operation
```

Sets the NSPrintOperation that represents the current operation. See also currentOperation.

Instance Methods

cleanUpOperation

```
- (void)cleanUpOperation
```

Invoked at the end of an operation's run to set the current operation to nil. See also runOperation.

context

```
- (NSDPSContext *)context
```

Returns the DPS context used for the receiver's operation. See also createContext, destroyContext.

createContext

```
- (NSDPSContext *)createContext
```

Used by the NSPrintOperation object to create the DPS context for output generation, using the current NSPrintInfo settings.

currentPage

```
- (int)currentPage
```

Returns the page number of the page being printed. See also pageOrder, setPageOrder:.

deliverResult

```
- (BOOL)deliverResult
```

Delivers the results generated by runOperation to the intended destination: the print spooler, preview application, and so on. Returns YES upon successful delivery and NO otherwise.

destroyContext

```
- (void)destroyContext
```

Used by the NSPrintOperation object to destroy the DPS context at the end of the operation. See also createContext, context.

initEPSOperationWithView:insideRect:toData:printInfo:

```
- (id)initEPSOperationWithView:(NSView *)aView
    insideRect:(NSRect)rect
    toData:(NSMutableData *)data
    printInfo:(NSPrintInfo *)aPrintInfo
```

Initializes a newly allocated NSPrintOperation to control the copying of EPS graphics from the area specified by rect in aView, using the parameters in aPrintInfo. The code is written to data. See also initWithView:printInfo:.

initWithView:printInfo:

```
- (id)initWithView:(NSView *)aView
    printInfo:(NSPrintInfo *)aPrintInfo
```

Initializes a newly allocated `NSPrintOperation` to control the printing of `aView`, using the parameters in `aPrintInfo`. See also `initEPSOperationWithView:insideRect:toData: printInfo:`.

isEPSOperation

```
- (BOOL)isEPSOperation
```

Returns `YES` if the receiver controls an EPS operation and `NO` if the receiver controls a printing operation.

pageOrder

```
- (NSPrintingPageOrder)pageOrder
```

Returns the order in which pages will be printed, represented by one of the following values:

- `NSDescendingPageOrder`
- `NSSpecialPageOrder`
- `NSAscendingPageOrder`
- `NSUnknownPageOrder`

See the Printing section of the Application Kit's Types and Constants chapter for more information. See also `setPageOrder:`.

printInfo

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

Returns the receiver's `NSPrintInfo` object. See also `setPrintInfo:`.

runOperation

```
- (BOOL)runOperation
```

Causes the operation (copying EPS graphics or printing) to take place. Returns YES upon successful completion and NO otherwise. See also cleanUpOperation, deliverResult.

setAccessoryView:

```
- (void)setAccessoryView:(NSView *)aView
```

Adds aView to the printing panel's view hierarchy. Applications can invoke this method to add an NSView that contains its own controls. The panel is automatically resized to accommodate aView. This method can be invoked repeatedly to change the accessory view depending on the situation. If aView is nil, then the panel's current accessory view, if any, is removed.

setPageOrder:

```
- (void)setPageOrder:(NSPrintingPageOrder)order
```

Sets the order in which pages will be printed. See also pageOrder, currentPage.

setPrintInfo:

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

Sets the receiver's NSPrintInfo object to aPrintInfo. See also printInfo.

setShowPanels:

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

Sets whether the Print panel appears when the operation is run. See also showPanels.

showPanels

```
- (BOOL)showPanels
```

Returns whether the Print panel will appear when the operation is run. See also setShowPanels:.

view

```
- (NSView *)view
```

Returns the NSView object that performs the operation controlled by the receiving object.

Methods Implemented by the Delegate

finalWritePrintInfo

```
- (void)finalWritePrintInfo
```

Writes NSPrintOperation's values to the application's NSPrintInfo object if the user changes information on the Print panel. See also updateFromPrintInfo.

updateFromPrintInfo

```
- (void)updateFromPrintInfo
```

Reads the application's NSPrintInfo object, setting the initial values of the Print panel. This method is invoked automatically.

NSResponder

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

Class Description

NSResponder is an abstract class that forms the basis of command and event processing in the Application Kit. Most Application Kit classes inherit from NSResponder. When an NSResponder receives an event or action message that it can't respond to—that it doesn't have a method for—the message is sent to its *next responder*. For an NSView, the next responder is usually its

superview; the content view's next responder is the `NSWindow`. Each `NSWindow`, therefore, has its own *responder chain*. Messages are passed up the chain until they reach an object that can respond.

Action messages and keyboard event messages are sent first to the *first responder*, the object that displays the current selection and is expected to handle most user actions within a window. Each `NSWindow` has its own first responder. Messages the first responder can't handle work their way up the responder chain. This class defines the methods that pass event and action messages along the responder chain.

Method Types

Activity	Class Method
Managing the next responder	– nextResponder – setNextResponder:
Determining the first responder	– acceptsFirstResponder – becomeFirstResponder – resignFirstResponder
Aiding event processing	– performKeyEquivalent: – tryToPerform:with:
Forwarding event messages	– flagsChanged: – helpRequested: – keyDown: – keyUp: – mouseDown: – mouseDragged: – mouseEntered: – mouseExited: – mouseMoved: – mouseUp: – noResponderFor: – rightMouseDown: – rightMouseDragged: – rightMouseUp:
Services menu support	– validRequestorForSendType:returnType:

Instance Methods

acceptsFirstResponder

```
– (BOOL)acceptsFirstResponder
```

Subclasses override to accept or reject first responder status. `NSResponder`'s implementation simply returns `NO`. Before making any object the first responder, the Application Kit gives it an opportunity to refuse by sending it an `acceptsFirstResponder` message. Objects that can display a selection should override this default to return `YES`. Objects that respond with this default version of the method will receive mouse event messages, but no others. See also `becomeFirstResponder`, `resignFirstResponder`, `nextResponder`, `setNextResponder:`.

becomeFirstResponder

```
- (BOOL)becomeFirstResponder
```

Notifies the receiver that it has become the first responder for its NSWindow. This default version of the method simply returns YES. NSResponder subclasses can implement their own versions to take whatever action may be necessary, such as highlighting the selection.

By returning YES, the receiver accepts being made the first responder. A NSResponder can refuse to become the first responder by returning NO. becomeFirstResponder messages are initiated by the NSWindow object through NSWindow's makeFirstResponder: method in response to mouse-down events. See also becomeFirstResponder, acceptsFirstResponder.

flagsChanged:

```
- (void)flagsChanged:(NSEvent *)theEvent
```

Subclasses override this method to handle flags-changed events. NSResponder's implementation passes the message to the receiver's next responder, or sounds a beep via noReponderFor: if there is no next responder.

helpRequested:

```
- (void)helpRequested:(NSEvent *)theEvent
```

Causes the Help panel to display the help attached to the receiver. If there's no attached help, passes the message to the receiver's next responder. This method is invoked by an NSWindow instance when the user has clicked for help. Your application should never invoke this method directly. The NSWindow instance sends this message to the first responder. The receiver shows its Help panel if it has one, or sounds a beep (via noReponderFor:) if there is no next responder.

keyDown:

```
- (void)keyDown:(NSEvent *)theEvent
```

Subclasses override this method to handle key-down events. NSResponder's implementation passes the message to the receiver's next responder. If the first responder changes, this method posts the notification NSTextDidEndEditingNotification with the current object and, in the notification's dictionary, and the key NSTextMovement to the default notification center.

keyUp:

```
- (void)keyUp:(NSEvent *)theEvent
```

Subclasses override to handle key-up events. NSResponder's implementation passes the message to the receiver's next responder. See also mouseDown:.

mouseDown:

```
- (void)mouseDown:(NSEvent *)theEvent
```

Subclasses override to handle mouse-down events. NSResponder's implementation passes the message to the receiver's next responder. See also mouseDragged:, mouseEntered:, mouseExited:, mouseMoved:, mouseUp:, noResponderFor:, rightMouseDown:, rightMouseUp:.

mouseDragged:

```
- (void)mouseDragged:(NSEvent *)theEvent
```

Subclasses override to handle mouse-dragged events. NSResponder's implementation passes the message to the receiver's next responder. See also mouseDown:.

mouseEntered:

```
- (void)mouseEntered:(NSEvent *)theEvent
```

Subclasses override to handle mouse-entered events. NSResponder's implementation passes the message to the receiver's next responder. See also mouseDown:.

mouseExited:

```
- (void)mouseExited:(NSEvent *)theEvent
```

Subclasses override to handle mouse-exited events. NSResponder's implementation passes the message to the receiver's next responder. See also mouseDown:.

mouseMoved:

```
- (void)mouseMoved:(NSEvent *)theEvent
```

Subclasses override to handle mouse-moved events. NSResponder's implementation passes the message to the receiver's next responder. See also mouseDown:.

mouseUp:

```
- (void)mouseUp:(NSEvent *)theEvent
```

Subclasses override to handle mouse-up events. NSResponder's implementation passes the message to the receiver's next responder. See also mouseDown:.

nextResponder

```
- (NSResponder *)nextResponder
```

Returns the receiver's next responder. See also mouseDown:.

noResponderFor:

```
- (void)noResponderFor:(SEL)eventSelector
```

Responds to an event message that has reached the end of the responder chain without finding an object that can respond. When the event is a key down, this default method generates a beep. See also mouseDown:.

performKeyEquivalent:

```
- (BOOL)performKeyEquivalent:(NSEvent *)theEvent
```

Subclasses override this method to respond to keyboard input passed as `theEvent`. `NSResponder`'s implementation returns `NO` to indicate that, by default, the `NSResponder`-based object doesn't have a key equivalent, and can't respond to key-down events as keyboard alternatives. See also `mouseDown:`.

resignFirstResponder

```
- (BOOL)resignFirstResponder
```

Notifies the receiver that it's not the first responder. See also `mouseDown:`.

rightMouseDown:

```
- (void)rightMouseDown:(NSEvent *)theEvent
```

Subclasses override to handle right mouse-down events. `NSResponder`'s implementation passes the message to the receiver's next responder. See also `mouseDown:`.

rightMouseDragged:

```
- (void)rightMouseDragged:(NSEvent *)theEvent
```

Subclasses override to handle right mouse-dragged events. `NSResponder`'s implementation passes the message to the receiver's next responder. See also `mouseDown:`.

rightMouseUp:

```
- (void)rightMouseUp:(NSEvent *)theEvent
```

Subclasses override to handle right mouse-up events. `NSResponder`'s implementation passes the message to the receiver's next responder. See also `mouseDown:`.

setNextResponder:

```
- (void)setNextResponder:(NSResponder *)aResponder
```

Makes `aResponder` the receiver's next responder. See also `mouseDown:`.

tryToPerform:with:

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

Aids in dispatching action messages. Returns YES if a responder in the responder chain can perform the anAction method, which takes the single argument anObject. See also performKeyEquivalent:.

validRequestorForSendType:returnType:

```
- (id)validRequestorForSendType:(NSString *)typeSent
    returnType:(NSString *)typeReturned
```

Subclasses override to determine which Services menu items are enabled at a given time. Returning self enables services that can receive typeSent pasteboard types and can return typeReturned pasteboard types. Returning nil disables them. NSResponder's implementation passes the message to the receiver's next responder.

NSSavePanel

Inherits From:	NSPanel : NSWindow : NSResponder : NSObject
Conforms To:	NSCoding (NSResponder) NSObject (NSObject)
Declared In:	AppKit/NSSavePanel.h

Class Description

NSSavePanel creates a Save panel. The Save panel provides a simple way for a user to specify a file to use when saving a document or other data. It can restrict the user to files of a certain type, as specified by a file name extension.

When the user decides on a file name, the message panel:isValidFilename: is sent to the NSSavePanel's delegate (if it responds to that message). The delegate can then determine whether that file name can be used; it returns YES if the file name is valid, or NO if the Save panel should stay up and wait for the user to type in a different file name.

Typically, you access an NSSavePanel by invoking the savePanel method. When the class receives a savePanel message, it tries to reuse an existing panel rather than create a new one. When a panel is reused, its attributes are

reset to the default values so that the effect is the same as receiving a new panel. Because a Save panel may be reused, you shouldn't modify the instance returned by `savePanel`, except through the methods listed in the following. For example, you can set the panel's title and required file type, but not the arrangement of the buttons within the panel. If you must modify the Save panel substantially, create and manage your own instance using the `alloc...` and `init...` methods rather than the `savePanel` method.

Note that Save and Open panels remember the last directory to which the user traversed. That is, anytime a Save or Open panel is shown to the user, the default directory it displays is the directory the user was at the last time they clicked "ok", or double-clicked a file. If no such previous directory exists, the panels will go to the user's home directory. Because of this default behavior, it is not necessary to send the `setDirectory:` method unless behavior different from this is desired.

See also `NSOpenPanel`.

Method Types

Activity	Class Method
Creating an NSSavePanel	+ savePanel
Customizing the NSSavePanel	– accessoryView – prompt – title – setAccessoryView: – setPrompt: – setTitle:
Setting directory and file type	– requiredFileType – setDirectory: – setRequiredFileType: – setTreatsFilePackagesAsDirectories: – treatsFilePackagesAsDirectories
Running the NSSavePanel	– runModalForDirectory:file: – runModal
Reading save information	– directory – filename
Target and action methods	– ok: – cancel:
Responding to user input	– selectText:
Manipulating columns	– validateVisibleColumns
Setting the delegate	– setDelegate:
Methods Implemented by theDelegate	– panel:compareFilename:with:caseSensitive: – panel:isValidFilename: – panel:shouldShowFilename:

Class Methods

savePanel

```
+(NSSavePanel *)savePanel
```

Returns an `NSSavePanel` object, creating it if necessary. Each application shares just one instance of `NSSavePanel`; this method returns the shared instance if it exists.

Instance Methods

accessoryView

```
- (NSView *)accessoryView
```

Returns the application-customized view object. See also `setAccessoryView:`.

cancel:

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

This is the method invoked by the Cancel button. See also `ok:`.

directory

```
- (NSString *)directory
```

Returns the path of the directory that the Save panel is currently showing. See also `filename`, `setDirectory:`.

filename

```
- (NSString *)filename
```

Returns the absolute path name of the file to be saved. See also `directory`.

ok:

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

Method invoked by the OK button. See also `cancel:`.

prompt

```
- (NSString *)prompt
```

Returns the title of the form field for the path. See also `setPrompt:`.

requiredFileType

```
- (NSString *)requiredFileType
```

Gets the required file type (if any). See also `setRequiredFileType:`.

runModal

```
- (int)runModal
```

Displays the save panel and begins its event loop. Invokes `NSApplication`'s `runModalFor:` method with `self` as the argument. Returns `NSOKButton` if the user clicks the OK button, or `NSCancelButton` if the user clicks the Cancel button. See also `runModalForDirectory:file:`.

runModalForDirectory:file:

```
- (int)runModalForDirectory:(NSString *)path
    file:(NSString *)filename
```

Displays the save panel and begins its event loop, showing `path` in the browser and selecting `filename`. See also `runModal`.

selectText:

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

Invoked when users press Tab, Shift-Tab, or an arrow key.

setAccessoryView:

```
- (void)setAccessoryView:(NSView *)aView
```

Adds an application-customized view to the save panel. `aView` should be the top view in a view hierarchy, and will be added just above the OK and Cancel buttons at the bottom of the panel. The panel is automatically resized to accommodate `aView`. This method can be called repeatedly to change the accessory view depending on the situation. If `aView` is `nil`, any accessory view in the panel will be removed.

setDelegate:

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

Makes `anObject` the save panel's delegate.

setDirectory:

```
- (void)setDirectory:(NSString *)path
```

Sets the current path in the Save panel browser. Since the Save and Open panels remember the last directory the user traversed to, provided that the user doesn't press the Cancel button, it is only necessary to send this method if you want to change this behavior. See also `directory`.

setPrompt:

```
- (void)setPrompt:(NSString *)prompt
```

Sets the title for the form field in which users type their entries into the panel. This title will appear on all `NSSavePanel`s (or all `NSOpenPanel`s if the receiver of this message is an `NSOpenPanel`) in your application. "File:" is the default prompt string. See also `prompt`.

setRequiredFileType:

```
- (void)setRequiredFileType:(NSString *)type
```

Specifies the required file `type`, a file name extension to be appended to any selected files that don't already have that extension; for example, "nib". `type` should not include the period that begins the extension. Be careful to invoke this method each time the save panel is used for another file type within the application. See also `requiredFileType`.

setTitle:

```
- (void)setTitle:(NSString *)title
```

Sets the title of the save panel to `title`. By default, "Save" is the title string. If a save panel is adapted to other uses, its title should reflect the user action that brings it to the screen. See also `title`.

setTreatsFilePackagesAsDirectories:

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

Sets whether the NSSavePanel object treats file packages as directories by showing their contents in the browser.

title

```
- (NSString *)title
```

Returns the save panel title. See also setTitle:.

treatsFilePackagesAsDirectories

```
- (BOOL)treatsFilePackagesAsDirectories
```

Returns YES if the save panel treats file packages as directories, thereby allowing users to browse the contents of file packages. See also setTreatsFilePackagesAsDirectories:.

validateVisibleColumns

```
- (void)validateVisibleColumns
```

Validates the columns visible in the Save panel. Use this method to confirm that the entries displayed in each visible column are valid before redrawing. See also validateVisibleColumns (NSBrowser).

Methods Implemented by the Delegate

panel:compareFilename:with:caseSensitive:

```
- (NSComparisonResult)panel:(id)sender
    compareFilename:(NSString *)filename1
    with:(NSString *)filename2
    caseSensitive:(BOOL)caseSensitive
```

Returns NSOrderedDescending if filename1 precedes filename2, NSOrderedAscending in the opposite case, NSOrderedSame if the two are equivalent. Use caution when reordering save panel file names, since it may confuse the user to have files in one Save panel or Open panel ordered

differently than those in other such panels or in the Workspace Manager. `NSSavePanel` and `NSOpenPanel`'s default behavior is to order files as they are in the Workspace Manager file viewer. Note also that implementing this method will reduce the operating performance of the panel.

`panel:isValidFilename:`

```
- (BOOL)panel:(id)sender isValidFilename:(NSString*)filename
```

Returns `YES` if `filename` is acceptable to the delegate.

`panel:shouldShowFilename:`

```
- (BOOL)panel:(id)sender shouldShowFilename:(NSString *)filename
```

Returns `YES` if `filename` should be displayed in the browser. A Save panel sends this message to the panel's delegate for each file or directory it is about to display in the browser. The delegate can then determine whether the `filename` should be displayed in the panel, giving it the ability to filter out items that it doesn't want the user to see or choose.

NSScreen

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

Class Description

An `NSScreen` object describes the attributes of a computer's monitor, or screen. An application may use an `NSScreen` object to retrieve information about a screen and use this information to decide what to display upon that screen. For example, an application may use the `deepestScreen` method to find out which of the available screens can best represent color and then may choose to display all of its windows on that screen.

The two main attributes of a screen are its depth and its dimensions. The `depth` method describes the screen depth (such as two-bit, eight-bit, or twelve-bit) and tells you if the screen can display color. The `frame` method gives the screen's dimensions and location as an `NSRect`.

The device description dictionary contains more complete information about the screen. Use `NSScreen`'s `deviceDescription` method to access the dictionary, and use these keys to retrieve information about a screen:

Table 1-19 Device-Description Dictionary Keys

Dictionary Key	Returns
NSDeviceResolution	NSValue describing the screen's resolution in dots per inch (dpi).
NSDeviceColorSpaceName	Screen's color space name. See NSGraphics.h for a list of possible values.
NSDeviceBitsPerSample	Bit depth of screen images (2-bit, 8-bit, and so on).
NSDeviceIsScreen	YES, indicating the device is a screen.
NSDeviceSize	NSValue describing the screen's size in points.

The device description dictionary contains information about not only screens, but all other system devices such as printers and windows. There are other keys into the dictionary that you would use to obtain information about these other devices. For a complete list of device dictionary keys, see `NSGraphics.h`.

Method Types

Activity	Class Method
Creating NSScreen instances	+ deepestScreen + mainScreen + screens
Reading screen information	– depth – deviceDescription – frame – supportedWindowDepths

Class Methods

deepestScreen

```
+ (NSScreen *)deepestScreen
```

Returns an NSscreen object representing the screen that can best represent color. This method always returns an object, even if there is only one screen and it is not a color screen.

mainScreen

```
+ (NSScreen *)mainScreen
```

Returns an NSScreen object representing the main screen. The main screen is the screen with the key window.

screens

```
+ (NSArray *)screens
```

Returns an array of NSScreen objects representing all of the screens available on the system. Raises NSWindowServerCommunicationException if the screen's information can't be obtained from the window system.

Instance Methods

depth

```
- (NSWindowDepth)depth
```

Returns the screen's depth, including whether the screen can display color.

deviceDescription

```
- (NSDictionary *)deviceDescription
```

Returns the device dictionary as described in the class description, above.

frame

```
- (NSRect)frame
```

Returns the dimensions and location of the screen in an NSRect.

supportedWindowDepths

```
- (const NSWindowDepth *const)supportedWindowDepths
```

Returns a 0-terminated list of supported window depths.

NSScroller

Inherits From:	NSControl : NSView : NSResponder : NSObject
Conforms To:	NSCoding (NSResponder) NSObject (NSObject)
Declared In:	AppKit/NSScroller.h

Class Description

The `NSScroller` class defines a control that's used by an `NSScrollView` object to position a document that's too large to be displayed in its entirety within an `NSView`. An `NSScroller` is typically represented on the screen by a bar, a knob, and two scroll buttons, although it may contain only some of these. The knob indicates both the position within the document and the amount displayed relative to the size of the document. The bar is the rectangular region that the knob slides within. The scroll buttons allow the user to scroll in small increments by clicking, or in large increments by Alternate-clicking. In discussions of the `NSScroller` class, a small increment is referred to as a "line increment" (even if the `NSScroller` is oriented horizontally), and a large increment is referred to as a "page increment," although a page increment actually advances the document by one windowful. When you create an `NSScroller`, you can specify either a vertical or a horizontal orientation.

As an `NSControl`, an `NSScroller` handles mouse events and sends action messages to its target (usually its parent `NSScrollView`) to implement user-controlled scrolling. The `NSScroller` must also respond to messages from an `NSScrollView` to represent changes in document positioning.

`NSScroller` is a public class primarily for programmers who decide not to use an `NSScrollView` but want to present a consistent user interface. Its use is not encouraged except in cases where the porting of an existing application is made more straightforward. In these situations, you initialize a newly created `NSScroller` by calling `initWithFrame:`. Then, you use `setTarget:`

(NSControl) to set the object that will receive messages from the NSScroller, and you use setAction: (NSControl) to specify the message that will be sent to the target by the NSScroller. When your target receives a message from the NSScroller, it will probably need to query the NSScroller using the hitPart and floatValue (NSControl) methods to determine what action to take.

The NSScroller class has several constants referring to the parts of an NSScroller. A scroll button with an up arrow (or left arrow, if the NSScroller is oriented horizontally) is known as a "decrement line" button if it receives a normal click, and as a "decrement page" button if it receives an Alternate-click. Similarly, a scroll button with a down or right arrow functions as both an "increment line" button and an "increment page" button. See the hitPart method for a list of NSScroller part values.

An NSScroller can be made too small for all its parts to be displayed. See the usableParts method for information on when parts are no longer usable due to size constraints.

Method Types

Activity	Class Method
Laying out the NSScroller	+ scrollerWidth – arrowsPosition – checkSpaceForParts – rectForPart: – setArrowsPosition: – usableParts
Setting the NSScroller's values	– knobProportion – setFloatValue:knobProportion:
Displaying	– drawArrow:highlight: – drawKnob – drawParts – highlight:
Handling events	– hitPart – testPart: – trackKnob: – trackScrollButtons:

Class Methods

scrollerWidth

```
+ (float)scrollerWidth
```

Returns the width of the scroller, a constant value. See also `arrowsPosition`, `checkSpaceForParts`, `rectForPart:`, `setArrowsPosition:`, `usableParts`.

Instance Methods

arrowsPosition

```
- (NSScrollArrowPosition)arrowsPosition
```

Returns the scroll arrows position within the `NSScroller`. The following constants are used to set the position of the scroll-arrow buttons within the scroller:

Table 1-20 Constants Affecting the Position of Scroll Buttons

Constant	Meaning
NSScrollerArrowsMaxEnd	Scroll buttons are placed at the bottom or right end of the scroller.
NSScrollerArrowsMinEnd	Scroll buttons are placed at the top or left part of the scroller.
NSScrollerArrowsNone	Scroller doesn't have scroll buttons.

See also `setArrowsPosition:`, `drawArrow:highlight:`.

checkSpaceForParts

```
- (void)checkSpaceForParts
```

Checks for room for knob and scroll buttons, based on the `NSScroller` size. This method is used by other `NSScroller` methods. You should not invoke it yourself. See also `usableParts`.

drawArrow:highlight:

```
- (void)drawArrow:(NSScrollerArrow)whichButton highlight:(BOOL)flag
```

Draws highlighted and unhighlighted arrows. The following constants are used as values for the first argument to indicate which scroll button is to be drawn:

Table 1-21 Constants Indicating Which Scroll Button Is to Be Drawn

Constant	Meaning
NSScrollerIncrementArrow	Scroll button that scrolls forward.
NSScrollerDecrementArrow	Scroll button that scrolls backward.

If highlight is YES, the button is drawn highlighted, otherwise it's drawn normally. This method is invoked by other NSScroller methods; it's a public method so that you can override it, but you should not invoke it directly. See also drawKnob, drawParts, highlight:, setArrowsPosition:.

drawKnob

```
- (void)drawKnob
```

Draws the knob. Don't send this message directly. See also knobProportion, setFloatValue:knobProportion:, drawParts, drawArrow:highlight:.

drawParts

```
- (void)drawParts
```

This method caches images for the graphic entities (knob and scroll arrows) composing the NSScroller. It's invoked only once by initWithFrame:. You may want to override this method if you alter the look of the NSScroller, but you should not invoke it directly. See also drawArrow:highlight:.

highlight:

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

This method highlights or unhighlights the scroll button that the user clicked. The scroller invokes this method while tracking the mouse, and you should not invoke it directly. If flag is YES, the button is drawn highlighted, otherwise it's drawn normally. See also drawArrow:highlight:.

hitPart

```
- (NSScrollerPart)hitPart
```

Returns the part of the NSScroller that is causing the current action, typically the part that received a mouse-down event. The constants defining the parts of an NSScroller, and which describe this method's possible return values, are as follows:

Table 1-22 Constants Defining Parts of an NSScroller

Constant	Refers To
NSScrollerNoPart	No part of the NSScroller
NSScrollerKnob	Knob
NSScrollerDecrementPage	Button that decrements a windowful (up or left arrow)
NSScrollerIncrementPage	Button that increments a windowful (down or right arrow)
NSScrollerDecrementLine	Button that decrements a line (up or left arrow)
NSScrollerIncrementLine	Button that increments a line (down or right arrow)
NSScrollerKnobSlot	Bar

This method is typically invoked by the NSScrollView to determine what action to take when the NSScrollView receives an action message from the NSScroller. See also checkSpaceForParts, rectForPart:, usableParts, testPart:.

knobProportion

```
- (float)knobProportion
```

Returns the ratio of the knob's length to the NSScroller's length. See also setFloatValue:knobProportion:.

rectForPart:

```
- (NSRect)rectForPart:(NSScrollerPart)partCode
```

Gets the rectangle that encloses partCode. See the hitPart method description for a list of the NSScroller parts constants. See also checkSpaceForParts, usableParts, testPart:.

setArrowsPosition:

```
- (void)setArrowsPosition:(NSScrollArrowPosition)where
```

Sets position of scroll arrows in the NSScroller. See the arrowPostion method description for a description of the NSScroller button position constants. See also arrowsPosition, drawArrow:highlight:.

setFloatValue:knobProportion:

```
- (void)setFloatValue:(float)aFloat knobProportion:(float)ratio
```

Sets the NSScroller's position and size of the knob, repositioning the knob according to aFloat and resizing it according to ratio. Both arguments are clipped to the range from 0.0 to 1.0, inclusive. aFloat value of 0.0 positions and displays the knob at the top or left of the bar, depending on the orientation of the NSScroller. The size of the knob is determined by ratio, which is a value between 0.0 and 1.0. A value of 0.0 sets the knob to a predefined minimum size, and a value of 1.0 makes the knob fill the bar. See also drawKnob, trackKnob:.

testPart:

```
- (NSScrollerPart)testPart:(NSPoint)thePoint
```

Returns the NSScroller part that's under thePoint. See hitPart for a list and description of NSScrollerPart values.

trackKnob:

```
- (void)trackKnob:(NSEvent *)theEvent
```

Tracks the knob and sends action messages to the NSScroller's target. This method is invoked when the NSScroller receives a mouse-down event in the knob. You should not invoke this method directly. See also drawKnob, trackScrollButtons:.

trackScrollButtons:

```
- (void)trackScrollButtons:(NSEvent *)theEvent
```

Invoked in response to mouse-down events on buttons. Tracks the scroll buttons and sends action messages to the NSScroller's target. This method is invoked when the NSScroller receives a mouse-down event in a scroll button. You should not invoke this method directly. See also trackKnob:.

usableParts

```
- (NSUsableScrollerParts)usableParts
```

Indicates which parts of the scroller can be displayed, given the NSScroller's current size. An NSScroller can be made too small for all its parts to be displayed. The usableParts method returns one of the following constants to indicate whether such a condition is present:

Table 1-23 Constants Indicating Which Parts of a Scroller Are Usable

Constant	Meaning
NSNoScrollerParts	Sroller has no usable parts, only the bar.
NSOnlyScrollerArrows	Scroller has only scroll buttons.
NSAllScrollerParts	Scroller has all parts.

NSScrollView

Inherits From:	NSView : NSResponder : NSObject
Conforms To:	NSCoding (NSResponder) NSObject (NSObject)
Declared In:	AppKit/NSScrollView.h

Class Description

An NSScrollView object lets the user interact with a document that's too large to be shown in its entirety within an NSView and must therefore be scrolled. The responsibility of an NSScrollView is to coordinate scrolling behavior between NSScroller objects and a NSClipView object. The user can drag the knob of an NSScroller and the NSScrollView will send a message to its NSClipView to ensure that the viewed portion of the document reflects

the position of the knob. Similarly, the application can change the viewed position within a document and the NSScrollView will send a message to the NSScrollers advising them of this change.

The NSScrollView has at least one subview (an NSClipView object), which is called the *content view*. The content view in turn has a subview called the *document view*, which is the view to be scrolled. When an NSScrollView is created, it has neither a vertical nor a horizontal scroller. If NSScrollers are required, the application must send setHasHorizontalScroller:YES and setHasVerticalScroller:YES messages to the NSScrollView; the content view is resized to fill the area of the NSScrollView not occupied by the NSScrollers.

When the application modifies the scroll position within the document, it should send a reflectScrolledClipView: message to the NSScrollView, which will then query the content view and set the NSScroller(s) accordingly. The reflectScrolledClipView: message may also cause the NSScrollView to enable or disable the NSScrollers as required.

Method Types

Activity	Class Method
Determining component sizes	– contentSize – documentVisibleRect
Laying out the NSScrollView	+ contentSizeForFrameSize:hasHorizontalScroller: hasVerticalScroller:borderType: + frameSizeForContentSize:hasHorizontalScroller: hasVerticalScroller:borderType: – hasHorizontalScroller – hasVerticalScroller – isRulerVisible – setHasHorizontalScroller: – setHasVerticalScroller: – tile – toggleRuler:
Managing component views	– contentView – documentView – horizontalScroller – reflectScrolledClipView: – setContentView: – setDocumentView: – setHorizontalScroller: – setVerticalScroller: – verticalScroller
Modifying graphic attributes	– backgroundColor – borderType – setBackgroundColor: – setBorderType:
Setting scrolling behavior	– lineScroll – pageScroll – scrollsDynamically – setLineScroll: – setPageScroll: – setScrollsDynamically:
Managing the cursor	– documentCursor – setDocumentCursor:

Class Methods

contentSizeForFrameSize:hasHorizontalScroller:hasVerticalScroller:borderType:

```
+ (NSSize)contentSizeForFrameSize:(NSSize)size
    hasHorizontalScroller:(BOOL)horizFlag
    hasVerticalScroller:(BOOL)vertFlag
    borderType:(NSBorderType)aType
```

Calculates and returns the size of a content view for an `NSScrollView` with frame size `size`. `horizFlag` is `YES` if the `NSScrollView` has a horizontal scroller, and `vertFlag` is `YES` if it has a vertical scroller. `aType` indicates whether there's a line, a bezel, groove, or no border around the frame of the `NSScrollView`, and is either `NSLineBorder`, `NSBezelBorder` (the default), `NSGrooveBorder`, or `NSNoBorder`. If the `NSScrollView` object already exists, you can send it a `contentSize:` message to get the size of its content view. See also `frameSizeForContentSize:hasHorizontalScroller:hasVerticalScroller:borderType:`.

frameSizeForContentSize:hasHorizontalScroller:hasVerticalScroller:borderType:

```
+ (NSSize)frameSizeForContentSize:(NSSize)size
    hasHorizontalScroller:(BOOL)horizFlag
    hasVerticalScroller:(BOOL)vertFlag
    borderType:(NSBorderType)aType
```

Calculates and returns the frame size required for an `NSScrollView` with a content view size `size`. `horizFlag` is `YES` if the `NSScrollView` has a horizontal scroller, and `vertFlag` is `YES` if it has a vertical scroller. `aType` indicates whether there's a line, a bezel, groove, or no border around the frame of the `NSScrollView`, and is either `NSLineBorder`, `NSBezelBorder`, `NSGrooveBorder`, or `NSNoBorder`. See also `contentSizeForFrameSize:hasHorizontalScroller:hasVerticalScroller:borderType:`.

Instance Methods

backgroundColor

```
- (NSColor *)backgroundColor
```

Returns the content view background color.

borderType

```
- (NSBorderType)borderType
```

Returns the `NSScrollView` border type. Border types are:

- `NSLineBorder`
- `NSBezelBorder`
- `NSGrooveBorder`
- `NSNoBorder`

See also `setBorderType:`.

contentSize

```
- (NSSize)contentSize
```

Returns the content view's size, in the `NSScollView`'s superview coordinates. See also `documentVisibleRect`.

contentView

```
- (NSClipView *)contentView
```

Returns the scroll view's content view. See the `NSScrollView` class description for a brief description of content views. See also `setContentView:`.

documentCursor

```
- (NSCursor)documentCursor
```

Returns the cursor object used inside the document view. See also `setDocumentCursor:`.

documentView

```
- (id)documentView
```

Returns the current document view. See also `setDocumentView:`.

documentVisibleRect

```
- (NSRect)documentVisibleRect
```

Gets the visible portion of the document view. See also `contentSize`.

hasHorizontalScroller

```
- (BOOL)hasHorizontalScroller
```

Returns `YES` if the `NSScrollView` object has a horizontal scroller. See also `setHasHorizontalScroller:`, `horizontalScroller`, `setHorizontalScroller:`, `hasVerticalScroller`.

hasVerticalScroller

```
- (BOOL)hasVerticalScroller
```

Returns `YES` if the `NSScrollView` object has a vertical scroller. See also `setHasVerticalScroller:`, `verticalScroller`, `setVerticalScroller:`, `hasHorizontalScroller`.

horizontalScroller

```
- (NSScroller *)horizontalScroller
```

Returns the horizontal `NSScroller` object. See also `setHorizontalScroller:`, `verticalScroller`, `hasHorizontalScroller`.

isRulerVisible

```
- (BOOL)isRulerVisible
```

Returns `YES` if the `NSScrollView` ruler is visible. See also `toggleRuler:`.

lineScroll

```
- (float)lineScroll
```

Returns the amount scrolled when scrolling a line. The return value is expressed in the NSScrollView's coordinate system units. See also setLineScroll:, pageScroll, scrollsDynamically.

pageScroll

```
- (float)pageScroll
```

Returns the amount scrolled when scrolling a page. The return value is expressed in the NSScrollView's coordinate system units. See also setPageScroll:, lineScroll, scrollsDynamically.

reflectScrolledClipView:

```
- (void)reflectScrolledClipView:(NSClipView *)cView
```

Moves the scrollers to reflect change in the coordinates of the clip view.

scrollsDynamically

```
- (BOOL)scrollsDynamically
```

Returns whether the NSScrollView scrolls dynamically. See also setScrollsDynamically:, lineScroll, pageScroll.

setBackgroundColor:

```
- (void)setBackgroundColor:(NSColor *)color
```

Sets the content view's background color. See also backgroundColor.

setBorderType:

```
- (void)setBorderType:(NSBorderType)aType
```

Sets the NSScrollView border type. Border types are:

- NSLineBorder
- NSBezelBorder

- NSGrooveBorder
- NSNoBorder

See also borderType.

setContentView:

```
- (void)setContentView:(NSClipView *)contentView
```

Sets the scroll view's content view. See the NSScrollView class description for a brief description of content views. See also contentView.

setDocumentCursor:

```
– (void)setDocumentCursor:(NSCursor *)anObject
```

Sets the cursor object to be used inside the document view. See also documentCursor.

setDocumentView:

```
– (void)setDocumentView:(NSView *)aView
```

Makes aView the NSScrollView's document view. See also documentView.

setHasHorizontalScroller:

```
– (void)setHasHorizontalScroller:(BOOL)flag
```

Adds or removes a horizontal scroller for the NSScrollView. If flag is YES, the NSScrollView creates a new NSScroller, and shrinks its other subviews to accommodate it. If flag is NO, the NSScroller is removed from the NSScrollView and the other subviews are resized to fill the NSScrollView. When an NSScrollView is created, it doesn't have a horizontal scroller. Once an NSScroller is added, it will be enabled and disabled automatically by the NSScrollView. This method retiles and redisplays the NSScrollView. See also setHasVerticalScroller:.

setHasVerticalScroller:

```
– (void)setHasVerticalScroller:(BOOL)flag
```

Adds or removes a vertical scroller to the NSScrollView. If flag is YES, the NSScrollView creates a new NSScroller, and shrinks its other subviews to accommodate it. If flag is NO, the NSScroller is removed from the NSScrollView and the other subviews are resized to fill the NSScrollView. When an NSScrollView is created, it doesn't have a vertical scroller. Once an NSScroller is added, it will be enabled and disabled automatically by the NSScrollView. This method retiles and redisplays the NSScrollView. See also setHasHorizontalScroller:.

setHorizontalScroller:

```
- (void)setHorizontalScroller:(NSScroller *)anObject
```

Sets the horizontal scroller to anObject which should be an NSScroller subclass instance. This method sets anObject's target to the NSScrollView and sets anObject's action method to the NSScrollView's private method that responds to the NSScrollers and invokes the appropriate scrolling behavior. To make the scroller visible, you must send a setHasHorizontalScroller:YES message to the NSScrollView.

setLineScroll:

```
- (void)setLineScroll:(float)value
```

Sets the amount to scroll the document view when the NSScrollView receives a message to scroll one line. value is expressed in the content view's coordinates. See also lineScroll, setPageScroll:, setScrollsDynamically:.

setPageScroll:

```
- (void)setPageScroll:(float)value
```

Sets the amount to scroll the document view when the NSScrollView receives a message to scroll one page. value is the amount of text common to the content view before and after the page scroll and is expressed in the content view's coordinates. Therefore, setting value to 0.0 implies that the entire content view is replaced when a page scroll occurs. See also pageScroll, setLineScroll:, setScrollsDynamically:.

setScrollsDynamically:

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

Determines whether dragging a scroller's knob will result in dynamic redisplay of the document. If `flag` is `YES`, scrolling will occur as the knob is dragged. If `flag` is `NO`, scrolling will occur only after the knob is released. By default, scrolling occurs as the knob is dragged. See also `scrollsDynamically`, `setLineScroll:`, `setPageScroll:`.

setVerticalScroller:

```
- (void)setVerticalScroller:(NSScroller *)anObject
```

Sets the vertical scroller to `anObject` (which should be an `NSScroller` subclass instance). This method sets `anObject`'s target to the `NSScrollView` and sets `anObject`'s action method to the `NSScrollView`'s private method that responds to the `NSScroller`s and invokes the appropriate scrolling behavior. To make the scroller visible, you must send a `setHasVerticalScroller:YES` message to the `NSScrollView`. See also `verticalScroller`, `setHorizontalScroller:`.

tile

```
- (void)tile
```

Determines `NSScrollView` layout by setting the sizes and locations of the object's subviews. You rarely send a `tile` message directly; you may override it if you need to have the `NSScrollView` manage additional views. A `tile` message is sent whenever the `NSScrollView` is resized, or a vertical or horizontal scroller is added or removed. This method *doesn't* redisplay the `NSScrollView`.

toggleRuler:

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

Makes the ruler visible or invisible, whichever is the opposite of its current state. See also `isRulerVisible`.

verticalScroller

```
- (NSScroller *)verticalScroller
```

Returns the vertical `NSScroller` object. See also `setVerticalScroller:`, `horizontalScroller`.

NSSelection

Inherits From:	NSObject
Conforms To:	NSCoding, NSCopying NSObject (NSObject)
Declared In:	AppKit / NSSelection.h

Class Description

The `NSSelection` class defines an object that describes a selection within a document. An `NSSelection`, or simply, selection, is an immutable description; it may be held by the system or other documents, and it cannot change over time. Selections are typically used by `NSDataLink` objects to represent the source and destination of a link.

Because a selection description can't be changed once it's been exported, it's a good idea to construct general descriptions that can survive changes to a document and don't require selection-specific information to be stored in the document. This description may be simple or complex, depending upon the application. For example, a painting application might describe a selection in an image as a simple rectangle. This description doesn't require that any information be stored in the image's file, and the description can be expected to remain valid through the life of the image. An object-based drawing application might describe a selection as a list of object identifiers (though *not* `id`s), where an object identifier is unique throughout the life of the document. Based on this list, a selection could be meaningfully reconstructed, even if new objects are added to the document or selected objects are deleted. Such a scheme doesn't require that any selection-specific information be stored in the document's file, with the benefit that links can be made to read-only documents.

Maintaining a character-range selection in a text document is more problematic. A possible solution is to insert selection-begin and selection-end markers that define a specific selection into the text stream. A selection description would then refer to a specific selection marker. This solution requires that selection state information be stored and maintained within the document. Furthermore, this information generally shouldn't be purged from the document because the document can't know how many references to the selection exist. (References to the selection could be stored with documents on removable media, like floppy disks.) This selection-state information should be maintained as long as it refers to any meaningful data. For this reason, it is desirable whenever possible to describe selection in a manner that doesn't require that selection-state information be maintained in the document.

Three well-known selection descriptions can apply to any document: the empty selection, the entire document, and the abstract concept of the current selection. `NSSelection` objects for these selections are returned by the `emptySelection`, `allSelection`, and `currentSelection` class methods.

Since an `NSSelection` may be used in a document that is read by machines with different architectures, care should be taken to write machine-independent descriptions. For example, using a binary structure as a selection description will fail on a machine where an identically defined structure has a different size or is kept in memory with different byte ordering. Exporting (and then parsing) ASCII descriptions is often a good solution. If binary descriptions must be used, it's prudent to preface the description with a token specifying the description's byte ordering.

It may also be prudent to version-stamp selection descriptions, so that old selections can be accurately read by updated versions of an application.

Note – `NSSelection` is not part of the OpenStep specification.

Method Types

Activity	Class Method
Returning special selection shared instances	+ allSelection + currentSelection + emptySelection
Describing a selection	– descriptionData – isWellKnownSelection

Class Methods

allSelection

```
+ (NSSelection *)allSelection
```

Returns the shared instance of the well-known selection representing the entire document. See also `currentSelection`, `emptySelection`.

currentSelection

```
+ (NSSelection *)currentSelection
```

Returns the shared instance of the well-known selection representing the abstract concept of the current selection. The current selection never describes a specific selection; it describes a selection that may change frequently. See also `allSelection`, `emptySelection`.

emptySelection

```
+ (NSSelection *)emptySelection
```

Returns the shared instance of the well-known selection representing no data. See also `allSelection`, `currentSelection`.

Instance Methods

descriptionData

```
– (NSData *)descriptionData
```

Returns the data that describes the selection as set by `selectionWithDescriptionData:` or `initWithDescriptionData:`.

`isWellKnownSelection`

`- (BOOL)isWellKnownSelection`

Returns `YES` if the receiver is one of the well-known selection types (those representing the entire document, current selection, or empty selection) and returns `NO` otherwise. See also `allSelection`, `currentSelection`, `emptySelection`.

NSSlider

Inherits From:	NSControl : NSView : NSResponder : NSObject
Conforms To:	NSCoding (NSResponder) NSObject (NSObject)
Declared In:	AppKit/NSSlider.h

Class Description

`NSSlider` is a type of `NSControl` with a sliding knob that can be moved to represent a value between a minimum and a maximum setting. A slider may be either horizontal or vertical, but its minimum value is always at the left or bottom end of the bar, and the maximum at the right or top. By default, an `NSSlider` is a continuous `NSControl`: It sends its action message to its target continuously while the user drags its knob. To configure an NSSlider to send its action only when the mouse is released, send `setContinuous:` (an `NSControl` method) with an argument of `NO`.

An `NSSlider` can be configured to display an image, a title, or both, in the area behind its knob. An `NSSlider`'s title can be drawn in any gray level or color, and in any font available. An `NSSlider`'s value can be set programmatically with any of the standard `NSControl` value-setting methods, such as `setFloatValue:`. For more information, see the method descriptions in the `NSSliderCell` class specification.

Method Types

Activity	Class Method
Setting the cell class	+ cellClass + setCellClass:
Modifying an NSSlider's appearance	– image – isVertical – knobThickness – setImage: – setKnobThickness: – setTitle: – setTitleCell: – setTitleColor: – setTitleFont: – title – titleCell – titleColor – titleFont
Setting and getting value limits	– maxValue – minValue – setMaxValue: – setMinValue:
Handling events	– acceptsFirstMouse:

Class Methods

cellClass

```
+ (Class)cellClass
```

Returns the class last set in a `setCellClass:` message, or the `NSSliderCell` class if `setCellClass:` has never been called.

setCellClass:

```
+ (void)setCellClass:(Class)classId
```

Configures the NSSlider class to use instances of classId for its cells. classId should be an NSSliderCell subclass id, obtained by sending the class message (NSObject) to either the NSSliderCell subclass object or to an instance of that subclass. The default NSCell class is NSSliderCell.

If this method isn't overridden by a NSSlider subclass, then when it's sent to that subclass, NSSlider and any other subclasses of NSSlider that don't override the methods mentioned the "Instance Methods" section will use the new NSCell subclass as well. To safely set an NSCell class for your subclass of NSSlider, override this method to store the NSCell class in a static id. Also, override the designated initializer to replace the NSSlider subclass instance's NSCell with an instance of the NSCell subclass stored in that static id. See "Creating New NSControls" in the NSControl class specification for more information.

Instance Methods

acceptsFirstMouse:

```
- (BOOL)acceptsFirstMouse:(NSEvent *)theEvent
```

Returns YES by default, since NSSliders always accept a mouse-down event that activates a window, whether or not the NSSlider is enabled. Override this if you want different behavior.

image

```
- (NSImage *)image
```

Returns the NSImage displayed within the NSSlider bar. See also setImage:.

isVertical

```
- (int)isVertical
```

Returns 1 if the NSSlider is vertical, 0 if horizontal, and -1 if unknown (because the slider hasn't been initialized). A slider is vertical if its height is greater than its width.

knobThickness

```
- (float)knobThickness
```

Returns the knob's thickness as a float value (width if horizontal slider, height if vertical slider) in the slider's coordinate system. See also `setKnobThickness:`.

maxValue

```
- (double)maxValue
```

Returns the `NSSlider`'s maximum value. See also `setMaxValue:`, `minValue`.

minValue

```
- (double)minValue
```

Returns the `NSSlider`'s minimum value. See also `setMinValue:`, `maxValue`.

setImage:

```
- (void)setImage:(NSImage *)backgroundImage
```

Sets the image used as the slider's bar to `backgroundImage`. See also `image`.

setKnobThickness:

```
- (void)setKnobThickness:(float)aFloat
```

Sets the knob's thickness (its width if the slider is horizontal, height if vertical) to `aFloat`, expressed in units of the `NSSlider`'s coordinate system. See also `knobThickness`.

setMaxValue:

```
- (void)setMaxValue:(double)aDouble
```

Sets the `NSSlider`'s maximum value to `aDouble`. See also `maxValue`, `minValue`.

setMinValue:

```
- (void)setMinValue:(double)aDouble
```

Sets the NSSlider's minimum value to aDouble. See also minValue, maxValue.

setTitle:

```
- (void)setTitle:(NSString *)aString
```

Sets the title within the NSSlider to a copy of aString. See also title, setTitleCell:, setTitleColor:, setTitleFont:.

setTitleCell:

```
- (void)setTitleCell:(NSCell *)aCell
```

Sets the NSCell (or subclass) object used to draw the NSSlider's title. The cell object should ideally be an instance of NSTextFieldCell or one of its subclasses. Doesn't redraw the slider; a setTitle: message is required to display a title, even if aCell already has a string value. See also titleCell, setTitleColor:, setTitleFont:.

setTitleColor:

```
- (void)setTitleColor:(NSColor *)aColor
```

Sets the color of text in the title to aColor. See also titleColor, setTitleFont:, setTitleCell:.

setTitleFont:

```
- (void)setTitleFont:(NSFont *)fontObject
```

Sets the NSFont object used for the slider title. See also titleFont, setTitleColor:, setTitleCell:.

title

```
- (NSString *)title
```

Returns the NSSlider title. See also titleFont, titleCell, titleColor.

titleCell

```
- (id)titleCell
```

Returns the NSCell (or subclass thereof) object used to draw the title within the NSSlider. If the slider doesn't have a title, an new NSTextFieldCell is created and returned. See title, titleFont, titleColor.

titleColor

```
- (NSColor *)titleColor
```

Returns the color of text in the title. See also title, titleCell, titleFont.

titleFont

```
- (NSFont *)titleFont
```

Returns the NSFont object used in drawing the title within the NSSlider. See also title, titleCell, titleColor.

NSSliderCell

Inherits From:	NSActionCell : NSCell : NSObject
Conforms To:	NSCoding, NSCopying (NSCell) NSObject (NSObject)
Declared In:	AppKit/NSSliderCell.h

Class Description

NSSliderCell is a type of NSCell used to assist the NSSlider class, and to build matrices of sliders. The NSSliderCell encompasses all the visible portions of the NSSlider—the knob, the area along which the knob slides, and the optional title within this area. See the NSSlider class specification for an overview of how NSSliderCells work.

Method Types

Activity	Class Method
Determining component sizes	– cellSizeForBounds: – knobRectFlipped:
Setting value limits	– maxValue – minValue – setMaxValue: – setMinValue:
Modifying graphic attributes	– isVertical – knobThickness – setKnobThickness: – setTitle: – setTitleCell: – setTitleColor: – setTitleFont: – title – titleCell – titleColor – titleFont
Displaying the NSSliderCell	– drawBarInside:flipped: – drawKnob – drawKnob:
Modifying behavior	– altIncrementValue – setAltIncrementValue:
Tracking the mouse	+ prefersTrackingUntilMouseUp – trackRect

Class Methods

prefersTrackingUntilMouseUp

```
+ (BOOL)prefersTrackingUntilMouseUp
```

Returns `YES` to allow `NSSliderCell` objects to track even when the mouse leaves the cell bounds. This ensures that an `NSSliderCell` in a `NSMatrix` doesn't stop responding to user input (and its neighbor start responding) just because the knob isn't dragged in a perfectly straight line. Override this method to return `NO` if you want the `NSSliderCell` to stop tracking once the mouse leaves its bounds.

.Instance Methods

altIncrementValue

```
- (double)altIncrementValue
```

Returns the amount that the NSSliderCell will alter its value when the user drags the knob one pixel with the Alternate key held down. If the Alternate-dragging feature isn't enabled, this method returns –1.0. See also setAltIncrementValue:.

cellSizeForBounds:

```
- (NSSize)cellSizeForBounds:(NSRect)aRect
```

Returns the minimum width and height needed to draw the NSSliderCell in aRect. If aRect is too small to fit the knob and bezel, aRect's dimensions are set to 0.0. If the NSSliderCell hasn't had its tracking rectangle set, this method will set it. If you draw your own knob on the NSSliderCell and that knob is not the same size as a standard NSSliderCell knob, or if you draw the NSSliderCell itself differently, you should override this method to take your knob's dimensions into account. You must also override knobRectflipped: and drawKnob:.

drawBarInside:flipped:

```
- (void)drawBarInside:(NSRect)aRect flipped:(BOOL)flipped
```

Draws the NSSliderCell's background bar (but not the bezel around it or the knob) in aRect. flipped indicates whether the NSView's coordinate system is flipped. Override this method if you want to draw your own slider bar. See also drawKnob, drawKnob:.

drawKnob

```
- (void)drawKnob
```

Calculates the knobs drawing rectangle, and invokes drawKnob: to actually draw the knob. The PostScript focus must be locked on the NSSliderCell's NSView when this message is sent. Do not override this method; override drawKnob: instead. See also drawBarInside:flipped:.

drawKnob:

```
- (void)drawKnob:(NSRect)knobRect
```

Draws the knob in `knobRect`. The PostScript focus must be locked on the `NSSliderCell`'s `NSView` when this message is sent. Override this method and `knobRectFlipped:` if you want to draw your own knob. You should also override `cellSizeForBounds:` if your knob is of a different size from the standard `NSSliderCell` knob.

isVertical

```
- (int)isVertical
```

Returns 1 if the `NSSliderCell` is vertical, 0 if horizontal. Returns -1 if the orientation can't be determined (for example, if the `NSSliderCell` hasn't been drawn in an `NSView`). An `NSSliderCell` is vertical if its height is greater than its width.

knobRectFlipped:

```
- (NSRect)knobRectFlipped:(BOOL)flipped
```

Gets the rectangle the knob will be drawn in. `flipped` indicates whether the `NSSliderCell`'s view has a flipped coordinate system. This rectangle is determined from the `NSSliderCell`'s value in relation to its tracking rectangle and its minimum and maximum values. Override this method and `drawKnob:` if you want to draw your own knob. You should also override `cellSizeForBounds:` if your knob is of a different size from the standard `NSSliderCell` knob (and be careful of setting the knob's width). Remember to take into account the flipping of the `NSView` in vertical `NSSliderCell`s; otherwise, your knob might appear the correct distance from the wrong end. See also `knobThickness`.

knobThickness

```
- (float)knobThickness
```

Returns the `NSSliderCell`'s knob thickness (that is, its extent along the bar's length) in the `NSSliderCell`'s coordinate system. See also `setKnobThickness:`, `knobRectFlipped:`, `drawKnob`, `drawKnob:`.

maxValue

```
- (double)maxValue
```

Returns the NSSliderCell's maximum value. See also setMaxValue:, minValue.

minValue

```
- (double)minValue
```

Returns the NSSliderCell's minimum value. See also setMinValue:.

setAltIncrementValue:

```
- (void)setAltIncrementValue:(double)incValue
```

Sets the amount by which the NSSliderCell modifies its value when the knob is dragged one pixel with the Alternate key held down. incValue should be greater than 0.0, and less than the NSSliderCell's maximum value (maxValue); it can also be –1, in which case this feature is disabled. Normally, you'll want to use this method with incValue less than 1.0, so the knob will move more slowly than the mouse. See also altIncrementValue.

setKnobThickness:

```
- (void)setKnobThickness:(float)aFloat
```

Sets the NSSliderCell's knob thickness (width if a horizontal slider, height if vertical) in its own coordinate system. aFloat must be greater than 0.0, and shouldn't be greater than the slider's length. If the knob thickness changes, the NSSliderCell's inside is redrawn. See also knobThickness.

setMaxValue:

```
- (void)setMaxValue:(double)aDouble
```

Sets the slider cell's maximum value to aDouble. If the maximum value changes, the slider cell's inside is redrawn to reposition the knob relative to the new maximum. See also maxValue, setMinValue:.

setMinValue:

```
- (void)setMinValue:(double)aDouble
```

Sets the slider cell's minimum value to `aDouble`. If the minimum value changes, the slider cell's inside is redrawn to reposition the knob relative to the new minimum. See also `minValue`, `setMinValue:`.

setTitle:

```
- (void)setTitle:(NSString *)aString
```

Sets the title within the slider cell to a copy of `aString`. See also `title`, `setTitleColor:`, `titleColor`, `setTitleFont:`, `titleFont`, `setTitleCell:`, `titleCell`.

setTitleCell:

```
- (void)setTitleCell:(NSCell *)aCell
```

Sets the `NSCell` (or subclass thereof) object used to draw the title within the `NSSliderCell`. The cell object should ideally be an instance of `NSTextFieldCell` or one of its subclasses. See also `titleCell`, `setTitle:`.

setTitleColor:

```
- (void)setTitleColor:(NSColor *)aColor
```

Sets the color of text in the title to `aColor`, and redraws the slider cell's inside. See also `titleColor`, `setTitle:`.

setTitleFont:

```
- (void)setTitleFont:(NSFont *)fontObject
```

Sets the `NSFont` object used to draw the title within the slider cell, and redraws the slider cell's inside. The default font is the default system font as set by the user with the Preferences application, and its size is 12.0 point. See also `titleFont`, `setTitle:`.

title

```
- (NSString *)title
```

Returns the title within the slider cell. See also `setTitle:`.

titleCell

```
- (id)titleCell
```

Returns the `NSCell` (or subclass thereof) object used to draw the title within the slider cell. If the slider cell doesn't have a title, a new `NSTextFieldCell` is created and returned. This doesn't result in a title getting set. See also `setTitleCell:`, `setTitle:`.

titleColor

```
- (NSColor *)titleColor
```

Returns the color of text in the title. See also `setTitleColor:`, `setTitle:`.

titleFont

```
- (NSFont *)titleFont
```

Returns the `NSFont` object used in drawing the title within the slider cell. See also `setTitleFont:`, `setTitle:`.

trackRect

```
- (NSRect)trackRect
```

Returns the rectangle used in tracking the mouse (only valid while tracking). See also `prefersTrackingUntilMouseUp`.

NSSpellChecker

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

Class Description

The `NSSpellChecker` class gives any application an interface to the OpenStep spell-checking service. To handle all its spell checking, an application needs only one instance of `NSSpellChecker`. It provides a panel in which the user can specify decisions about words that are suspect. To check the spelling of a piece of text, the application:

- Includes in its user interface a menu item (or a button or command) by which the user will request spell checking.
- Makes the text available by way of an `NSString` object.
- Creates an instance of the `NSSpellChecker` class and sends it a `checkSpellingOfString:startingAt:` message.

For example, you might use the following statement to create an `NSSpellChecker`:

```
range = [[NSSpellChecker sharedSpellChecker]
    checkSpellingOfString:aString startingAt:0];
```

The `checkSpellingOfString:startingAt:` method checks the spelling of the words in the specified string beginning at the specified offset until it finds a word that is misspelled (this example uses 0 to start at the beginning of the string) . Then it returns an `NSRange` to indicate the location of the misspelled word.

In a graphical application, whenever a misspelled word is found, you'll probably want to highlight the word in the document, using the `NSRange` that `checkSpellingOfString:startingAt:` returned to determine the text to highlight. Then you should show the misspelled word in the Spelling panel's misspelled-word field by calling `updateSpellingPanelWithMisspelledWord:`. If `checkSpellingOfString:startingAt:` does not find a misspelled word, you should call `updateSpellingPanelWithMisspelledWord:` with the

empty string. This causes the system to beep, letting the user know that the spell check is complete and no misspelled words were found. None of these steps is required, but if you do one, you should do them all.

The object that provides the string being checked should adopt the following protocols.

Protocol	Meaning
NSChangeSpelling	A message in this protocol (changeSpelling:) is sent down the responder chain when the user presses the Correct button.
NSIgnoreMisspelledWords	When the object being checked responds to this protocol, the spell server keeps a list of words that are acceptable in the document and enables the Ignore button in the Spelling panel.

The application may choose to split a document's text into segments and check them separately. This will be necessary when the text has segments in different languages. Spell checking is invoked for one language at a time, so a document that contains portions in three languages will require at least three checks.

Dictionaries and Word Lists

The process of checking spelling makes use of three references:

- A dictionary registered with the system's spell-checking service. When the Spelling panel first appears, by default it shows the dictionary for the user's preferred language. The user may select a different dictionary from the list in the Spelling panel.
- The user's "learn" list of correctly-spelled words in the current language. The `NSSpellChecker` updates the list when the user presses the Learn or Forget buttons in the Spelling panel.
- The document's list of words to be ignored while checking it (if the first responder conforms to the `NSIgnoreMisspelledWords` protocol). The `NSSpellChecker` updates its copy of this list when the user presses the Ignore button in the Spelling panel.

A word is considered to be misspelled if none of these three accepts it.

Matching a List of Ignored Words to the Document It Belongs To

The `NSString` being checked isn't the same as the document. In the course of processing a document, an application might run several checks based on different parts or different versions of the text. But they'd all belong to the same document. The `NSSpellChecker` keeps a separate "ignored words" list for each document that it checks. To help match "ignored words" lists to documents, you should call `uniqueSpellDocumentTag` once for each document. This method returns a unique arbitrary integer that will serve to distinguish one document from the others being checked and to match each "ignored words" list to a document. When searching for misspelled words, pass the tag as the fourth argument of `checkSpellingOfString:startingAt:language:wrap:inSpellDocumentWithTag:wordCount:`. The convenience method `checkSpellingOfString:startingAt:` takes no tag. This method is suitable when the first responder does not conform to the `NSIgnoreMisspelledWords` protocol.

When the application saves a document, it may choose to retrieve the "ignored words" list and save it along with the document. To get back the right list, it must send the `NSSpellChecker` an `ignoredWordsInSpellDocumentWithTag:` message. When the application has closed a document, it should notify the `NSSpellChecker` that the document's "ignored words" list can now be discarded, by sending it a `closeSpellDocumentWithTag:` message. When the application reopens the document, it should restore the "ignored words" list with the message `setIgnoredWords:inSpellDocumentWithTag:`.

Method Types

Activity	Class Method
Making a checker available	+ sharedSpellChecker + sharedSpellCheckerExists
Managing the spelling panel	– accessoryView – setAccessoryView: – spellingPanel
Checking spelling	– countWordsInString:language: – checkSpellingOfString:startingAt: – checkSpellingOfString:startingAt:language: wrap:inSpellDocumentWithTag:wordCount:
Setting the language	– language – setLanguage:
Managing the spelling process	– uniqueSpellDocumentTag – closeSpellDocumentWithTag: – ignoreWord:inSpellDocumentWithTag: – ignoredWordsInSpellDocumentWithTag: – setIgnoredWords:inSpellDocumentWithTag: – setWordFieldStringValue: – updateSpellingPanelWithMisspelledWord:

Class Methods

sharedSpellChecker

```
+ (NSSpellChecker *)sharedSpellChecker
```

Returns the `NSSpellChecker` (one per application). If the application has not yet asked for an `NSSpellChecker` object, this method allocates and initializes a new instance. See also `sharedSpellCheckerExists`.

sharedSpellCheckerExists

```
+ (BOOL)sharedSpellCheckerExists
```

Returns `YES` if the application's `NSSpellChecker` has already been created. See also `sharedSpellChecker`.

uniqueSpellDocumentTag

```
+ (int)uniqueSpellDocumentTag
```

Returns a guaranteed unique tag to use as the spell-document tag for a document. Use this method to generate tags to avoid collisions with other objects that can be spell-checked.

Instance Methods

accessoryView

```
- (NSView *)accessoryView
```

Returns the Spelling panel's accessory `NSView` object. See also `setAccessoryView:`.

checkSpellingOfString:startingAt:

```
- (NSRange)checkSpellingOfString:(NSString *)stringToCheck
    startingAt:(int)startingOffset
```

Starts the search for a misspelled word in `stringToCheck` starting at `startingOffset` within the string object. Returns the range of the first misspelled word. Wrapping occurs but no ignored-words dictionary is used. See also `checkSpellingOfString:startingAt:language:wrap:inSpellDocumentWithTag:wordCount:`.

checkSpellingOfString:startingAt:language:wrap:inSpellDocumentWithTag:wordCount:

```
- (NSRange)checkSpellingOfString:(NSString *)stringToCheck
    startingAt:(int)startingOffset language:(NSString *)language
     wrap:(BOOL)wrapFlag inSpellDocumentWithTag:(int)tag
     wordCount:(int *)wordCount
```

Starts the search for a misspelled word in `stringToCheck` starting at `startingOffset` within the string object. Returns the range of the first misspelled word and optionally the word count by reference. `tag` is an identifier unique within the application used to inform the spell check which document (actually, a dictionary) of ignored words to use. `wrapFlag` determines whether spell checking continues at the beginning of the string

when the end is reached. `language` is the language used in the string. If `language` is the empty string, the current selection in the Spelling panel's pop-up menu is used. See also `checkSpellingOfString:startingAt:`.

`closeSpellDocumentWithTag:`

```
- (void)closeSpellDocumentWithTag:(int)tag
```

Notifies the `NSSpellChecker` that the user has finished with the ignored-word document identified by `tag`, causing it to throw that dictionary away.

`countWordsInString:language:`

```
- (int)countWordsInString:(NSString *)aString
    language:(NSString *)language
```

Returns the number of words in `string`. The `language` argument specifies the language used in the string. If `language` is the empty string, the current selection in the Spelling panel's pop-up menu is used.

`ignoreWord:inSpellDocumentWithTag:`

```
- (void)ignoreWord:(NSString *)wordToIgnore
    inSpellDocumentWithTag:(int)tag
```

Instructs the `NSSpellChecker` to ignore all future occurrences of `wordToIgnore` in the document identified by `tag`. You should call this method from within your implementation of the `NSIgnoreMisspelledWords` protocol's `ignoreSpelling:`. See also `setIgnoredWords:inSpellDocumentWithTag:`, `ignoredWordsInSpellDocumentWithTag:`.

`ignoredWordsInSpellDocumentWithTag:`

```
- (NSArray *)ignoredWordsInSpellDocumentWithTag:(int)tag
```

Returns the array of ignored words for a document identified by `tag`. Invoke this before `closeSpellDocument:` if you want to store the ignored words. See also `setIgnoredWords:inSpellDocumentWithTag:`, `ignoreWord:inSpellDocumentWithTag:`.

language

```
- (NSString *)language
```

Returns the character string that identifies the English name of the currently selected language. If the application elects to temporarily override the current language (by invoking `setLanguage:`), this method will be useful to record the current language so that it can subsequently be restored. Otherwise, the application will not ordinarily need to use this method.See also `setLanguage:`.

setAccessoryView:

```
- (void)setAccessoryView:(NSView *)aView
```

Makes an `NSView` object an accessory of the Spelling panel by making it a subview of the panel's content view. This method posts the notification `NSWindowDidResizeNotification` with the Spelling panel object to the default notification center. An application can invoke this method to add controls that extend the panel's functions. The Spelling panel is automatically resized to accommodate `aView`. This method can be invoked repeatedly to change the accessory view depending on the situation. When `aView` is `nil`, the effect is to remove any accessory view that's already in the panel. See also `accessoryView`.

setIgnoredWords:inSpellDocumentWithTag:

```
- (void)setIgnoredWords:(NSArray *)someWords
    inSpellDocumentWithTag:(int)tag
```

Initializes the spell checker's list of acceptable words for the document identified by `tag`. `someWords` identifies the ignored words. `tag` identifies the document for which the list is being maintained (described in the section "Matching a List of Ignored Words to the Document It Belongs To"). See also `ignoredWordsInSpellDocumentWithTag:`.

setLanguage:

```
- (BOOL)setLanguage:(NSString *)aLanguage
```

Tells the NSSpellChecker object what language to use in subsequent spell check requests. This method is needed only if the application sometimes overrides the language established by the user's system defaults or by the user's choice of dictionary in the Spelling panel. A different dictionary might be required while checking a document whose text is distributed among several objects, each in a different language. Upon completion of the check in a different language, the application should restore the default. To do so, before using setLanguage:, use language to get the language previously in effect, and afterwards use setLanguage: to restore it.

Setting a different language causes corresponding changes to the selected dictionary and to the user's list of acceptable words. Suppose that checker is the id of an NSSpellChecker instance, and the application sends the following messages:

```
[checker setLanguage:"French"]
[checker checkSpelling:how of:textToBeChecked]
```

During the check invoked by the second message, the spelling system refers to the French dictionary rather than the dictionary selected in the Spelling panel, and adds "learned" words to the user's French word list. If aLanguage is NULL, this method sets the language to the first language for which there is a dictionary from the list of system languages. Returns YES if the Language pop-up list in the Spelling panel lists aLanguage. See also language.

setWordFieldStringValue:

```
- (void)setWordFieldStringValue:(NSString *)aString
```

Sets the string that appears in the misspelled word field, using the string object aString.

spellingPanel

```
- (NSPanel *)spellingPanel
```

Returns the NSSpellChecker's panel.

updateSpellingPanelWithMisspelledWord:

```
- (void)updateSpellingPanelWithMisspelledWord:(NSString *)word
```

Causes spell checker to update the Spelling panel's misspelled-word field to reflect `word`. You are responsible for highlighting `word` in the document and for extracting it from the document using the range returned by the `checkSpelling:...` methods. Pass the empty string as `word` to have the system beep, indicating no misspelled words were found.

NSSpellServer

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

Class Description

The `NSSpellServer` class gives you a way to make your particular spelling checker a service that's available to any application. A *service* is an application that declares its availability in a standard way, so that any other applications that wish to use it can do so. If you build a spelling checker that makes use of the `NSSpellServer` class and list it as an available service, then users of any application that makes use of `NSSpellChecker` or includes a Services menu will see your spelling checker as one of the available dictionaries.

To make use of `NSSpellServer`, you write a small program that creates an `NSSpellServer` instance and a delegate that responds to messages asking it to find a misspelled word and to suggest guesses for a misspelled word. Send the `NSSpellServer` `registerLanguage:byVendor:` messages to tell it the languages your delegate can handle.

The program that runs your spelling checker should not be built as an Application Kit application, but as a simple program. Suppose you supply spelling checkers under the vendor name "Acme." Suppose the file containing the code for your delegate is called `AcmeEnglishSpellChecker`. Then the following might be your program's `main` function:

```
void main()
{
  NSSpellServer *aServer = [[NSSpellServer alloc] init];
  if ([aServer registerLanguage:@"English" byVendor:@"Acme"]) {
    [aServer setDelegate:[AcmeEnglishSpellChecker alloc] init]];
    [aServer run];
    fprintf(stderr, "Unexpected death of Acme SpellChecker!\n");
```

```
  } else {
    fprintf(stderr, "Unable to check in Acme SpellChecker.\n");
  }
}
```

Your delegate is an instance of a custom subclass. It is simplest to make it an `NSObject` subclass, but that's not a requirement. Given an `NSString`, your delegate must be able to find a misspelled word by implementing the method `spellServer:findMisspelledWordInString:language:wordCount:countOnly:`. Usually, this method also reports the number of words it has scanned, but that isn't mandatory.

Optionally, the delegate may also suggest corrections for misspelled words. It does so by implementing the method `spellServer:suggestGuessesForWord:inLanguage:`.

Service Availability Notice

When there's more than one spelling checker available, the user selects the one required. The application that requests a spelling check uses an `NSSpellChecker` object, and it provides a Spelling panel; in the panel there's a pop-up list of available spelling checkers. Your spelling checker appears in that list if it has a *service descriptor.*

A service descriptor is an entry in a text file called `services`. Usually it's located within the bundle that also contains your spelling checker's executable file. The bundle (or directory) that contains the services file must have a name ending in `.service` or `.app`. The system looks for service bundles in a standard set of directories.

A spell-checker service availability notice has a standard format, illustrated in the following example for the Acme spelling checker:

```
Spell Checker:  Acme
Language:  French
Language:  English
Executable:  franglais.daemon
```

The first line identifies the type of service; for a spelling checker, it must say "Spell Checker:" followed by your vendor name. The next line contains the English name of a language your spelling checker is prepared to check. (The language must be one your system recognizes.) If your program can check

more than one language, use an additional line for each additional language. The last line of a descriptor gives the name of the service's executable file. (It requires a complete path if it's in a different directory.)

If there's a service descriptor for your Acme spelling checker and also a service descriptor for the English checker provided by a vendor named Consolidated, a user looking at the Spelling panel's pop-up list would see:

```
English (Acme)
English (Consolidated)
French (Acme)
```

Illustrative Sequence of Messages to an NSSpellServer

The act of checking spelling usually involves the interplay of objects in two classes: the user application's `NSSpellChecker`, which responds to interactions with the user, and your spelling checker's `NSSpellServer`, which provides the application interface for your spelling checker. You can see the interaction between the two in the following list of steps involved in finding a misspelled word.

- The user of an application selects a menu item to request a spelling check. The application sends a message to its `NSSpellChecker` object. The `NSSpellChecker` in turn sends a corresponding message to the appropriate `NSSpellServer`.
- The `NSSpellServer` receives the message asking it to check the spelling of an `NSString`. It forwards the message to its delegate.
- The delegate searches for a misspelled word. If it finds one, it returns an `NSRange` identifying the word's location in the string.
- The `NSSpellServer` receives a message asking it to suggest guesses for the correct spelling of a misspelled word, and forwards the message to its delegate.
- The delegate returns a list of possible corrections, which the `NSSpellServer` in turn returns to the `NSSpellChecker` that initiated the request.
- The `NSSpellServer` doesn't know what the user does with the errors its delegate has found or with the guesses its delegate has proposed. Perhaps the user corrects the document, perhaps by selecting a correction from the `NSSpellChecker`'s display of guesses; but that's not the `NSSpellServer`'s responsibility. However, if the user presses the Learn or

Forget buttons, causing the NSSpellChecker to revise the user's word list, the NSSpellServer receives a notification of the word thus learned or forgotten. It's up to you whether your spell checker acts on this information. If the user presses the Ignore button, the delegate is not notified (but the next time that word occurs in the text, the method isWordInUserDictionaries:caseSensitive: will report YES rather than NO).

- Once the NSSpellServer delegate has reported a misspelled word, it has completed its search. Of course, it's likely that the user's application will then send a new message, this time asking the NSSpellServer to check a string containing the part of the text it didn't get to earlier.

Method Types

Activity	Class Method
Checking in your service	– registerLanguage:byVendor:
Assigning a delegate	– delegate – setDelegate:
Running the service	– run
Checking user dictionaries	– isWordInUserDictionaries:caseSensitive:
Methods Implemented by the Delegate	– spellServer:didForgetWord:inLanguage: – spellServer:didLearnWord:inLanguage: – spellServer:findMisspelledWordInString: language:wordCount:countOnly: – spellServer:suggestGuessesForWord:inLanguage:

Instance Methods

delegate

```
- (id)delegate
```

Returns the NSSpellServer's delegate. See also setDelegate:.

isWordInUserDictionaries:caseSensitive:

```
(BOOL)isWordInUserDictionaries:(NSString *)word
   caseSensitive:(BOOL)flag
```

Returns whether word is in any open user dictionary; the search is case-sensitive if flag is YES.

registerLanguage:byVendor:

```
- (BOOL)registerLanguage:(NSString *)language
    byVendor:(NSString *)vendor
```

Notifies the spell server of a language your spelling checker can check. The argument language is the English name of a language. The argument vendor identifies the vendor (to distinguish your spelling checker from those that others may offer for the same language). If your spelling checker supports more than one language, it should invoke this method once for each language. Registering a language/vendor combination causes it to appear in the Spelling Panel's pop-up labeled "Dictionary". Returns YES when the language is registered, NO if for some reason it can't be registered.

run

```
- (void)run
```

Makes the spell server start listening for spell-checking requests. This method should not return.

setDelegate:

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

Sets the spell-server delegate. Since the delegate is where the real work is done, this is an essential step before your program sends the NSSpellServer its run message.

Methods Implemented by the Delegate

spellServer:didForgetWord:inLanguage:

```
- (void)spellServer:(NSSpellServer *)sender
    didForgetWord:(NSString *)word inLanguage:(NSString *)language
```

Notifies the delegate that the user has pressed Forget in an NSSpellChecker's Spelling Panel (and presumably the NSSpellChecker has removed word from the user's list of acceptable words). If the delegate maintains a similar auxiliary word list, it may wish to edit its list accordingly.

spellServer:didLearnWord:inLanguage:

```
- (void)spellServer:(NSSpellServer *)sender
    didLearnWord:(NSString *)word
    inLanguage:(NSString *)language
```

Notifies the delegate that the user has pressed Learn in an NSSpellChecker's Spelling Panel (and presumably the NSSpellChecker has added word to the user's list of acceptable words). If the delegate maintains a similar auxiliary word list, it may wish to edit it accordingly.

spellServer:findMisspelledWordInString:language:wordCount:countOnly:

```
- (NSRange)spellServer:(NSSpellServer *)sender
    findMisspelledWordInString:(NSString *)stringToCheck
    language:(NSString *)language wordCount:(int *)wordCount
    countOnly:(BOOL)countOnly
```

Search for a misspelled word in stringToCheck, using language, and marking the first misspelled word found by returning its range within the string object. wordCount returns by reference the number of words from the beginning of the string object until the misspelled word (or the end-of-string). If countOnly is YES, just count the words in the string object; do not spell-check. Send isWordInUserDictionaries:caseSensitive: to the spelling server to determine if word exists in the user's language dictionaries.

spellServer:suggestGuessesForWord:inLanguage:

```
- (NSArray *)spellServer:(NSSpellServer *)sender
    suggestGuessesForWord:(NSString *)word
    inLanguage:(NSString *)language
```

Search for alternatives to the misspelled word in language. Return guesses as an array of string objects.

NSSplitView

Inherits From:	NSView : NSResponder : NSObject
Conforms To:	NSCoding (NSResponder) NSObject (NSObject)
Declared In:	AppKit/NSSplitView.h

Class Description

An `NSSplitView` object lets several views share a region within a window. The `NSSplitView` resizes its subviews so that each subview is the same width as the `NSSplitView`, and the total of the subviews' heights is equal to the height of the `NSSplitView`. The `NSSplitView` positions its subviews so that the first subview is at the top of the `NSSplitView`, and each successive subview is positioned below the previous one. The user can set the height of two subviews by moving a horizontal bar called the *divider*, which makes one subview smaller and the other larger.

To add a view to an `NSSplitView`, you use the `NSView` method `addSubview:`. When the `NSSplitView` is displayed, it checks to see if its subviews are properly tiled. If not, it invokes the delegate method `splitView:resizeSubviewsWithOldSize:`, allowing the delegate to specify the heights of specific subviews. If the delegate doesn't implement this method, the `NSSplitView` sends `adjustSubviews` to itself to yield the default tiling behavior.

When a mouse-down occurs in an `NSSplitView`'s divider, the `NSSplitView` determines the limits of the divider's travel and tracks the mouse to allow the user to drag the divider within these limits. With the following mouse-up, the `NSSplitView` resizes the two affected subviews, informs the delegate that the subviews were resized, and displays the affected views and divider. The `NSSplitView`'s delegate can constrain the travel of specific dividers by implementing the method `splitView:constrainMinCoordinate:maxCoordinate:ofSubviewAt:`.

Method Types

Activity	Class Method
Managing component views	– adjustSubviews – dividerThickness – drawDividerInRect:
Assigning a delegate	– delegate – setDelegate:
Methods Implemented by the Delegate	– splitView:constrainMinCoordinate: maxCoordinate: ofSubviewAt: – splitView:resizeSubviewsWithOldSize: – splitViewDidResizeSubviews: – splitViewWillResizeSubviews:

Instance Methods

adjustSubviews

```
- (void)adjustSubviews
```

Adjusts the heights of the subviews so the total height fills the split view. The subviews are resized proportionally; the size of a subview relative to the other subviews doesn't change. This method is invoked if the split view's delegate doesn't respond to a `splitView:resizeSubviewsWithOldSize:` message.

delegate

```
- (id)delegate
```

Returns the split view's delegate. See also `setDelegate:`.

dividerThickness

```
- (float)dividerThickness
```

Returns the thickness of the divider. See also `adjustSubviews`, `drawDividerInRect:`.

drawDividerInRect:

```
- (void)drawDividerInRect:(NSRect)aRect
```

Draws a divider between two of the split view's subviews. `aRect` describes the entire divider rectangle in the split view's coordinates, which are flipped. The default implementation composites a default image to the center of `aRect`; if you override this method and use a different icon to identify the divider, you may want to change the height of the divider. See also `dividerThickness`.

setDelegate:

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

Sets the NSSplitView's delegate.

Makes `anObject` the split view's delegate. The delegate doesn't need to implement all the delegate methods. See also `delegate`.

Methods Implemented by the Delegate

splitView:constrainMinCoordinate: maxCoordinate: ofSubviewAt:

```
- (void)splitView:(NSSplitView *)splitView
    constrainMinCoordinate:(float *)min
    maxCoordinate:(float *)max ofSubviewAt:(int)offset
```

Sent directly by `splitView` to the delegate. Allows the delegate to constrain further `min` and `max` vertical travel of a divider. `offset` is an index that identifies the dividers in a split view from top to bottom starting with divider 0.

splitView:resizeSubviewsWithOldSize:

```
- (void)splitView:(NSSplitView *)sender
    resizeSubviewsWithOldSize:(NSSize)oldSize
```

Sent directly by `splitView` to the delegate. Allows the delegate to add custom resizing behavior after users resize an splitView. `oldSize` is the size of the split view before the user resized it.

splitViewDidResizeSubviews:

```
- (void)splitViewDidResizeSubviews:(NSNotification *)notification
```

Sent by the default notification center to the delegate; `aNotification` is always `NSSplitViewDidResizeSubviewsNotification`. If the delegate implements this method, it's automatically registered to receive this notification.

splitViewWillResizeSubviews:

```
- (void)splitViewWillResizeSubviews:(NSNotification *)notification
```

Sent by the default notification center to the delegate; `aNotification` is always `NSSplitViewWillResizeSubviewsNotification`. If the delegate implements this method, it's automatically registered to receive this notification.

NSTableColumn

Characteristic	Description
Inherits From:	NSObject
Conforms To:	NSObject(NSObject)
Declared In:	AppKit/NSTableColumn.h

Class Description

An `NSTableColumn` stores the display characteristics and attribute identifier for a column in an `NSTableView`. The `NSTableColumn` determines the following characteristics for its column in the `NSTableView`:

- width and width limits
- resizability
- editability

It also stores two `NSCell` objects:

- The header cell - which is used to draw the column header
- The data cell - used to draw the values for each row

You can control the display of the column by setting the `NSCell` subclasses used, and by setting the font and other display characteristics for these `NSCell`s. For example, you can use the default `NSTextFieldCell` for displaying string values or substitute an `NSImageCell` to display pictures. See the `NSTableView` class specification for a general overview.

Method Types

Activity	Class Method
Creating an NSTableColumn	– initWithIdentifier:
Setting the identifier	– identifier – setIdentifier:
Setting the NSTableView	– tableView – setTableView:
Controlling size	– isResizable – setResizable: – maxWidth – setMaxWidth: – minWidth – setMinWidth: – sizeToFit – width – setWidth:
Controlling editability	– isEditable – setEditable:
Setting component cells	– dataCell – setDataCell: – headerCell – setHeaderCell:

Instance Methods

`dataCell`

```
- (id)dataCell
```

Returns the `NSCell` object used by the `NSTableView` to draw values for the table column.

headerCell

```
- (id)headerCell
```

Returns the NSTableHeaderCell object used to draw the header of the table column. You can set the column title by sending setStringValue: to this object.

initWithIdentifier:

```
- (id)initWithIdentifier:anObject
```

Initializes a newly created table column with anObject as its identifier and with an NSTextFieldCell as its data cell. Send setStringValue: to the header cell to set the column title. This is the designated initializer for the NSTableColumn class. Returns self. See the NSTableView class specification for information on identifiers.

identifier

```
- (id)identifier
```

Returns the object used by the data source to identify the attribute corresponding to the NSTableColumn.

isEditable

```
- (BOOL)isEditable
```

Returns YES if the user can edit cells associated with the table column by double-clicking the column in the NSTableView, and returns NO otherwise. You can initiate editing programmatically regardless of this setting with NSTableView's editColumn:row:withEvent:select: method.

isResizable

```
- (BOOL)isResizable
```

Returns YES if the user is allowed to resize the table column in its NSTableView, and returns NO otherwise. You can change the size programmatically regardless of this setting.

maxWidth

```
- (float)maxWidth
```

Returns the maximum width for the table column. The table column's width can't be made larger than this either by the user or programmatically.

minWidth

```
- (float)minWidth
```

Returns the minimum width for the table column. The table column's width can't be made less than this either by the user or programmatically.

setDataCell:

```
- (void)setDataCell:(NSCell *)aCell
```

Sets to `aCell` the `NSCell` used by the `NSTableView` to draw individual values for the table column. You can use this method to control the font, alignment, and other text attributes for an table column. You can also assign a cell to display things other than text (for example, an NSImageCell to display images).

setEditable:

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

Sets, according to flag, whether the user is allowed to edit cells for the table column by double-clicking the column in the `NSTableView`. You can initiate editing programmatically regardless of this setting with `NSTableView`'s `editColumn:row:withEvent:select:` method.

setHeaderCell:

```
- (void)setHeaderCell:(NSCell *)aCell
```

Sets to `aCell` the `NSCell` used to draw the table column's header. `aCell` should never be `nil`.

setIdentifier:

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

Sets the table column's identifier to `anObject`. This object is used by the data source to identify the attribute corresponding to the table column.

setMaxWidth:

```
- (void)setMaxWidth:(float)maxWidth
```

Sets the table column's maximum width to `maxWidth`, also adjusting the current width if it's greater than this value. The `NSTableView` can be made no wider than this, either by the user or programmatically.

setMinWidth:

```
- (void)setMinWidth:(float)minWidth
```

Sets the table column's minimum width to `minWidth`, also adjusting the current width if it's less than this value. The `NSTableView` can be made no less wide than this, either by the user or programmatically.

setResizable:

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

Sets according to flag whether the user can resize the table column in its `NSTableView`.

setTableView:

```
- (void)setTableView:(NSTableView *)aTableView
```

Sets `aTableView` as the table column's `NSTableView`. You should never need to invoke this method; it's invoked automatically when you add an `NSTableColumn` to an `NSTableView`.

setWidth:

```
- (void)setWidth:(float)newWidth
```

Sets the table column's width to `newWidth`. If `newWidth` exceeds the minimum or maximum width, it's adjusted to the appropriate limiting value. Marks the `NSTableView` as needing display. This method posts `NSTableViewColumnDidResizeNotification` on behalf of the table column's `NSTableView`.

sizeToFit

```
- (void)sizeToFit
```

Resizes the table column to fit the width of its header cell. If the maximum width is less than the width of the header, the maximum is increased to the header's width. Similarly, if the minimum width is greater than the width of the header, the minimum is reduced to the header's width. Marks the `NSTableView` as needing display if the width actually changes.

tableView

```
- (NSTableView *)tableView
```

Returns the `NSTableView` that the table column belongs to.

width

```
- (float)width
```

Returns the table column width.

NSTableHeaderCell

Characteristic	Description
Inherits From:	NSTextFieldCell
Conforms To:	NSCoding, NSCopying (NSCell) NSObject (NSObject)
Declared In:	AppKit / NSTableHeaderCell.h

Class Description

An `NSTableHeaderCell` is used by an `NSTableHeaderView` to draw its column headers. See the `NSTableView` class specification for more information on how it's used. You can subclass `NSTableHeaderCell` and override the `drawWithFrame:inView:` and `highlight:withFrame:inView:` methods to change the way your headers appear.

Method Types

None declared in this class.

NSTableHeaderView

Characteristic	Description
Inherits From:	NSView
Conforms To:	NSCoding(NSResponder), NSObject(NSObject)
Declared In:	AppKit / NSTableHeaderView.h

Class Description

An `NSTableHeaderView` is used by an `NSTableView` to draw headers over its columns and to handle mouse events in those headers. See the `NSTableView` class specification for more information.

Method Types

Activity	Class Method
Setting the table view	– tableView – setTableView:
Checking altered columns	– draggedColumn – draggedDistance – resizedColumn
Utility methods	– columnAtPoint: – headerRectOfColumn:

Instance Methods

columnAtPoint:

```
- (int)columnAtPoint:(NSPoint)aPoint
```

Returns the index of the column whose header lies under `aPoint` in the `NSTableHeaderView`, or -1 is no such column is found. `aPoint` is expressed in the `NSTableHeaderView`'s coordinate system.

draggedColumn

```
- (int)draggedColumn
```

If the user is dragging a column in the `NSTableView`, this method returns the index of that column. Otherwise the return value is meaningless.

draggedDistance

```
- (float)draggedDistance
```

If the user is dragging a column in the `NSTableView`, returns the column's horizontal distance from its original position. Otherwise the return value is meaningless.

headerRectOfColumn:

```
- (NSRect)headerRectOfColumn:(int)columnIndex
```

Returns the rectangle containing the header tile for the column at `columnIndex`. Raises an exception if `columnIndex` is out of bounds.

resizedColumn

```
- (int)resizedColumn
```

If the user is resizing a column in the `NSTableView`, returns the index of that column. Otherwise the return value is meaningless.

setTableView:

```
- (void)setTableView:(NSTableView *)aTableView
```

Sets `aTableView` as the `NSTableColumn`'s `NSTableView`. You should never need to invoke this method; it's invoked automatically when you set the header view for an `NSTableView`.

tableView

```
- (NSTableView *)tableView
```

Returns the `NSTableView` that the `NSTableHeaderView` belongs to.

NSTableView

Characteristic	Description
Inherits From:	NSControl
Conforms To:	NSCoding(NSResponder), NSObject(NSObject)
Declared In:	AppKit / NSTableView.h

Class Description

`NSTableView` displays data for a set of related records, with rows representing individual records and columns representing the attributes of those records. A record is a set of values for a particular object real-world entity, such as an employee or a bank account. For example, in a table of

employee records, each row represents one employee, and the columns represent such attributes as the first and last name, address, salary, and so on. An NSTableView is usually displayed in an NSScrollView.

The user selects rows or columns in the table by clicking, and edits individual cells by double-clicking. The user can also rearrange columns by dragging the column headers and can resize the columns by dragging the divider between two column headers. You can configure the table's parameters so that the user can select more than one rows or columns (or have none selected), so that the user isn't allowed to edit particular columns or rearrange them, and so on. You can also specify an action message to be sent when the user double-clicks something other than an editable cell.

Providing Data for Display

Unlike most NSControls, an NSTableView doesn't store the data it displays. Instead, it gets all of its data from an object that you provide, called its data source. Your data source object can store records in any way, but it must be able to identify them by integer index and must implement methods to provide the following information: how many records the data source contains, and what the value is for a particular record's attribute. If you want to allow the user to edit the records, you must also provide a method for changing the value of an attribute. These methods are described in the NSTableDataSource informal protocol specification.

A record attribute is indicated by an object called its identifier, which is associated with a column in the NSTableView, as described in "Auxiliary Components". The data source uses the identifier as a key to retrieve values for the attribute. The identifier can be any kind of object that uniquely identifies attributes for the data source. For example, if you specify identifiers as NSStrings containing the names of attributes, such as Last Name, Address, and so on, the data source object can use these strings as keys into NSDictionary objects. See the NSTableDataSource informal protocol specification for example of how to use identifiers.

Auxiliary Components

As indicated earlier, NSTableView is usually displayed in an NSScrollView along with its two auxiliary views, the header view and the corner view. The header view is usually an instance of the NSTableHeaderView class, which

draws the column headers and handles column selection, rearranging, and resizing. The corner view is by default a simple view that merely fills in the corner above the vertical scroller. You can replace the default corner view with a custom view; for example, a button that selects all data in the column. See the method description for `setCornerView:` for more information.

Since the `NSTableView` and the `NSTableHeaderView` both need access to information about columns (such as their width), this information is encapsulated in `NSTableColumn` objects. An `NSTableColumn` stores its column's width, and determines whether the user can resize the column or edit its cells. It also holds an `NSCell` object that the `NSTableHeaderView` uses to draw the column header, and an `NSCell` object that the `NSTableView` uses to draw values in the column (it uses the one `NSCell` for every column cell). Finally, the `NSTableColumn` holds the attribute identifier mentioned in Providing Data for Display. Columns are usually identified by index, but can also be identified by their `NSTableColumn` objects.

The cell for the column header is by default an instance of the `NSTableHeaderCell` class; it's used by the `NSTableHeaderView` to draw the column's header. An `NSTableHeaderCell` contains the title displayed over the column, as well as the font and color for that title. You use the API of its superclass, `NSTextFieldCell`, to set a column's title and display attributes for that title.

The cell for the column values is typically an instance of `NSTextFieldCell`, but can be an instance of any `NSCell` subclass, such as `NSImageCell`. This object is used to draw all values in the column and determines the font, alignment, text color, and other such display attributes for those values.

Delegate Messages

`NSTableView` adds a handful of delegate messages to those defined by its superclass, `NSControl`. These methods give the delegate control over the appearance of individual cells in the table, over changes in selection, and over editing of cells. They're invoked during user actions that affect the `NSTableView`, but not when you change things programmatically; when making changes programmaticaly you decide whether you want the delegate to intervene and send the appropriate message (checking that the delegate responds first, of course).

`tableView:willDisplayCell:forTableColumn:row:` informs the delegate that the `NSTableView` is about to draw a particular cell. The delegate can modify the `NSCell` provided to alter the display attributes for that cell: for example, putting uneditable cells in italic or gray text.

`tableView:shouldSelectRow:` and `tableView:shouldSelectTableColumn:` give the delegate control over whether a particular row or column can be selected by the user (though columns can still be reordered). This is useful for disabling particular rows or columns. For example, in a database client application, when another user is editing a record you might want all other users not to be able to select it.

`selectionShouldChangeInTableView:` allows the delegate to deny a change in selection; for example, if the user is editing a cell and enters an improper value, the delegate can prevent the user from selecting or editing any other cells until a proper value has been entered into the original cell.

`tableView:shouldEditTableColumn:row:` asks the delegate whether it's okay to edit a particular cell. The delegate can approve or deny the request.

Method Types

Activity	Class Method
Setting the data source	– dataSource – setDataSource:
Loading data	– reloadData
Setting the delegate	– delegate – setDelegate:
Setting auxiliary views	– cornerView – setCornerView: – headerView – setHeaderView:
Configuring behavior	– allowsColumnReordering – setAllowsColumnReordering: – allowsColumnResizing – setAllowsColumnResizing: – allowsColumnSelection – setAllowsColumnSelection: – allowsEmptySelection – setAllowsEmptySelection: – allowsMultipleSelection – setAllowsMultipleSelection:
Manuipulating columns	– addTableColumn: – autoresizesAllColumnsToFit – columnWithIdentifier: – moveColumn:toColumn: – removeTableColumn: – setAutoresizesAllColumnsToFit: – tableColumns – tableColumnWithIdentifier:
Setting grid attributes	– drawsGrid – setDrawsGrid: – gridColor – setGridColor:

Activity	Class Method
Setting display attributes	– backgroundColor – setBackgroundColor: – intercellSpacing – setIntercellSpacing: – rowHeight – setRowHeight:
Getting table dimensions	– numberOfColumns – numberOfRows
Layout support	– columnsInRect: – rowsInRect: – columnAtPoint: – rowAtPoint: – frameOfCellAtColumn:row: – noteNumberOfRowsChanged – rectOfColumn: – rectOfRow: – sizeLastColumnToFit – tile
Scrolling	– scrollColumnToVisible: – scrollRowToVisible:
Editing cells	– editColumn:row:withEvent:select: – editedColumn – editedRow
Mouse clicking	– clickedColumn – clickedRow – doubleAction – setDoubleAction:

Activity	Class Method
Selecting columns and rows	– deselectAll: – selectAll: – deselectColumn: – deselectRow: – isColumnSelected: – isRowSelected: – numberOfSelectedColumns – numberOfSelectedColumns – selectColumn:byExtendingSelection: – selectRow:byExtendingSelection: – selectedColumn – selectedRow – selectedColumnEnumerator – selectedRowEnumerator
Drawing	– drawGridInClipRect: – drawRow:clipRect: – highlightSelectionInClipRect:
Delegate methods	– selectionShouldChangeInTableView: – tableView:shouldEditTableColumn:row: – tableView:shouldSelectRow: – tableView:shouldSelectTableColumn: – tableView:willDisplayCell:forTableColumn:row:

Instance Methods

addTableColumn:

```
- (void)addTableColumn:(NSTableColumn *)column
```

Appends `aColumn` to the table view. See also `removeTableColumn:`.

allowsColumnReordering

```
- (BOOL)allowsColumnReordering
```

Returns `YES` if the table view allows the user to rearrange columns by dragging their headers, and returns `NO` otherwise. The default is `YES`. You can rearrange columns programmatically regardless of this setting. See also `setAllowsColumnReordering:`.

allowsColumnResizing

```
- (BOOL)allowsColumnResizing
```

Returns YES if the table view allows the user to resize columns by dragging between their headers, and returns NO otherwise. The default is YES. You can resize columns programmatically regardless of this setting. See also `setAllowsColumnResizing:`.

allowsColumnSelection

```
- (BOOL)allowsColumnSelection;
```

Returns YES if the NSTableView allows the user to select columns by clicking their headers, NO otherwise. The default is YES. You can select columns programmatically regardless of this setting. See also `setAllowsColumnSelection:`.

allowsEmptySelection

```
- (BOOL)allowsEmptySelection;
```

Returns YES if the table view allows the user to select zero columns or rows, NO otherwise. The default is YES. You can not set an empty selection programmatically if this setting is NO, unlike with the other settings that affect selection behavior. See also `setAllowsEmptySelection:`.

allowsMultipleSelection

```
- (BOOL)allowsMultipleSelection
```

Controls whether the the user can select more than one row or column at a time. If `flag` is YES the user can select multiple rows or columns; if `flag` is NO the user can't. The default is NO. You can select multiple columns or rows programmatically regardless of this setting. See also `setAllowsMultipleSelection:`.

autoresizesAllColumnsToFit

```
- (BOOL)autoresizesAllColumnsToFit
```

Returns YES if columns are resized to fit, and returns NO otherwise. See also setAutoresizesAllColumnsToFit:.

backgroundColor

```
- (NSColor *)backgroundColor
```

Returns the color used to draw the table view background. The default background color is light gray. See also setBackgroundColor:.

clickedColumn

```
- (int)clickedColumn
```

Returns the index of the column clicked by the user. See also clickedRow.

clickedRow

```
- (int)clickedRow
```

Returns the index of the row clicked by the user. See also clickedColumn.

columnAtPoint:

```
- (int)columnAtPoint:(NSPoint)aPoint
```

Returns the index of the column that aPoint lies in, or -1 if aPoint lies outside the table view's bounds. See also rowAtPoint:.

columnsInRect:

```
- (NSRange)columnsInRect:(NSRect)aRect;
```

Returns a range of indices for the columns that lie wholly or partially within the horizontal boundaries of aRect. The location of the range is the first such column's index, and the length is the number of columns that lie in aRect. Both the width and height of aRect must be nonzero values, or columnsInRect: returns an NSRange whose location and length are zero. See also rowsInRect:.

columnWithIdentifier:

```
- (int)columnWithIdentifier:(id)identifier;
```

Returns the index of the first column whose identifier is equal to `anObject`, when compared using `isEqual:`, or -1 if no columns are found with the specified identifier. See also `tableColumnWithIdentifier:`.

cornerView

```
- (NSView *)cornerView
```

Returns the `NSView` used to draw the area to the left of the column headers and above the vertical scroller of the enclosing `NSScrollView`. This is by default a simple view that merely fills in its frame, but you can replace it with a custom view using `setCornerView:`. See also `setCornerView:`.

dataSource

```
- (id)dataSource
```

Returns the object that provides the data displayed by the table view. See the class description and the `NSTableDataSource` informal protocol specification for more information. See also `setDataSource:`.

delegate

```
- (id)delegate;
```

Returns the table view's delegate. See also `setDelegate:`.

deselectAll:

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

Deselects all selected rows or columns if empty selection is allowed, otherwise does nothing. Posts `NSTableViewSelectionDidChangeNotification` to the default notification center if the selection does in fact change. As a target-action method, `deselectAll:` checks with the delegate before changing the selection, using `selectionShouldChangeInTableView:`.

deselectColumn:

```
- (void)deselectColumn:(int)columnIndex
```

Deselects the column at `columnIndex` if it's selected, regardless of whether empty selection is allowed. If the selection does in fact change, posts `NSTableViewSelectionDidChangeNotification` to the default notification center. If the indicated column was the last column selected by the user, the column nearest it effectively becomes the last selected column. In case of a tie, priority is given to the column on the left. This method doesn't check with the delegate before changing the selection. See also `deselectRow:`.

deselectRow:

```
- (void)deselectRow:(int)rowIndex
```

Deselects the row at `rowIndex` if it's selected, regardless of whether empty selection is allowed. If the selection does in fact change, posts `NSTableViewSelectionDidChangeNotification` to the default notification center. If the indicated row was the last row selected by the user, the row nearest it effectively becomes the last selected row. In case of a tie, priority is given to the row above. This method doesn't check with the delegate before changing the selection. See also `deselectColumn:`.

doubleAction

```
- (SEL)doubleAction
```

Returns the message sent to the target when the user double-clicks a column header or an uneditable cell. See also `setDoubleAction:`.

drawGridInClipRect:

```
- (void)drawGridInClipRect:(NSRect)clipRect
```

Draws the grid lines within `clipRect`, using the grid color set with `setGridColor:`. This method draws a grid regardless of whether the table view is set to draw one automatically. Override this method to draw grid lines other than the default gray lines.

drawRow:clipRect:

```
- (void)drawRow:(int)rowIndex clipRect:(NSRect)clipRect
```

Draws the cells for the row at `rowIndex` in the columns that intersect `clipRect`. Sends `tableView:willDisplayCell:forTableColum:row:` to the delegate before drawing each cell. Override this method to customize the appearance of your subclass.

drawsGrid

```
- (BOOL)drawsGrid
```

Returns `YES` if the table view draws grid lines around cells, and return `NO` if it does not. The default is `YES`. See also `setDrawsGrid:`.

editColumn:row:withEvent:select:

```
- (void)editColumn:(int)columnIndex
    row:(int)rowIndex
    withEvent:(NSEvent *)theEvent
    select:(BOOL)select
```

Edits the cell at `columnIndex` and `rowIndex`. Scrolls the table view so that the cell is visible, sets up the field editor, and sends `selectWithFrame:inView:editor:delegate:start:length:` and `editWithFrame:inView:editor:delegate:event:` to the field editor's `NSCell` object with the table view as the text delegate. This method is invoked automatically in response to user actions. You should rarely need to invoke it directly.

editedColumn

```
- (int)editedColumn
```

If sent during `editColumn:row:withEvent:select:` returns the index of the column being edited; otherwise returns -1. See also `editedRow`.

editedRow

```
- (int)editedRow
```

If sent during `editColumn:row:withEvent:select:` returns the index of the row being edited; otherwise returns -1. See also `editedColumn`.

frameOfCellAtColumn:row:

```
- (NSRect)frameOfCellAtColumn:(int)columnIndex row:(int)rowIndex
```

Returns a rectangle locating the cell that lies at the intersection of `columnIndex` and `rowIndex`. Returns `NSZeroRect` if `columnIndex` or `rowIndex` are greater than the number of columns or rows in the table view. The result of this method is used in a `drawWithFrame:inView:` message to the `NSTableColumn`'s data cell.

gridColor

```
- (NSColor *)gridColor
```

Returns the color used to draw grid lines. The default color is gray. See also `setGridColor:`.

headerView

```
- (NSTableHeaderView *)headerView
```

Returns the `NSView` used to draw headers over columns, or nil if the `NSTableView` has no header view. See the class description and the `NSTableHeaderView` class specification for more information.. See also `setHeaderView:`.

highlightSelectionInClipRect:

```
- (void)highlightSelectionInClipRect:(NSRect)clipRect
```

Highlights the region of the table view in `clipRect`. This method is invoked after `drawRow:clipRect:`. Override this method to change the manner in which your subclass highlights selections.

intercellSpacing

```
- (NSSize)intercellSpacing
```

Returns the horizontal and vertical spacing between cells. The default spacing is (3.0, 2.0). See also `setIntercellSpacing:`.

isColumnSelected:

```
- (BOOL)isColumnSelected:(int)columnIndex
```

Returns `YES` if the column at `columnIndex` is selected, and returns `NO` otherwise.`isRowSelected:`

isRowSelected:

```
- (BOOL)isRowSelected:(int)rowIndex
```

Returns `YES` if the row at `rowIndex` is selected, and returns `NO` otherwise. See also `isColumnSelected:`.

moveColumn:toColumn:

```
- (void)moveColumn:(int)columnIndex toColumn:(int)newIndex
```

Moves the column and heading at `columnIndex` to `newIndex`, inserting the column before the existing column at `newIndex`. This method posts `NSTableViewColumnDidMoveNotification` to the default notification center, along with an `NSDictionary` that contains integer `NSNumber`s for both the index of the column moved (dictionary key `NSOldColumn`) and the index to which it was moved (dictionary key `NSNewColumn`).

noteNumberOfRowsChanged

```
- (void)noteNumberOfRowsChanged
```

Informs the table view that the number of records in its data source has changed. This method allows the table view to update the scrollers in its `NSScrollView` without actually reloading data. It's useful for a data source that continually receives data in the background over a period of time, in which case the table view can remain responsive to the user while the data is loaded. See the `NSTableDataSource` informal protocol specification for information on the messages an table view sends to its data source.

numberOfColumns

```
- (int)numberOfColumns
```

Returns the number of table view columns. See also `numberOfRows`.

numberOfRows

```
- (int)numberOfRows
```

Returns the number of table view rows. See also `numberOfColumns`.

numberOfSelectedColumns

```
- (int)numberOfSelectedColumns
```

Returns the number of selected columns.

numberOfSelectedRows

```
- (int)numberOfSelectedRows
```

Returns the number of selected rows. See also `numberOfSelectedColumns`.

rectOfColumn:

```
- (NSRect)rectOfColumn:(int)columnIndex
```

Returns the rectangle containing the column at `columnIndex`. Raises an exception if columnIndex lies outside the range of valid column indices for the table view. See also `rectOfRow:`.

rectOfRow:

```
- (NSRect)rectOfRow:(int)rowIndex
```

Returns the rectangle containing the row at `rowIndex`. Raises an exception if columnIndex lies outside the range of valid column indices for the table view. See also `rectOfColumn:`.

reloadData

```
- (void)reloadData
```

Reloads all values from the data source and redisplays the table view.

removeTableColumn:

```
- (void)removeTableColumn:(NSTableColumn *)column
```

Deletes `aTableColumn` from the table view. See also `addTableColumn:`.

rowAtPoint:

```
- (int)rowAtPoint:(NSPoint)point;
```

Returns the index of the row that `aPoint` lies in, or -1 if `aPoint` lies outside the t table view's bounds. See also `columnAtPoint:`.

rowHeight

```
- (float)rowHeight;
```

Returns the height of each row in the table view. The default row height is 16.0. See also `setRowHeight:`.

rowsInRect:

```
- (NSRange)rowsInRect:(NSRect)aRect;
```

Returns a range of indices for the rows that lie wholly or partially within the vertical boundaries of `aRect`. The location of the range is the first such row's index, and the length is the number of rows that lie in `aRect`. Both the width and height of `aRect` must be nonzero values, or columnsInRect: returns an `NSRange` whose location and length are zero. See also `columnsInRect:`.

scrollColumnToVisible:

```
- (void)scrollColumnToVisible:(int)columnIndex
```

Scrolls the table view and header view horizontally in an enclosing `NSClipView` so that the column specified by `columnIndex` is visible. See also `scrollRowToVisible:`.

scrollRowToVisible:

```
- (void)scrollRowToVisible:(int)rowIndex
```

Scrolls the table view vertically in an enclosing NSClipView so that the row specified by rowIndex is visible. See also scrollColumnToVisible:.

selectAll:

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

If the table allows multiple selection, selects all rows or all columns, according to whether rows or columns were most recently selected; otherwise does nothing. Posts NSTableViewSelectionDidChangeNotification to the default notification center if the selection does in fact change. As a target-action method, deselectAll: checks with the delegate before changing the selection. See also deselectAll:.

selectColumn:byExtendingSelection:

```
- (void)selectColumn:(int)columnIndex
    byExtendingSelection:(BOOL)extend
```

Selects the column at columnIndex, regardless of whether column selection is allowed. If extend is NO, deselects all before selecting the new column. Raises an exception if multiple selection is not allowed and extend is YES. Posts NSTableViewSelectionDidChangeNotification to the default notification center if the selection does in fact change. This method doesn't check with the delegate before changing the selection. If the user is editing a cell, editing is simply forced to end and the selection is changed. See also selectRow:byExtendingSelection:.

selectedColumn

```
- (int)selectedColumn
```

Returns the index of the last column selected or added to the selection, or -1 if no column is selected. See also selectedRow.

selectedColumnEnumerator

```
- (NSEnumerator *)selectedColumnEnumerator
```

Returns an object that enumerates the indices of the selected columns as NSNumbers. See also selectedRowEnumerator.

selectedRow

```
- (int)selectedRow
```

Returns the index of the last row selected or added to the selection, or -1 if no row is selected. See also selectedColumn.

selectedRowEnumerator

```
- (NSEnumerator *)selectedRowEnumerator
```

Returns an object that enumerates the indices of the selected rows as NSNumbers. See also selectedColumnEnumerator.

selectRow:byExtendingSelection:

```
- (void)selectRow:(int)rowIndex byExtendingSelection:(BOOL)extend;
```

Selects the row at rowIndex. If extend is NO, deselects all before selecting the new row. Raises an exception if multiple selection isn't allowed and extend is YES. Posts NSTableViewSelectionDidChangeNotification to the default notification center if the selection does in fact change. This method doesn't check with the delegate before changing the selection. If the user is editing a cell, editing is simply forced to end and the selection is changed. See also selectColumn:byExtendingSelection:.

setAllowsColumnReordering:

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

Controls whether the user can drag column headers to reorder columns. If flag is YES the user can reorder columns; if flag is NO the user cannot. The default is YES. You can rearrange columns programmatically regardless of this setting. See also allowsColumnReordering.

setAllowsColumnResizing:

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

Controls whether the user can resize columns by dragging between headers. If `flag` is YES the user can resize columns; if `flag` is NO the user can't. The default is YES. You can resize columns programmatically regardless of this setting. See also `allowsColumnResizing`.

setAllowsColumnSelection:

```
- (void)setAllowsColumnSelection:(BOOL)flag;
```

Controls whether the user can select an entire column by clicking its header. If `flag` is YES the user can select columns; if `flag` is NO the user can't. The default is YES. You can select columns programmatically regardless of this setting. See also `allowsColumnSelection`.

setAllowsEmptySelection:

```
- (void)setAllowsEmptySelection:(BOOL)flag;
```

Controls whether the table view allows zero rows or columns to be selected. If `flag` is YES empty selection is allowed; if `flag` is NO it isn't. The default is YES. See also `allowsEmptySelection`.

setAllowsMultipleSelection:

```
- (void)setAllowsMultipleSelection:(BOOL)flag;
```

Returns YES if the table view allows the user to select more than one column or row at a time, and returns NO otherwise. The default is NO. You can select multiple columns or rows programmatically regardless of this setting. See also `allowsMultipleSelection`.

setAutoresizesAllColumnsToFit:

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

Controls whether columns are resized to fit. If `flag` is YES, all columns are resized to fit; if `flag` is NO, columns are not resized to fit. See also `autoresizesAllColumnsToFit`.

setBackgroundColor:

```
- (void)setBackgroundColor:(NSColor *)color
```

Sets the table view's background color to aColor. Doesn't redisplay the table view or mark it as needing display. See also backgroundColor.

setCornerView:

```
- (void)setCornerView:(NSView *)cornerView;
```

Sets the table view's corner view to aView. The default corner view merely draws a bezeled rectangle, but you can replace it with a custom view that displays an image or with a control that can handle mouse events, such as a select-all button. Your custom corner view should be as wide as a vertical NSScroller and as tall as the NSTableView's header view. See also cornerView.

setDataSource:

```
- (void)setDataSource:(id)aSource
```

Sets the table view's data source to anObject and invokes tile. anObject should implement the appropriate methods of the NSTableDataSource informal protocol. This method raises an exception if anObject doesn't respond to either numberOfRowsInTableView: or tableView:objectValueForTableColumn:row:. See also dataSource.

setDelegate:

```
- (void)setDelegate:(id)delegate;
```

Sets the table view's delegate to anObject. See also delegate.

setDoubleAction:

```
- (void)setDoubleAction:(SEL)aSelector
```

Sets the message sent to the target to aSelector when the user double-clicks an uneditable cell or a column header. If the double-clicked cell is editable, this message isn't sent and the cell is edited instead. You can use this method to implement features such as sorting records according to the column that was double-clicked. See also doubleAction.

setDrawsGrid:

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

Controls whether the table view draws grid lines around cells. If `flag` is `YES` it draws grid; if `flag` is `NO` it does not draw the grid. The default is `YES`. See also `drawsGrid`.

setGridColor:

```
- (void)setGridColor:(NSColor *)color
```

Sets the color used to draw grid lines to `aColor`. The default color is gray. See also `gridColor`.

setHeaderView:

```
- (void)setHeaderView:(NSTableHeaderView *)headerView;
```

Sets the table view's header view to to `aHeaderView`. See also `headerView`.

setIntercellSpacing:

```
- (void)setIntercellSpacing:(NSSize)aSize
```

Sets the width and height between cells to those in `aSize` and redisplays the table view. The default intercell spacing is (3.0, 2.0). See also `intercellSpacing`.

setRowHeight:

```
- (void)setRowHeight:(float)rowHeight;
```

Sets the height for rows to `rowHeight` and invokes `tile`. See also `rowHeight`.

sizeLastColumnToFit

```
- (void)sizeLastColumnToFit;
```

Resizes the last column if there's room so that the table view fits exactly within its enclosing `NSClipView`.

tableColumns

```
- (NSArray *)tableColumns
```

Returns the NSTableColumns in the table view.

tableColumnWithIdentifier:

```
- (NSTableColumn *)tableColumnWithIdentifier:(id)identifier
```

Returns the NSTableColumn object for the first column whose identifier is equal to anObject, as compared using isEqual:, or nil if no columns are found with the specified identifier. See also columnWithIdentifier:.

tile

```
- (void)tile;
```

Properly sizes the table view and its header view, and marks the table view as needing display. Also resets cursor rectangles for the header view and line scroll amounts for the NSScrollView.

Methods Implemented by the Delegate

selectionShouldChangeInTableView:

```
- (BOOL)selectionShouldChangeInTableView:(NSTableView *)aTableView
```

Returns YES to permit a table view to change its selection (typically a row being edited), NO to deny permission. The user can select and edit different cells within the same row, but can't select another row unless the delegate approves. The delegate can implement this method for complex validation of edited rows based on the values of any of their cells.

tableView:shouldEditTableColumn:row:

```
- (BOOL)tableView:(NSTableView *)tableView
    shouldEditTableColumn:(NSTableColumn *)tableColumn
    row:(int)row
```

Returns YES to permit aTableView to edit the cell at rowIndex in aTableColumn, NO to deny permission. The delegate can implement this method to disallow editing of specific cells.

tableView:shouldSelectRow:

```
- (BOOL)tableView:(NSTableView *)aTableView
    shouldSelectRow:(int)rowIndex
```

Returns YES to permit aTableView to select the row at rowIndex, NO to deny permission. The delegate can implement this method to disallow selection of particular rows.

tableView:shouldSelectTableColumn:

```
- (BOOL)tableView:(NSTableView *)aTableView
    shouldSelectTableColumn:(NSTableColumn *)aTableColumn
```

Returns YES to permit aTableView to select aTableColumn, NO to deny permission. The delegate can implement this method to disallow selection of particular columns.

tableView:willDisplayCell:forTableColumn:row:

```
- (void)tableView:(NSTableView *)aTableView
    willDisplayCell:(id)aCell
    forTableColumn:(NSTableColumn *)aTableColumn
    row:(int)row
```

Informs the delegate that aTableView will display the cell at rowIndex in aTableColumn using aCell. The delegate can modify the display attributes of aCell to alter the appearance of the cell. Since aCell is reused for every row in aTableColumn, the delegate must set the display attributes both when drawing special cells and when drawing normal cells.

NSText

Inherits From:	NSView : NSResponder : NSObject
Conforms To:	NSChangeSpelling NSIgnoreMisspelledWords NSCoding (NSResponder) NSObject (NSObject)
Declared In:	AppKit/NSTextView.h

Class Description

`NSText` is an abstract superclass that declares the programmatic interface to objects that manage text (see `NSCStringText` for a concrete subclass). `NSText` objects are used by the Application Kit wherever text appears in interface objects: An `NSText` object draws the title of a window, the commands in a menu, the title of a button, and the items in a browser. Your application inherits these uses of the `NSText` class when it incorporates any of these objects into its interface. Your application can also create `NSText` objects for its own purposes.

The `NSText` class is unlike most other classes in the Application Kit in its complexity and range of features. One of its design goals is to provide a comprehensive set of text-handling features so that you'll rarely need to create a subclass. An `NSText` object can (among other things):

- Control the color of its text and background.
- Control the font and layout characteristics of its text.
- Control whether text is editable.
- Wrap text on a word or character basis.
- Display graphic images within its text.
- Write text to or read text from files in the form of RTFD—Rich Text Format files that contain TIFF or EPS images.
- Let another object, the delegate, dynamically control its properties.
- Let the user copy and paste text within and between applications.
- Let the user copy and paste font and format information between `NSText` objects.

- Let the user check the spelling of words in its text.
- Let the user control the format of paragraphs by manipulating a ruler.

Graphical user-interface building tools (such as Interface Builder) may give you access to `NSText` objects in several different configurations, such as those found in the `NSTextField`, `NSForm`, and `NSScrollView` objects. These classes configure an `NSText` object for their own specific purposes. Additionally, all `NSTextField`s, `NSForm`s, `NSButton`s within the same window—in short, all objects that access an `NSText` object through associated `NSCell`s—share the same `NSText` object, reducing the memory demands of an application. Thus, it's generally best to use one of these classes whenever it meets your needs, rather than create `NSText` objects yourself. If one of these classes doesn't provide enough flexibility for your purposes, you can create `NSText` objects programatically.

Plain and Rich NSText Objects

When you create an `NSText` object directly, by default it allows only one font, line height, text color, and paragraph format for the entire text. Once an `NSText` object is created, you can alter its global settings using methods such as `setFont:` and `setTextColor:`. For convenience, such an `NSText` object will be called a *plain* `NSText` object.

To allow multiple values for attributes such as font and color, you must send the `NSText` object a `setRichText:YES` message. An `NSText` object that allows multiple fonts also allows multiple paragraph formats, line heights, and so on. For convenience, such an `NSText` object will be called a *rich* `NSText` object.

A rich NSText object can use RTF (Rich Text Format) as an interchange format. Not all RTF control words are supported: On input, an NSText object ignores any control word it doesn't recognize; some of those it can read and interpret it doesn't write out. These are the RTF control words that an NSText object recognizes.

Table 1-24 RTF Control Words Recognized by the NSText Object

Control Word	Read	Write
\ansi	yes	yes
\b	yes	yes
\cb	yes	yes
\cf	yes	yes
\colortbl	yes	yes
\dnn	yes	yes
\fin	yes	yes
\fn	yes	yes
\fonttbl	yes	yes
\fsn	yes	yes
\i	yes	yes
\lin	yes	yes
\margrn	yes	yes
\paperwn	yes	yes
\mac	yes	no
\margln	yes	yes
\par	yes	yes
\pard	yes	no
\pca	yes	no
\qc	yes	yes
\ql	yes	yes
\qr	yes	yes

Table 1-24 RTF Control Words Recognized by the NSText Object

Control Word	Read	Write
\sn	yes	no
\tab	yes	yes
\upn	yes	yes

NSText objects are designed to work closely with various other objects. Some of these—such as the delegate or an embedded graphic object—require a degree of programming on your part. Others—such as the Font panel, spelling checker, or ruler—take no effort other than deciding whether the service should be enabled or disabled. The following sections discuss these interrelationships.

Notifying the NSText Object's Delegate

Many of an NSText object's actions can be controlled through an associated object, the NSText object's delegate. If it implements any of the following methods, the delegate receives the corresponding message at the appropriate time:

- `textDidBeginEditing:`
- `textDidChange:`
- `textDidEndEditing:`
- `textShouldBeginEditing:`
- `textShouldEndEditing:`

For example, if the delegate implements the `textDidBeginEditing:` method, it will receive notification when the user first attempts to change the text. Depending on the method's return value, the delegate can either allow or prohibit changes to the text. See "Methods Implemented by the Delegate". The delegate can be any object you choose, and one delegate can control multiple NSText objects.

Adding Graphics to the Text

A rich NSText object allows graphics to be embedded in the text. Each graphic is treated as a single (possibly large) "character": The text's line height and character placement are adjusted to accommodate the graphic "character."

Graphics are embedded in the text in either of two ways: programmatically or directly through user actions. In the programmatic approach, graphic objects are added using the `replaceRange:WithRTFD:` method.

An alternate means of adding an image to the text is for the user to drag an EPS or TIFF file icon directly into an `NSText` object. The `NSText` object automatically creates a graphic object to manage the display of the image. This feature requires a rich `NSText` object that has been configured to receive dragged images—see the `setImportsGraphics:` method.

Images that have been imported in this way can be written as RTFD documents. Programmatic creation of RTFD documents is not supported in this version of OpenStep. RTFD documents use a file package, or directory, to store the components of the document (the "D" stands for "directory"). The file package has the name of the document plus a `.rtfd` extension. The file package always contains a file called `TXT.rtf` for the text of the document, and one or more TIFF or EPS files for the images. An `NSText` object can transfer information in an RTFD document to a file and read it from a file—see the `writeRTFDToFile:atomically:` and `readRTFDFromFile:` methods.

Cooperating with Other Objects and Services

`NSText` objects are designed to work with the Application Kit's font conversion system. By default, an `NSText` object keeps the Font panel updated with the font of the current selection. It also changes the font of the selection (for a rich `NSText` object) or of the entire text (for a default `NSText` object) to reflect the user's choices in the Font panel or menu. To disconnect an `NSText` object from this service, send it a `setUsesFontPanel:NO` message.

If an `NSText` object is a subview of an `NSScrollView`, it can cooperate with the `NSScrollView` to display and update a ruler that displays formatting information. The `NSScrollView` retiles its subviews to make room for the ruler, and the `NSText` object updates the ruler with the format information of the paragraph containing the selection. The `toggleRuler:` method controls the display of this ruler. Users can modify paragraph formats by manipulating the components of the ruler.

Coordinates and sizes mentioned in the method descriptions that follow are in PostScript units—1/72 of an inch.

Method Types

Activity	Class Method
Getting and setting contents	– replaceCharactersInRange:withRTF: – replaceCharactersInRange:withRTFD: – replaceCharactersInRange:withString: – replaceRange:withRTF: – replaceRange:withRTFD: – RTFDFromRange: – RTFFromRange: – setString: – setText: – setText:range: – string
Managing global chracteristics	– alignment – drawsBackground – importsGraphics – isEditable – isRichText – isSelectable – setAlignment: – setDrawsBackground: – setEditable: – setImportsGraphics: – setRichText: – setSelectable:
Managing font and color	– backgroundColor – changeFont: – font – setBackgroundColor: – setTextColor:range: – setFont: – setFont:range: – setTextColor: – setTextColor:range: – setUsesFontPanel: – textColor – usesFontPanel
Managing the selection	– selectedRange – setSelectedRange:

Activity	Class Method
Sizing the frame rectangle	– isHorizontallyResizable – isVerticallyResizable – maxSize – minSize – setHorizontallyResizable: – setMaxSize: – setMinSize: – setVerticallyResizable: – sizeToFit
Responding to Editing Commands	– alignCenter: – alignLeft: – alignRight: – copy: – copyFont: – copyRuler: – cut: – delete: – paste: – pasteFont: – pasteRuler: – selectAll: – subscript: – superscript: – underline: – unscript:
Managing the ruler	– isRulerVisible – toggleRuler:
Spelling	– checkSpelling: – showGuessPanel:
Scrolling	– scrollRangeToVisible:
Reading and Writing RTFD files	– readRTFDFromFile: – writeRTFDToFile:atomically:

Activity	Class Method
Managing the field editor	– isFieldEditor – setFieldEditor:
Managing the delegate	– delegate – setDelegate:
Methods Implemented by the Delegate	– textDidBeginEditing: – textDidChange: – textDidEndEditing: – textShouldBeginEditing: – textShouldEndEditing:

Instance Methods

alignCenter:

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

Centers the selected text between the margins. The sending object passes its `id` as part of the `alignCenter:` message. The text is rewrapped and redrawn. See also `alignLeft:`, `alignRight:`, `alignment`.

alignLeft:

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

Aligns selected text to the left margin. The sending object passes its `id` as part of the `alignLeft:` message. The text is rewrapped and redrawn.

alignRight:

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

Aligns selected text to the right margin. The sending object passes its `id` as part of the `alignRight:` message. The text is rewrapped and redrawn. See also `alignLeft:`, `alignCenter:`, `alignment`.

alignment

```
- (NSTextAlignment)alignment
```

Returns how text in the NSText object is aligned between the margins. The return value can be one of the following constants:

- NSLeftTextAlignment
- NSRightTextAlignment
- NSCenterTextAlignment
- NSJustifiedTextAlignment
- NSNaturalTextAlignment

See also setAlignment:.

backgroundColor

```
- (NSColor *)backgroundColor
```

Returns the background color for the NSText object. See also setBackgroundColor:, textColor.

changeFont:

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

Changes the font of the selection for a rich NSText object. It changes the font for the entire NSText object for a plain NSText object. sender must respond to the convertFont: message. If the NSText object's delegate implements the method, it receives a textWillConvert:fromFont:toFont: message for each text run that's about to be converted.

checkSpelling:

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

Searches for a misspelled word in the text of the receiving NSText object. The search starts at the current selection and continues until it reaches a word suspected of being misspelled or the end of the text. If a word isn't recognized by the spelling server or listed in the user's local dictionary, it's highlighted. A showGuessPanel: message will then display the Guess panel and allow the user to make a correction or add the word to the local dictionary.

copy:

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

Copies the selected text from the NSText object to the pasteboard. The selection remains unchanged. The pasteboard receives the text and its corresponding run information. The pasteboard types used are NSStringPboardType and NSRTFPboardType. See also copyFont:, copyRuler:, cut:, delete:, paste:.

copyFont:

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

Copies font information for the selected text to the font pasteboard. If the selection spans more than one font, the information copied is that of the first font in the selection. The selection remains unchanged. The pasteboard type used is NSFontPboardType. The sender passes its id as the argument of the copyFont: message. See also pasteFont:, copy:, copyRuler:.

copyRuler:

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

Copies the selected text's ruler to the pasteboard. The selection expands to paragraph boundaries. The ruler controls a paragraph's text alignment, tab settings, and indentation. If the selection spans more than one paragraph, the information copied is that of the first paragraph in the selection. The pasteboard type used is NSRulerPboardType. Once copied to the pasteboard, ruler information can be pasted into another object or application that's able to paste RTF data into its document. The sender passes its id as the message argument. See also pasteRuler:, copy:, copyFont:.

cut:

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

Copies the selected text to the pasteboard and then deletes it from the NSText object. The pasteboard receives the text and its corresponding font information. If the NSText object's delegate implements the method, it receives a textDidChange: message immediately after the cut operation. If this is the first change since the NSText object became the first responder (and the delegate implements the method), a textDidEndEditing: message is also sent to the delegate. The sender passes its id as part of the cut: message. See also paste:, copy:, delete:.

delegate

```
- (id)delegate
```

Returns the delegate of the NSText object. See also setDelegate:.

delete:

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

Deletes the selected text without adding it to the pasteboard. This method posts the notification NSTextDidChangeNotification with the receiving object to the default notification center and may post the NSTextDidBeginEditing notification as well. (NSTextDidEndEditingNotification gets posted when the first responder changes.) If the NSText object's delegate implements the method, it receives a textDidChange: message immediately after the cut operation. If this is the first change since the NSText object became the first responder and the delegate implements the method, a textDidEndEditing: message is also sent to the delegate. The sender passes its id as part of the cut: message. See also paste:, copy:, cut:.

drawsBackground

```
- (BOOL)drawsBackground
```

Returns YES if the NSText object draws its own background, and returns NO otherwise. See also setDrawsBackground:.

font

```
- (NSFont *)font
```

Returns the default NSFont object for the NSText object. See also setFont:, changeFont:.

importsGraphics

```
- (BOOL)importsGraphics
```

Returns YES if the NSText object can import TIFF and EPS images dragged into it by the user, and returns NO otherwise. The default is NO. See also setImportsGraphics:, isRichText.

isEditable

- (BOOL)isEditable

Returns YES if users can edit the NSText object, and returns NO otherwise. The default is YES. See also setEditable:, isSelectable.

isFieldEditor

- (BOOL)isFieldEditor

Returns YES if the receiving NSText object gives up first responder status on tab, carriage return, and so on, and returns NO otherwise. The default is YES. See also setFieldEditor:.

isHorizontallyResizable

- (BOOL)isHorizontallyResizable

Returns YES if the frame width can automatically change size, and returns NO otherwise. The default is NO. See also setHorizontallyResizable:, isVerticallyResizable.

isRichText

- (BOOL)isRichText

Returns YES if the text in the NSText object is RTF, and returns NO otherwise. See also setRichText:.

isRulerVisible

- (BOOL)isRulerVisible

Returns YES if the ruler is visible in the NSText object's superview; otherwise, returns NO. See also toggleRuler:.

isSelectable

```
- (BOOL)isSelectable
```

Returns YES if the text can be selected, NO if not. The default value is YES. See also `setSelectable:`.

isVerticallyResizable

```
- (BOOL)isVerticallyResizable
```

Returns YES if the frame height can automatically change size vertically, NO if not. The default value is NO. See also `setVerticallyResizable:`, `isHorizontallyResizable`.

maxSize

```
- (NSSize)maxSize
```

Gets the maximum size of the NSText object's frame. See also `setMaxSize:`, `minSize`.

minSize

```
- (NSSize)minSize
```

Gets the minimum size of the NSText object's frame. See also `setMinSize:`, `maxSize`.

paste:

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

Replaces the selected text with the contents of the pasteboard. This method posts the notification `NSTextDidChangeNotification` with the receiving object to the default notification center and may post the `NSTextDidBeginEditing` notification as well.

pasteFont:

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

Places the contents of the selection pasteboard into the NSText object at the position of the current selection. If the selection is zero-width, the text is inserted at the caret. If the selection has positive width, the selection is replaced by the contents of the pasteboard. In either case, the text is rewrapped and redrawn. sender is the id of the sending object.

Before the paste operation, a textDidBeginEditing: message is sent to the delegate, assuming that this is the first change since the NSText object became the first responder and that the delegate implements the method. After the paste operation, the delegate receives a textDidChange: message, if it implements the method.

This method posts the NSTextDidChangeNotification notification with the receiving object to the default notification center and may post the NSTextDidBeginEditing notification as well.

pasteRuler:

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

Takes ruler information from the ruler pasteboard and applies it to the paragraph or paragraphs marked by the current selection. The ruler controls a paragraph's text alignment, tab settings, and indentation. sender is the id of the sending object. After the ruler is pasted, the text is rewrapped and redrawn. If the ruler is visible, it's also updated.

pasteRuler: works only with rich NSText objects. Attempting to paste a ruler into a plain NSText object generates a system beep without altering any ruler settings. Before the paste operation, a textDidBeginEditing: message is sent to the delegate, assuming that this is the first change since the Text object became the first responder and that the delegate implements the method. After the paste operation, the delegate receives a textDidChange: message, if it implements the method. See also copyRuler:, pasteFont:, paste:, copy:.

readRTFDFromFile:

```
- (BOOL)readRTFDFromFile:(NSString *)path
```

Reads RTFD or RTF data from the file package specified by path and initializes an NSText object with it; returns whether the operation succeeded. See also writeRTFDToFile:atomically:, RTFFromRange:.

replaceCharactersInRange:withRTF:

```
- (void)replaceCharactersInRange:(NSRange)range
    withRTF:(NSData *)rtfData
```

Replaces the characters within the specified `range` of text with the RTF data `rtfData`. This message is sent in response to pasting RTF data from the pasteboard. See also `replaceCharactersInRange:withRTFD:`, `replaceCharactersInRange:withString:`.

replaceCharactersInRange:withRTFD:

```
- (void)replaceCharactersInRange:(NSRange)range
    withRTFD:(NSData *)rtfdData
```

Replaces the characters within the specified `range` of text with the RTFD data `rtfdData`. This message is sent in response to pasting RTFD data from the pasteboard. After replacing the selection, this method rewraps and redisplays the text. See also `replaceCharactersInRange:withRTF:`, `replaceCharactersInRange:withString:`.

replaceCharactersInRange:withString:

```
- (void)replaceCharactersInRange:(NSRange)range
    withString:(NSString *)string
```

Replaces the characters in the specified `range` of text in the text object to be `string`. See also `replaceCharactersInRange:withRTF:`, `replaceCharactersInRange:withRTFD:`.

replaceRange:withRTF:

```
- (void)replaceRange:(NSRange)range withRTF:(NSData *)rtfData
```

Replaces the characters within the specified `range` of text with the RTF data `rtfData`. This message is sent in response to pasting RTF data from the pasteboard. This method is not part of the OpenStep specification. See also `replaceRange:withRTF:`, `RTFFromRange:`.

replaceRange:withRTFD:

```
- (void)replaceRange:(NSRange)range withRTFD:(NSData *)rtfdData
```

Replaces the characters within the specified `range` of text with the RTFD data `rtfdData`. This message is sent in response to pasting RTFD data from the pasteboard. After replacing the selection, this method rewraps and redisplays the text. This method is not part of the OpenStep specification. See also `replaceRange:withRTF:`, `RTFDFromRange:`.

RTFFromRange:

```
- (NSData *)RTFFromRange:(NSRange)range
```

Extracts the specified `range` of RTF text from the `NSText` object and returns a data object initialized with that text. This data is formatted according to the RTF file format. See also `RTFDFromRange:`, `NSData`.

RTFDFromRange:

```
- (NSData *)RTFDFromRange:(NSRange)range
```

Extracts the specified `range` of RTFD text from the `NSText` object and returns an data object initialized with that text. See also `RTFFromRange:`, `NSData`.

scrollRangeToVisible:

```
- (void)scrollRangeToVisible:(NSRange)range
```

Scrolls the `NSText` object so that the `range` of text is visible.

selectAll:

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

Selects all text in the `NSText` object.

selectedRange

```
- (NSRange)selectedRange
```

Returns the range of the selected text in the `NSText` object.

setAlignment:

```
- (void)setAlignment:(NSTextAlignment)mode
```

Sets how the text in the NSText object is aligned between the margins. The return value can be one of the following constants:

- NSLeftTextAlignment
- NSRightTextAlignment
- NSCenterTextAlignment
- NSJustifiedTextAlignment
- NSNaturalTextAlignment

See also alignment.

setBackgroundColor:

```
- (void)setBackgroundColor:(NSColor *)color
```

Sets the background color for the NSText object. color is displayed the next time the text is redrawn; this message doesn't cause the text to be redrawn. See also backgroundColor, textColor, setTextColor:range:.

setDelegate:

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

Makes anObject the NSText object's delegate. In response to user input, the NSText object can send the delegate any of several notification messages. See the class description for more information. See also delegate.

setDrawsBackground:

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

Sets whether the NSText object draws its own background.

setEditable:

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

Sets whether users can edit text in the NSText object. If flag is YES, the text is editable; if NO, the text is read-only. By default, text is editable. See also isEditable, setSelectable:.

setFieldEditor:

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

Sets whether the receiving NSText object is to be used as a field editor. flag indicates whether to end on carriage return, tab, or other terminating character. See also NSFieldFilter() and NSEditorFilter() (Application Kit "Functions" chapter).

setFont:

```
- (void)setFont:(NSFont *)obj
```

Sets the default font for the text object. The entire text is then rewrapped and redrawn. See also setFont:range:, changeFont:.

setFont:range:

```
- (void)setFont:(NSFont *)font range:(NSRange)range
```

Sets the font for the specified range of text in the text object to font. The text is then rewrapped and redrawn. See also setFont:.

setHorizontallyResizable:

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

Sets whether the frame's width can change size horizontally. If flag is YES, the text object's frame rectangle can change in the horizontal dimension in response to additions or deletions of text; if NO, it can't. By default, the text objects can't change size.

setImportsGraphics:

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

Sets whether the text object can import TIFF and EPS images dragged into it by the user. By default, text objects refuse to import such images. See importsGraphics.

setMaxSize:

```
- (void)setMaxSize:(NSSize)newMaxSize
```

Sets the maximum size of the text object to `newMaxSize`. This maximum size is ignored if the text object can't be resized. The default maximum size is {0.0, 0.0}. See also `maxSize`, `setMinSize:`.

setMinSize:

```
- (void)setMinSize:(NSSize)newMinSize
```

Sets the minimum size of the text object to `newMinSize`. This size is ignored if the text object can't be resized. The default minimum size is {0.0, 0.0}. See also `minSize`, `setMaxSize:`.

setRichText:

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

Sets whether the text in the text object allows for multiple values of attributes, such as color and font (that is, RTF and RTFD). See also `isRichText`.

setSelectable:

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

Sets whether users can select text in the text object. By default, text is selectable. See also `isSelectable`, `setSelectedRange:`, `setEditable:`.

setSelectedRange:

```
- (void)setSelectedRange:(NSRange)range
```

Makes the text object the first responder and then selects and highlights a portion of the text described by `range`. See also `setSelectable:`, `NSRange`.

setString:

```
- (void)setString:(NSString *)string
```

Replaces the current text with the text referred to by `aString`. The text object then wraps and redraws the text if autodisplay is enabled. This method doesn't affect the object's frame or bounds rectangle. To resize the text rectangle to make the text entirely visible, use the `sizeToFit` method. See also `setString:`.

setText:

```
- (void)setText:(NSString *)string
```

Replaces the current text with the text referred to by `aString`. The text object then wraps and redraws the text if autodisplay is enabled. This method doesn't affect the object's frame or bounds rectangle. To resize the text rectangle to make the text entirely visible, use the `sizeToFit` method. This method is not part of the OpenStep specification. See also `setString:`,`setText:range:`, `string`, `replaceRange:withRTF:`.

setTextColor:range:

```
- (void)setTextColor:(NSColor *)color range:(NSRange)range
```

Sets the color for the specified `range` of text in the `NSText` object to `color`. See also `textColor`, `setTextColor:`, `setFont:range:`.

setText:range:

```
- (void)setText:(NSString *)string range:(NSRange)range
```

Replaces the characters in the specified `range` of text in the text object to be `string`. This method is not part of the OpenStep specification. See also `setText:`, `NSRange`, `NSString`.

setTextColor:

```
- (void)setTextColor:(NSColor *)color
```

Sets `color` as the display color for the entire text. This method doesn't redraw the text. See also `textColor`, `setTextColor:range:`, `setBackgroundColor:`.

setUsesFontPanel:

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

Sets whether the text object will respond to the `changeFont:` message issued by the Font panel. If enabled, the text object will allow the user to change the font of the selection for a rich text object. For a plain text object, the font for the entire text is changed. If enabled, the text object also updates the Font panel's font selection information. See also `usesFontPanel`.

setVerticallyResizable:

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

Sets whether the text frame can change size vertically. If flag is `YES`, the text object's frame rectangle can change in the vertical dimension in response to additions or deletions of text; if `NO`, it can't. By default, a text object can't change size. See also `isVerticallyResizable`, `setHorizontallyResizable:`.

showGuessPanel:

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

Displays the spell-checker's Show Guess panel, which offers suggested alternate spellings for a word that's suspected of being misspelled. The user can either accept one of the alternates, add the word to a local dictionary, or skip the word. A word becomes a candidate for the Guess panel's actions by being selected as the result of the text object's receiving a `checkSpelling:` message. See also `checkSpelling:`.

sizeToFit

```
- (void)sizeToFit
```

Modifies the frame rectangle to completely display the text. This is often used with text objects in an `NSScrollView` object. The `setHorizontallyResizable:` and `setVerticallyResizable:` methods determine whether the text object will resize horizontally or vertically (by default, it won't change size in either dimension). After receiving a `calcLine`

(NSCStringText) message, a text object that is the document view of an NSScrollView sends itself a sizeToFit message. See also calcLine (NSCStringText).

string

```
- (NSString *)string
```

Returns the contents of the text object as an immutable string object. See also setText:.

subscript:

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

Subscripts the current selection. The text is then rewrapped and redrawn. The text is subscripted by 40% of the selection's font height. See also superscript:, unscript:.

superscript:

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

Superscripts the current selection. The text is then rewrapped and redrawn. The text is superscripted by 40% of the selection's font height. See also subscript:, unscript:.

textColor

```
- (NSColor *)textColor
```

Returns the text object's color for drawing text. See also setTextColor:.

toggleRuler:

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

Controls the display of the ruler. This method has effect only if the receiving text object is a rich text object, and is a subview of an NSScrollView. This method causes the NSScrollView to display a ruler if one isn't already

present, or to remove the ruler if one is. When the ruler is displayed, its settings reflect the paragraph style of the paragraph containing the selection. `sender` is the `id` of the sending object. See also `isRulerVisible`.

`underline:`

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

Adds an underline to the selected text if one doesn't already exist or removes the underline if it does. If the selection is zero-width, this method affects the underline attribute of text that's subsequently entered at the insertion point. `sender` is the `id` of the sending object. See also `subscript:`.

`unscript:`

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

Removes superscript or subscript in the current selection. The text is then rewrapped and redrawn. See also `superscript:`, `subscript:`.

`usesFontPanel`

```
- (BOOL)usesFontPanel
```

Returns `YES` if the text object will respond to the Font panel, `NO` if not. The default is `YES`. See also `setUsesFontPanel:`.

`writeRTFDToFile:atomically:`

```
- (BOOL)writeRTFDToFile:(NSString *)path atomically:(BOOL)flag
```

Writes RTFD data from the receiving text object to the file package specified by `path`. `flag` determines whether writing occurs atomically. Returns whether or not the operation succeeded. See also `readRTFDFromFile:`.

Methods Implemented by the Delegate

`textDidBeginEditing:`

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

Sent by the default notification center to the delegate; `aNotification` is always `NSTextDidBeginEditingNotification`. If the delegate implements this method, it's automatically registered to receive this notification.

textDidChange:

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

Sent by the default notification center to the delegate; `aNotification` is always `NSTextDidChangeNotification`. If the delegate implements this method, it's automatically registered to receive this notification.

textDidEndEditing:

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

Sent by the default notification center to the delegate; `aNotification` is always `NSTextDidEndEditingNotification`. If the delegate implements this method, it's automatically registered to receive this notification.

textShouldBeginEditing:

```
- (BOOL)textShouldBeginEditing:(NSText *)textObject
```

Sent directly by `textObject` to the delegate. Informs delegate of an impending textual change. A return value of `YES` means go ahead and make the change.

textShouldEndEditing:

```
- (BOOL)textShouldEndEditing:(NSText *)textObject
```

Sent directly by `textObject` to the delegate. Warns delegate of the impending loss of first responder status. A return value of `YES` means go ahead and change status.

NSTextField

Inherits From:	NSControl : NSView : NSResponder : NSObject
Conforms To:	NSCoding (NSResponder) NSObject (NSObject)
Declared In:	AppKit / NSTextField.h

Class Description

An `NSTextField` is an `NSControl` object that can display a piece of text that a user can select or edit, and which sends an action message to its target if the user presses the Return key while editing. An `NSTextField` can also be linked to other `NSTextField`s, so that when the user presses Tab or Shift-Tab, the object assigned as the "next" or "previous" field gets a message to select its text.

An `NSTextField` is a good alternative to an `NSText` object for small regions of editable text, since the display of the `NSTextField` is achieved by using a global `NSText` object shared by objects all over your application, which saves on memory usage. Each `NSWindow` also has an `NSText` object used for editing of `NSTextField`s (and `NSTextFieldCell`s in `NSMatrix`s). An `NSWindow`'s global `NSText` object is called a *field editor*, since it's attached as needed to an `NSTextField` to perform its editing. `NSTextField` allows you to specify an object to act as an indirect delegate to the field editor; the `NSTextField` itself acts as the `NSText` delegate if it needs to, then passes the delegate method on to its own `NSText` delegate.

Method Types

Activity	Class Method
Setting user access to text	– isEditable – isSelectable – setEditable: – setSelectable:
Editing text	– selectText:
Setting Tab key behavior	– nextText – previousText – setNextText: – setPreviousText:
Assigning a delegate	– delegate – setDelegate:
Modifying graphic attributes	– backgroundColor – drawsBackground – isBezeled – isBordered – setBackgroundColor: – setBezeled: – setBordered: – setDrawsBackground: – setTextColor: – textColor
Target and action	– errorAction – setErrorAction:
Handling events	– acceptsFirstResponder – textDidBeginEditing: – textDidChange: – textDidEndEditing: – textShouldBeginEditing: – textShouldEndEditing:

Instance Methods

acceptsFirstResponder

```
- (BOOL)acceptsFirstResponder
```

Returns YES if text is editable or selectable, and returns NO otherwise.

backgroundColor

```
- (NSColor *)backgroundColor
```

Returns the background color of the background. See also `setBackgroundColor:`.

delegate

```
- (id)delegate
```

Returns the delegate for messages from the field editor. See also `setDelegate:`.

drawsBackground

```
- (BOOL)drawsBackground
```

Returns YES if the `NSTextField` draws its own background. See also `setDrawsBackground:`.

errorAction

```
- (SEL)errorAction
```

Returns the action sent to the text field target when the user enters an illegal value for the cell type (as set by `NSCell`'s `setEntryType:` method and checked by `NSCell`'s `isEntryAcceptable:` method). See also `setErrorAction:`.

isBezeled

```
- (BOOL)isBezeled
```

Returns YES if the text is drawn in a bezeled frame. See also `setBezeled:`, `isBordered`.

isBordered

```
- (BOOL)isBordered
```

Returns YES if the text has a solid black border around it. See also `setBordered:`, `isBezeled`.

isEditable

```
- (BOOL)isEditable
```

Returns YES if the text is editable and selectable, NO if the text is not editable (though it may be selectable). See also `setEditable:`, `isSelectable`.

isSelectable

```
- (BOOL)isSelectable
```

Returns YES if the text is selectable, NO otherwise. Selectable text isn't necessarily editable. See also `setSelectable:`, `isEditable`.

nextText

```
- (id)nextText
```

Returns the object whose text is selected when the user presses Tab while editing the text field. If that object responds to the `selectText:` message, the current text field is deactivated and `selectText:` is sent to the next text. See also `setNextText:`, `previousText`.

previousText

```
- (id)previousText
```

Returns the object that is selected when the user presses Shift-Tab while editing the text field. If that object responds to the `selectText:` message, the current text field is deactivated and `selectText:` is sent to the previous text. See also `setPreviousText:`, `nextText`.

selectText:

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

Selects the entire contents of the receiving text field if it is editable or selectable. If the text field isn't in a view hierarchy, it has no effect. See also `isSelectable`.

setBackgroundColor:

```
- (void)setBackgroundColor:(NSColor *)aColor
```

Sets the color of the background to aColor. See also backgroundColor.

setBezeled:

- (void)setBezeled:(BOOL)flag

If flag is YES, the NSTextField is drawn with a bezel around the edge; if flag is NO, nothing is drawn around the text. Bezels and borders are mutually exclusive. See also isBezeled, setBordered:.

setBordered:

- (void)setBordered:(BOOL)flag

If flag is YES, a 1-pixel black border will be drawn around the text; if flag is NO, nothing is drawn around the text. Borders and bezels are mutually exclusive. Does not affect the background color. See also isBordered, setBezeled:.

setDelegate:

- (void)setDelegate:(id)anObject

Sets the delegate for messages from the field editor to anObject. See also delegate.

setDrawsBackground:

- (void)setDrawsBackground:(BOOL)flag

Sets whether the text field draws its own background color. See also drawsBackground.

setEditable:

- (void)setEditable:(BOOL)flag

If flag is YES, then the text in the text field is made both editable and selectable. If flag is NO, the text can't be edited, and is restored to its previous selectable state. For example, if a t ext field is set selectable but not editable, then made editable for a time, then made not editable again, it will remain selectable. To guarantee that text will be neither editable nor selectable, simply turn off selectability explicitly. See also isEditable, setSelectable:.

setErrorAction:

```
- (void)setErrorAction:(SEL)aSelector
```

Sets the action sent to the text field's target when the user enters an illegal value for the NSCell's entry type as set by NSCell's setEntryType: method and checked by NSCell's isEntryAcceptable: method. See also errorAction.

setNextText:

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

Sets the object selected when the user presses Tab while editing the text field's text. anObject should respond to the selectText: message. If anObject also responds to both selectText: and setPreviousText:, it is sent setPreviousText: with the receiving text field as the argument; this builds a two-way connection, so that pressing Tab in the text field selects anObject's text, and pressing Shift-Tab in anObject selects the text field's text. See also nextText.

setPreviousText:

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

Sets the object selected when the user presses Shift-Tab while editing the text field's text. anObject should respond to the selectText: message. Your code shouldn't need to use this method directly, since it's invoked automatically by setNextText:. In deference to setNextText:, this method doesn't build a two-way connection. See also previousText.

setSelectable:

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

If flag is YES, then the text field is made selectable but *not* editable (use setEditable: to make text both selectable and editable). If flag is NO, then the text is made neither editable nor selectable. See also isSelectable, setEditable:.

setTextColor:

```
- (void)setTextColor:(NSColor *)aColor
```

Sets the text field's text color to `aColor`. This method doesn't cause the text to be redrawn. See also `textColor`.

textColor

```
- (NSColor *)textColor
```

Returns the text field's text color. See also `setTextColor:`.

textDidBeginEditing:

```
- (void)textDidBeginEditing:(NSNotification *)notification
```

Invoked when there's a change in the text after the receiver gains first responder status. The default behavior passes this message on to the text delegate by posting the notification `NSControlTextDidEndEditingNotification` with the receiving object and, in the notification's dictionary, the text object (with the key `NSFieldEditor`) to the default notification center.

textDidChange:

```
- (void)textDidChange:(NSNotification *)notification
```

Invoked upon a key-down event or paste operation that changes the receiver's contents. The default behavior passes this message on to the text delegate by posting the `NSControlTextDidChangeNotification` notification with the receiving object and, in the notification's dictionary, the text object (with the key `NSFieldEditor`) to the default notification center.

textDidEndEditing:

```
- (void)textDidEndEditing:(NSNotification *)notification
```

Invoked when text editing ends. The default behavior is to pass this message on to the text delegate by posting the notification `NSControlTextDidEndEditingNotification` with the receiving object and, in the notification's dictionary, the text object (with the key `NSFieldEditor`) to the default notification center.

textShouldBeginEditing:

```
- (BOOL)textShouldBeginEditing:(NSText *)textObject
```

Invoked to let the text field respond to impending changes to its text and then forwarded to the text delegate. See also `textShouldEndEditing:`.

textShouldEndEditing:

```
- (BOOL)textShouldEndEditing:(NSText *)textObject
```

Invoked to let the text field respond to impending loss of first responder status and then forwarded to the text delegate. See also `textShouldBeginEditing:`.

NSTextFieldCell

Inherits From:	NSActionCell : NSCell : NSObject
Conforms To:	NSCoding, NSCopying (NSCell) NSObject (NSObject)
Declared In:	AppKit / NSTextFieldCell.h

Class Description

`NSCell`s display text or images—an `NSTextFieldCell` is simply an `NSCell` that displays text and that keeps track of its background and text colors. Normally, the `NSCell` class assumes white as the background when bezeled, and light gray otherwise, and the text is always black. With `NSTextFieldCell`, you can specify those colors.

Method Types

Activity	Class Method
Modifying raphic attributes	– backgroundColor – setBackgroundColor: – setDrawsBackground: – setTextColor: – setUpFieldEditorAttributes: – textColor

Instance Methods

backgroundColor

```
– (NSColor *)backgroundColor
```

Returns the background color. See also `setBackgroundColor:`, `drawsBackground`.

drawsBackground

```
– (BOOL)drawsBackground
```

Returns YES if the text field cell draws its own background. See also `setDrawsBackground:`.

setBackgroundColor:

```
– (void)setBackgroundColor:(NSColor *)aColor
```

Sets the background color to `aColor`. See also `backgroundColor`.

setDrawsBackground:

```
– (void)setDrawsBackground:(BOOL)flag
```

If `flag` is YES, the `NSTextFieldCell` draws its own background. See also `drawsBackground`.

setTextColor:

```
- (void)setTextColor:(NSColor *)aColor
```

Sets the color of the text to `aColor`. See also `textColor`.

setUpFieldEditorAttributes:

```
- (NSText *)setUpFieldEditorAttributes:(NSText *)textObject
```

Sets the background and text colors of `textObject` to those of the `NSTextFieldCell`, and returns `textObject`. `textObject` should respond to the messages `setBackgroundColor:`, and `setTextColor:`. You rarely need to override this method; you never need to invoke it.

textColor

```
- (NSColor *)textColor
```

Returns the color of the text. See also `setTextColor:`.

NSView

Inherits From:	NSResponder : NSObject
Conforms To:	NSCoding (NSResponder) NSObject (NSObject)
Declared In:	AppKit / NSView.h AppKit / NSClipView.h

Class Description

`NSView` is an abstract class that provides its subclasses with a structure for drawing and for handling events. Any application that needs to display, print, or receive events must use `NSView` objects.

To be displayed, a view must be placed in a window represented by an `NSWindow` object. All the views within a window are arranged in a hierarchy, with each view having a single *superview* and zero or more *subviews*. Each view

has its own drawing area, and its own coordinate system expressed as a transformation of its superview's coordinate system. An `NSView` object can scale, translate, or rotate its coordinates, or flip the polarity of its y-axis.

An `NSView` keeps track of its size and location in two ways: as a frame rectangle (expressed in its superview's coordinate system) and as a bounds rectangle (expressed in its own coordinate system). Both are represented by `NSRect` structures.

Subclasses of `NSView` typically override `drawRect:` to implement an object's distinctive appearance. They also frequently override one or more of `NSView`'s or `NSResponder`'s event-handling methods, to react to the user's manipulations of the mouse and keyboard.

Note – Do not add or remove views from the view list during `display`*xxx* messages. Adding or removing views during display will raise `NSInternalInconsistancyException`.

Method Types

Activity	Class Method
Initializing NSView objects	– initWithFrame:
Managing the NSView hierarchy	– addSubview: – addSubview:positioned:relativeTo: – ancestorSharedWithView: – isDescendantOf: – opaqueAncestor – removeFromSuperview – replaceSubview:with: – sortSubviewsUsingFunction:context: – subviews – superview – window – viewWillMoveToSuperview: – viewWillMoveToWindow:
Modifying the frame rectangle	– frameRotation – frame – rotateByAngle: – setFrame: – setFrameOrigin: – setFrameRotation: – setFrameSize:
Modifying the coordinate system	– boundsRotation – bounds – isFlipped – isRotatedFromBase – isRotatedOrScaledFromBase – scaleUnitSquareToSize: – setBounds: – setBoundsOrigin: – setBoundsRotation: – setBoundsSize: – translateOriginToPoint:

Activity	Class Method
Converting coordinates	– centerScanRect: – convertPoint:fromView: – convertPoint:toView: – convertRect:fromView: – convertRect:toView: – convertSize:fromView: – convertSize:toView:
Notifying ancestor views	– postsBoundsChangedNotifications – postsFrameChangedNotifications – setPostsBoundsChangedNotifications: – setPostsFrameChangedNotifications:
Resizing subviews	– resizeSubviewsWithOldSize: – setAutoresizesSubviews: – autoresizesSubviews – setAutoresizingMask: – autoresizingMask – resizeWithOldSuperviewSize:
Graphics state objects	– allocateGState – releaseGState – gState – renewGState – setUpGState
Focusing	+ focusView – lockFocus – unlockFocus
Displaying	– canDraw – display – displayIfNeeded – displayIfNeededIgnoringOpacity – displayIfNeededInRect: – displayIfNeededInRectIgnoringOpacity: – displayRect: – displayRectIgnoringOpacity: – drawRect: – visibleRect – isOpaque – needsDisplay – setNeedsDisplay: – setNeedsDisplayInRect: – shouldDrawColor

Activity	Class Method
Scrolling	– adjustScroll: – autoscroll: – enclosingScrollView – reflectScrolledClipView: – scrollClipView:toPoint: – scrollPoint: – scrollRect:by: – scrollRectToVisible:
Managing the cursor	– addCursorRect:cursor: – discardCursorRects – removeCursorRect:cursor: – resetCursorRects
Assigning a tag	– tag – viewWithTag:
Aiding event handling	– acceptsFirstMouse: – hitTest: – mouse:inRect: – performKeyEquivalent: – removeTrackingRect: – shouldDelayWindowOrderingForEvent: – addTrackingRect:owner:userData:assumeInside:
Dragging	– dragFile:fromRect:slideBack:event: – dragImage:at:offset:event:pasteboard:source: slideBack: – registerForDraggedTypes: – unregisterDraggedTypes

Activity	Class Method
Printing	– dataWithEPSInsideRect: – fax: – print: – writeEPSInsideRect:toPasteboard:
Pagination	– adjustPageHeightNew:top:bottom:limit: – adjustPageWidthNew:left:right:limit: – heightAdjustLimit – knowsPagesFirst:last: – locationOfPrintRect: – rectForPage: – widthAdjustLimit
Writing conforming PostScript	– addToPageSetup – beginPage:label:bBox:fonts: – beginPageSetupRect:placement: – beginPrologueBBox:creationDate:createdBy: fonts:forWhom:pages:title: – beginSetup – beginTrailer – drawPageBorderWithSize: – drawSheetBorderWithSize: – endHeaderComments – endProlog – endSetup – endPageSetup – endPage – endTrailer

Class Methods

`focusView`

```
+ (NSView *)focusView
```

Returns the currently focused view object, or `nil` if none.

Instance Methods

acceptsFirstMouse:

```
- (BOOL)acceptsFirstMouse:(NSEvent *)theEvent
```

This method returns YES if an initial mouse-down event in the NSView—an event that causes the NSView's NSWindow to become the key window—is sent to the NSView (through a mouseDown: message). If only those mouse-downs that occur when the NSView's NSWindow is already the keywindow are sent, this returns NO (the default). The only way to change the default behavior is to implement this method in an NSView subclass. See also mouseDown: (NSResponder), hitTest:, mouse:inRect:, performKeyEquivalent:, removeTrackingRect:, shouldDelayWindowOrderingForEvent:, addTrackingRect:owner:userData:assumeInside:.

addCursorRect:cursor:

```
- (void)addCursorRect:(NSRect)aRect cursor:(NSCursor *)anObject
```

Adds cursor rectangle aRect for cursor anObject to the NSView. When the user moves the mouse within the rectangle specified by aRect, the cursor object that the mouse controls changes to anObject, which must be an NSCursor object. The rectangle is given in the NSView's coordinate system; however, the rectangle *isn't* automatically clipped to the NSView's frame—it's possible to create a cursor rectangle that extends beyond the NSView. You should also note that cursor rectangles don't work well in rotated NSViews. You never invoke this method directly from your application. It should only be used as part of the implementation of the resetCursorRects method. See also discardCursorRects, removeCursorRect:cursor:, resetCursorRects.

addSubview:

```
- (void)addSubview:(NSView *)aView
```

Adds aView to the NSView's list of subviews. The new subview will be displayed on top of its siblings. The receiving NSView is also made aView's next responder. This message should not be sent to an NSClipview object. Use NSClipview's setDocumentView: method. See also addSubview:positioned:relativeTo:, isDescendantOf:,

opaqueAncestor, removeFromSuperview, replaceSubview:with:, sortSubviewsUsingFunction:context:, subviews, superview, window, viewWillMoveToWindow:.

addSubview:positioned:relativeTo:

```
- (void)addSubview:(NSView *)aView
    positioned:(NSWindowOrderingMode)place
    relativeTo:(NSView *)otherView
```

Puts aView into the receiving NSView's list of subviews, so that it will be displayed immediately above or below otherView, as indicated by place. place can be one of the following values:

- NSWindowAbove
- NSWindowBelow
- NSWindowOut

If otherView is nil or isn't in the subview list, aView is added above or below all its siblings. This message should not be sent to an NSClipview object. Use NSClipview's setDocumentView: method. See also addSubview:, NSWindowOrderingMode (Display Postscript Types and Constants).

addToPageSetup

```
- (void)addToPageSetup
```

Lets you adjust for differences in the graphics state between the screen and the printer. See also beginPage:label:bBox:fonts:, beginPageSetupRect:placement:.

addTrackingRect:owner:userData:assumeInside:

```
-(NSTrackingRectTag)addTrackingRect:(NSRect)aRect
    owner:(id)anObject
    userData:(void *)data
    assumeInside:(BOOL)flag
```

Adds a tracking rectangle (aRect) owned by anObject to the receiving NSView. flag indicates whether the tracking rectangle will be only inside the NSView. Returns a unique tag that identifies the tracking rectangle. See also removeTrackingRect:.

adjustPageHeightNew:top:bottom:limit:

```
- (void)adjustPageHeightNew:(float *)newBottom top:(float)oldTop
```

Assists automatic pagination of the view object. See also `adjustPageWidthNew:left:right:limit:`.

adjustPageWidthNew:left:right:limit:

```
- (void)adjustPageWidthNew:(float *)newRight left:(float)oldLeft
    right:(float)oldRight limit:(float)rightLimit
```

Assists automatic pagination of the view object. See also `adjustPageHeightNew:top:bottom:limit:`.

adjustScroll:

```
- (NSRect)adjustScroll:(NSRect)newVisible
```

Lets the view object adjust the scroll position of a document. This method is invoked by a `NSClipView` immediately prior to scrolling its document view. You may want to override it to provide specific scrolling behavior. `newVisible` will be the visible rectangle after the scroll. You might use this for scrolling through a table, for example a spreadsheet. You could modify `newVisible->origin` such that the scroll would fall on column or row boundaries. The default implementation returns `newVisible`. See also `scrollRect:by:`.

allocateGState

```
- (void)allocateGState
```

Explicitly tells the `NSView` to allocate a graphics state object. Graphics state objects are Display PostScript objects that contain the entire state of the graphics environment. They are used by the Application Kit as a caching mechanism to save PostScript code used for focusing. This is purely a performance optimization. You can allocate a graphics state object for `NSView`s that will be focused on repeatedly, but you should exercise some discretion as these state objects can take a fair amount of memory. The graphics state object will be freed automatically when the `NSView` is freed. See also `releaseGState`, `gState`, `renewGState`, `setUpGState`.

ancestorSharedWithView:

```
- (NSView *)ancestorSharedWithView:(NSView *)aView
```

Returns the ancestor view shared by `aView` and the receiver. Returns `self` if `aView` and the receiving view are identical, or if the receiving view is the ancestor of `aView`. Returns `aView` if it is the superview of the receiving view. Returns `nil` in any other case. See also `addSubview:`.

autoresizesSubviews

```
- (BOOL)autoresizesSubviews
```

Returns `YES` if the `NSView` automatically notifies subviews of resizing, and `NO` otherwise. See also `setAutoresizesSubviews:`, `setAutoresizingMask:`, `autoresizingMask`, `resizeSubviewsWithOldSize:`, `resizeWithOldSuperviewSize:`.

autoresizingMask

```
- (unsigned int)autoresizingMask
```

Returns the `NSView`'s autoresizing mask. The mask is used to determine how the `NSView` is automatically resized when its superview is resized. For the mask to have an effect, the superview must be set to resize its subviews; this is done through the `setAutoresizeSubviews:` method. The autoresizing masks are

- `NSViewNotSizable`
- `NSViewMinXMargin`
- `NSViewWidthSizable`
- `NSViewMaxXMargin`
- `NSViewMinYMargin`
- `NSViewHeightSizable`
- `NSViewMaxYMargin`

See also `setAutoresizingMask:`, `autoresizesSubviews`.

autoscroll:

```
- (BOOL)autoscroll:(NSEvent *)theEvent
```

Scrolls in response to a mouse-dragged event.

beginPage:label:bBox:fonts:

```
- (void)beginPage:(int)ordinalNum label:(NSString *)aString
    bBox:(NSRect)pageRect fonts:(NSString *)fontNames
```

Writes a page separator.

beginPageSetupRect:placement:

```
- (void)beginPageSetupRect:(NSRect)aRect
placement:(NSPoint)location
```

Writes the beginning of a page setup section. It outputs a PostScript `save`, and generates the initial coordinate transformation to set up this view for printing the `aRect` rectangle within the view. This method does a `lockFocus` on the view, which must be balanced in `endPage` by an `unlockFocus`. The `save` output here should be balanced by a PostScript `restore` in `endPage`. `aRect` is the rectangle in the view's coordinates that is being printed. `location` is the offset in page coordinates of the rectangle on the physical page.

beginPrologueBBox:creationDate:createdBy: fonts:forWhom:pages:title:

```
- (void)beginPrologueBBox:(NSRect)boundingBox
    creationDate:(NSString *)dateCreated
    createdBy:(NSString *)anApplication fonts:(NSString *)fontNames
    forWhom:(NSString *)user pages:(int)numPages
    title:(NSString *)aTitle
```

Writes the start of the conforming PostScript header for a print job. `boundingBox` is the bounding box of the document. This rectangle should be in the default PostScript coordinate system on the page. If it is unknown, `boundingBox` should be `NULL` and the system will accumulate it as pages are printed. `dateCreated` is an ASCII string containing a human readable date. If `dateCreated` is `NULL` the current date is used. `anApplication` is a string containing the name of the document creator.

`user` is a string containing the name of the person the document is being printed for. If `NULL`, the login name of the user is used. `numPages` specifies the number of pages in the document. If unknown at the beginning of printing, `numPages` should have a value of -1. In this case the pages are counted as they are generated and the resulting count is written in the trailer. `aTitle` is a

string specifying the title of the document. If `aTitle` is `NULL`, then the title of the `NSView`'s `NSWindow` is used. If the `NSWindow` has no title, "Untitled" is output. See also `addToPageSetup`.

beginSetup

```
- (void)beginSetup
```

Writes the beginning of the job setup section, which begins with a %%BeginSetup comment and includes a %%PaperSize comment declaring the type of paper being used. This section of the output is intended for device setup or general initialization code. See also `addToPageSetup`.

beginTrailer

```
- (void)beginTrailer
```

Writes the beginning of the conforming PostScript trailer for the print job. See also `addToPageSetup`.

boundsRotation

```
- (float)boundsRotation
```

Returns the rotation of the `NSView`'s coordinate system. If the `NSView`'s coordinate system has been rotated, this angle will be the accumulation of all `setBoundsRotation:` messages; otherwise, it will be 0.0. See also `setBoundsRotation:`, `bounds`, `setBoundsOrigin:`, `setBoundsSize:`.

bounds

```
- (NSRect)bounds
```

Returns the `NSView`'s bounds rectangle. See also `setBounds:`, `boundsRotation`.

canDraw

```
- (BOOL)canDraw
```

Informs you of whether drawing will have any result. You only need to send this message when you want to draw, but are not invoking one of the display methods. You should not draw or send the `lockFocus:` message if this returns `NO`. This method returns `YES` if your `NSView` has an `NSWindow` object, your `NSView`'s `NSWindow` object has a corresponding window on the Window Server, and your `NSWindow` object is enabled for display; otherwise it returns `NO`.

centerScanRect:

```
- (NSRect)centerScanRect:(NSRect)aRect
```

Converts the corners of `aRect` to lie on the center of device pixels. This is useful in compensating for PostScript overscanning when the coordinate system has been scaled. This routine converts the given rectangle to device coordinates, adjusts the rectangle to lie in the center of the pixels, and converts the resulting rectangle back to the `NSView`'s coordinate system. Returns the transformed `aRect`. See also `convertRect:fromView:`, `convertRect:toView:`, `convertPoint:fromView:`, `convertPoint:toView:`, `convertSize:toView:`.

convertPoint:fromView:

```
- (NSPoint)convertPoint:(NSPoint)aPoint fromView:(NSView *)aView
```

Converts a point from `aView`'s coordinate system to the coordinate system of the receiving `NSView`. If `aView` is `nil`, then this method converts from `NSWindow`'s base coordinates. Both `aView` and the receiving `NSView` must belong to the same `NSWindow`. Returns the converted `aPoint`. See also `convertPoint:toView:`, `NSPoint` (Foundation Kit "Types and Constants" chapter).

convertPoint:toView:

```
- (NSPoint)convertPoint:(NSPoint)aPoint toView:(NSView *)aView
```

Converts a point from the receiving `NSView`'s coordinate system to the coordinate system of `aView`. If `aView` is `nil`, then this method converts to the `NSWindow`'s base coordinates. Both `aView` and the receiving `NSView` must

belong to the same NSWindow. Returns the converted aPoint. See also convertPoint:fromView:, NSPoint (Foundation Kit "Types and Constants" chapter).

convertRect:fromView:

```
- (NSRect)convertRect:(NSRect)aRect fromView:(NSView *)aView
```

Converts aRect from aView's coordinate system to the coordinate system of the receiving NSView. Both aView and the receiving NSView must belong to the same NSWindow. Returns the converted aRect. See also convertRect:toView:, NSRect (Foundation Kit "Types and Constants" chapter).

convertRect:toView:

```
- (NSRect)convertRect:(NSRect)aRect toView:(NSView *)aView
```

Converts aRect from the receiving NSView's coordinate system to the coordinate system of aView. Both aView and the receiving NSView must belong to the same NSWindow. See also convertRect:fromView:, NSRect (Foundation Kit "Types and Constants" chapter).

convertSize:fromView:

```
- (NSSize)convertSize:(NSSize)aSize fromView:(NSView *)aView
```

Converts aSize from the coordinate system of aView to the coordinate system of the receiving NSView. Both aView and the receiving NSView must belong to the same NSWindow. See also convertSize:toView:, NSSize (Foundation Kit "Types and Constants" chapter).

convertSize:toView:

```
- (NSSize)convertSize:(NSSize)aSize toView:(NSView *)aView
```

Converts aSize from the receiving NSView's coordinate system to the coordinate system of aView. Both aView and the receiving NSView must belong to the same NSWindow. See also convertSize:fromView:, NSSize (Foundation Kit "Types and Constants" chapter).

dataWithEPSInsideRect:

```
- (NSData *)dataWithEPSInsideRect:(NSRect)aRect
```

Returns the encapsulated PostScript inside `rect` as a data object. See also `EPSOperationWithView:insideRect:toData:` (`NSPrintOperation`),.

discardCursorRects

```
- (void)discardCursorRects
```

Removes the `NSView`'s cursor rectangles. You shouldn't need to invoke this method directly; it's invoked automatically before the `NSView`'s cursor rectangles are reset. See also `addCursorRect:cursor:`.

display

```
- (void)display
```

Displays the receiving view and its subviews, using each view's bounds rectangle. See also `displayIfNeeded`, `displayIfNeededIgnoringOpacity`, `displayRect:`, `displayRectIgnoringOpacity:`, `canDraw`, `isOpaque`, `needsDisplay`, `setNeedsDisplay:`.

displayIfNeeded

```
- (void)displayIfNeeded
```

Descends the `NSView` hierarchy starting at the receiving `NSView` and sends a `display` message to each opaque `NSView` that needs to be displayed. This is useful when you wish to disable display in the `NSWindow`, modify a series of `NSViews`, and then display only the ones whose appearance has changed. See also `display`.

displayIfNeededIgnoringOpacity

```
- (void)displayIfNeededIgnoringOpacity
```

Conditionally displays the receiving view and its subviews, regardless of opacity. Display is needed if the view contents have changed. See also `displayIfNeeded`, `display`.

displayIfNeededInRect:

```
- (void)displayIfNeededInRect:(NSRect)aRect
```

Displays the receiving view and its subviews, within `aRect`, if needed. See also `displayIfNeeded`, `displayRect:`.

displayIfNeededInRectIgnoringOpacity:

```
- (void)displayIfNeededInRectIgnoringOpacity:(NSRect)aRect
```

Displays the receiving view and its subviews within `aRect` if necessary, ignoring opacity. See also `displayIfNeededInRect:`, `displayIfNeededIgnoringOpacity`.

displayRect:

```
– (void)displayRect:(NSRect)aRect
```

Displays the receiving `NSView` and its subviews (if opaque) within `aRect`. See also `displayRectIgnoringOpacity:`, `display`.

displayRectIgnoringOpacity:

```
– (void)displayRectIgnoringOpacity:(NSRect)aRect
```

Displays the receiving `NSView` and its subviews, regardless of opacity, within `aRect`. See also `displayRect:`.

dragFile:fromRect:slideBack:event:

```
– (BOOL)dragFile:(NSString *)filename fromRect:(NSRect)rect
    slideBack:(BOOL)slideFlag event:(NSEvent *)event
```

Causes a file icon represented by an `NSImage`-derived object to be dragged from the `NSView` to any application that accepts files. This method only makes sense when invoked from within an implementation of the `mouseDown:` method. The arguments are:

- `filename` is the complete name (including path) of the file to be dragged.
- `rect` describes the position of the icon in the `NSView`'s coordinates.

- slideFlag indicates whether the icon should slide back to its position in the NSView if the file is not accepted. If slideFlag is YES and filename is not accepted and the user has not disabled icon animation, the icon will slide back; otherwise it will not.
- event is the mouse-down event record (or a copy).

This method returns YES if the NSView successfully initiated the file dragging session; otherwise it returns NO. See also dragImage:at:offset:event:pasteboard:source: slideBack:, registerForDraggedTypes:, unregisterDraggedTypes.

dragImage:at:offset:event:pasteboard:source: slideBack:

```
- (void)dragImage:(NSImage *)anImage at:(NSPoint)viewLocation
    offset:(NSSize)initialOffset event:(NSEvent *)event
    pasteboard:(NSPasteboard *)pboard source:(id)sourceObject
    slideBack:(BOOL)slideFlag
```

Initiates an image-dragging session, dragging anImage from viewLocation. initialOffset is the difference in the mouse location from the mouse-down. pboard is the pasteboard holding the data. sourceObject is the object receiving NSDraggingSource messages. slideFlag determines whether the NSImage should slide back if rejected.

Instigates an image-dragging session. This method only makes sense when invoked from within an implementation of the mouseDown: method. The arguments are:

- anImage is the NSImage (contained within the NSView) that's being dragged.
- location is the NSImage's origin in the NSView's coordinate system.
- initialOffset gives the mouse's current location relative to the mouse-down location.
- event is the mouse-down that started the dragging session.
- pboard is the pasteboard that holds the data that the NSImage represents.
- sourceObject is the object that receives NSDraggingSource messages.
- slideFlag determines whether the NSImage should slide back if it's rejected.

Before invoking this method, the NSView must place the data that's being dragged on the drag pasteboard. To do this, it must get the pasteboard, declare the type of data that it's placing, and then place the data:

```
/* You always use the NSDragPboard pasteboard when dragging. */
NSPasteboard *pboard = [Pasteboard newName:NSDragPboard];

/* Declare the type of data and place it on the pasteboard. */
[pboard declareTypes:... owner:...];
[pboard setData:... forType:...];

/* Now invoke dragImage:.*/
[self dragImage:... at:... offset:... event:...
       pasteboard:pboard source:... slideBack:...];
```

This method returns YES if the NSView successfully initiated the file dragging session; otherwise it returns NO.

If you ask for events inside the mouseDown: method before invoking this method (if, for example, you're making sure that the image is really being dragged before initiating a dragging session), you must copy the mouse-down event *before* asking for more events. You then pass the copy as the argument to the event: keyword of this method. See also dragFile:fromRect:slideBack:event:.

drawPageBorderWithSize:

```
- (void)drawPageBorderWithSize:(NSSize)borderSize
```

Implemented by subclasses to draw in margins for example, borders and numbering. borderSize is the size of the border.

drawRect:

```
- (void)drawRect:(NSRect)rect
```

Implemented by subclasses to supply drawing instructions for the NSView. Each NSView subclass must override this method to draw itself within its frame rectangle. The default implementation of this method does nothing. rect is a rectangle indicating the region within the NSView that needs to be drawn. This method is invoked by the display method; you shouldn't send a drawRect: message directly to an NSView.

Your implementation of drawRect: doesn't need to invoke lockFocus; focus is already locked on an object when it's told to draw itself. See also display.

drawSheetBorderWithSize:

```
- (void)drawSheetBorderWithSize:(NSSize)borderSize
```

Implemented by subclasses to draw in margins, for example borders and numbering. borderSize is the size of the border. This method is invoked by beginPageSetupRect:placement:.

enclosingScrollView

```
- (NSScrollView *)enclosingScrollView
```

Returns the scroll view that encloses the receiving view, or nil if the view isn't in a scroll view. In the unlikely event that the view is in nested scroll views, this method returns the "closest" one going up the view hierarchy.

endHeaderComments

```
- (void)endHeaderComments
```

Writes out the end of a conforming PostScript header. It prints out the %%EndComments line and then the start of the prologue, including the Application Kit's standard printing package. The prologue should contain definitions global to a print job. This method is indirectly invoked by print: or fax: after beginPrologueBBox:creationDate:createdBy: fonts:forWhom:pages:title:, and before endPrologue.

endPage

```
- (void)endPage
```

Writes the end of a conforming PostScript page. This method is invoked after each page is printed. It performs an unlockFocus to balance the lockFocus done in beginPageSetupRect:placement:. It also generates a PostScript showpage and a restore.

endPageSetup

```
- (void)endPageSetup
```

Writes the end of a page setup section, which begins with a %%EndPageSetup comment. This method is invoked by `print:` and `fax:` just after `beginPageSetupRect:placement:` is invoked.

endPrologue

```
- (void)endPrologue
```

Writes out the end of the conforming PostScript prologue. This method is invoked by `print:` and `fax:` after the prologue of the document has been written. Applications can override this method to add their own definitions to the prologue. For example:

```
- endPrologue
{
    DPSPrintf(DPSGetCurrentContext(), "/littleProc {pop} def");
    return [super endPrologue];
}
```

See also `addToPageSetup`.

endSetup

```
- (void)endSetup
```

Writes out the end of the conforming PostScript setup section, which begins with a %%EndSetup comment. This method is invoked by `print:` and `fax:` just after `beginSetup` is invoked. See also `addToPageSetup`.

endTrailer

```
- (void)endTrailer
```

Writes the end of the conforming PostScript trailer. This method is invoked by `print:` and `fax:` just after `beginTrailer` is invoked.

fax:

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

Prints the `NSView` and all its subviews to a fax modem. Note that faxing is platform specific, therefore this method is not part of the OpenStep specification. See also `print:`, `addToPageSetup`.

frame

```
- (NSRect)frame
```

Returns the NSView's frame rectangle. The frame rectangle is specified in the coordinate system of the NSView's superview. See also frameRotation, rotateByAngle:, setFrame:, setFrameOrigin:, setFrameRotation:, setFrameSize:.

frameRotation

```
- (float)frameRotation
```

Returns the angle of the frame rectangle's rotation, relative to its superview's coordinate system. See also setFrameRotation:.

gState

```
- (int)gState
```

Returns the graphics state object allocated to the NSView. If no graphics state object has been allocated, or if the NSView has not been focused on since receiving the allocateGState message, this method will return 0. Graphics state objects are not immediately allocated by invoking the allocateGState method, but are done in a "lazy" fashion upon subsequent focusing.

heightAdjustLimit

```
- (float)heightAdjustLimit
```

Overide to return the fraction (between 0.0 and 1.0) of the page that can be pushed onto the next page during automatic pagination to prevent items from being cut in half. This limit applies to vertical pagination. This method is invoked by print: and fax:. By default, this method returns 0.2.

hitTest:

```
- (NSView *)hitTest:(NSPoint)aPoint
```

Returns the lowest subview containing the point aPoint. Returns the NSView if it contains the point but none of its subviews do, or nil if the point isn't located within the receiving NSView. This method is used primarily by an

NSWindow to determine which NSView in the view hierarchy should receive a mouse-down event. You'd rarely have reason to invoke this method, but you might want to override it to have an NSView trap mouse-down events before they get to its subviews. aPoint is in the receiving NSView's superview's coordinates.

initWithFrame:

```
- (id)initWithFrame:(NSRect)frameRect
```

Initializes the NSView, which must be a newly allocated NSView instance, to the location and dimensions of frameRect. This method is the designated initializer for the NSView class, and can be used to initialize an NSView allocated from your own zone. Programs generally use instances of NSView subclasses rather than direct instances of the NSView class. Returns self. See also NSRect (Foundation Kit "Types and Constants" chapter).

isDescendantOf:

```
- (BOOL)isDescendantOf:(NSView *)aView
```

Returns YES if aView is an ancestor of the receiving NSView in the view hierarchy or if it's identical to the receiving NSView. Otherwise, this method returns NO. See also addSubview:, ancestorSharedWithView:, superview, subviews.

isFlipped

```
- (BOOL)isFlipped
```

Returns YES if the receiver uses flipped drawing coordinates, or NO if it uses native PostScript coordinates. By default, NSViews are not flipped, and the NSView implementation of this simply returns NO.

isOpaque

```
- (BOOL)isOpaque
```

Returns whether the NSView is opaque. Returns YES if the NSView guarantees that it will completely cover the area within its frame when it draws itself; otherwise returns NO. See also opaqueAncestor, display.

isRotatedFromBase

```
- (BOOL)isRotatedFromBase
```

Returns YES if the receiving NSView or any of its ancestors in the NSView hierarchy have been rotated; otherwise returns NO.

isRotatedOrScaledFromBase

```
- (BOOL)isRotatedOrScaledFromBase
```

Returns YES if the receiving NSView or any of its ancestors in the NSView hierarchy have been rotated or scaled; otherwise returns NO.

knowsPagesFirst:last:

```
- (BOOL)knowsPagesFirst:(int *)firstPageNum last:(int *)lastPageNum
```

Indicates whether this NSView can return a rectangle specifying the region that must be displayed to print a specific page. The default implementation simply returns NO. This method is invoked by print: and fax:. Just before invoking this method, the first page to be printed is set to 1, and the last page to be printed is set to the maximum integer size. You can override this method to change the first page to be printed, and also the last page to be printed if the view knows where its pages lie. If this method is made to return YES, the printing mechanism will later query the NSView for the rectangle corresponding to a specific page using rectForPage:.

locationOfPrintRect:

```
- (NSPoint)locationOfPrintRect:(NSRect)aRect
```

Places the printing rectangle on the physical page. This method is invoked by print: and fax:. aRect is the rectangle being printed on the current page. Returns the location of the lower left corner of the placed rectangle. All coordinates are in the default PostScript coordinate system of the page. By default, if the flags for centering are YES in the global NSPrintInfo object, this routine centers the rectangle within the margins. If the flags are NO, it defaults to abutting the rectangle against the top left margin. See also rectForPage:.

lockFocus

```
- (void)lockFocus
```

Locks the focus on the NSView so that subsequent graphics commands are applied to the NSView. This method ensures that the NSView draws in the correct coordinates and to the correct device. You must send this message to the NSView before you draw to it, and you must balance it with an unlockFocus message to the NSView when you finish drawing.

lockFocus and unlockFocus messages are automatically sent when you use a display method; you don't have to include lockFocus or unlockFocus in your drawRect: method. See also unlockFocus, focusView.

mouse:inRect:

```
- (BOOL)mouse:(NSPoint)aPoint inRect:(NSRect)aRect
```

Returns whether the point aPoint lies inside the aRect. aPoint and aRect must be expressed in the same coordinate system. See also convertPoint:fromView:, hitTest:, acceptsFirstMouse:.

needsDisplay

```
- (BOOL)needsDisplay
```

Returns YES if the NSView needs to be displayed to reflect changes to its contents, otherwise returns NO. If automatic display is disabled, the NSView will not redisplay itself automatically, so you can invoke this method to determine whether you need to send a display message to the NSView. The flag indicating that the NSView needs to be displayed is cleared by the display methods when the NSView is displayed. See also setNeedsDisplay:, display.

opaqueAncestor

```
- (NSView *)opaqueAncestor
```

Returns the receiver's nearest opaque ancestor (including the receiving NSView itself). See also isOpaque.

performKeyEquivalent:

```
- (BOOL)performKeyEquivalent:(NSEvent *)theEvent
```

Implemented by subclasses to allow them to respond to keyboard input. If the NSView responds to the key, it should take the appropriate action and return YES. Otherwise, it should return the result of passing the message along to super, which will pass the message down the NSView hierarchy:

```
return [super performKeyEquivalent:theEvent];
```

The default implementation of this method simply passes the message down the NSView hierarchy and returns NO if none of the NSView's subviews responds to the key. theEvent points to the event record of a key-down event. See also acceptsFirstMouse:.

postsBoundsChangedNotifications

```
- (BOOL)postsBoundsChangedNotifications
```

Returns YES if the view posts bounds changed notifications whenever the view's bounds are translated, scaled, or rotated. Returns NO otherwise. See also setPostsBoundsChangedNotifications:.

postsFrameChangedNotifications

```
- (BOOL)postsFrameChangedNotifications
```

Returns whether notifications of frame changes to ancestors are activated. If YES is returned, the receiving NSView will inform its ancestors in the view hierarchy whenever its frame changes in size or location. If NO is returned, the ancestors are not informed of any frame size or location changes. See also setPostsFrameChangedNotifications:.

print:

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

Prints the NSView and all its subviews. This method brings up a Print panel before printing begins. See also fax:, runOperation (NSPrintOperation).

rectForPage:

```
- (NSRect)rectForPage:(int)page
```

This method should be implemented by subclasses to determine how much of the NSView will be printed for page number page. The default implementation returns an NSRect initialized to zero. You should override this method to return an NSRect with the coordinates of the NSView (in its own coordinate system) that represent the page requested. The NSView will later be told to display that NSRect region in order to generate the image for this page. This method is invoked by print: and fax: if the NSView's knowsPagesFirst:last: method returns YES. The NSView should not assume that the pages will be generated in any particular order.

reflectScrolledClipView:

```
- (void)reflectScrolledClipView:(NSClipView *)aClipView
```

Reflects scrolling within clip view aClipView. See also scrollClipView:toPoint:, adjustScroll:.

registerForDraggedTypes:

```
- (void)registerForDraggedTypes:(NSArray *)newTypes
```

Registers the pasteboard types that the NSView will accept in an image-dragging session. the values in the NSArray are NSPasteboard types, not file extensions (you can't register for specific file extensions). See the NSPasteboard section of the Application Kit's "Types and Constants" chapter for a list of valid pasteboard types.

Note – the values in the first argument are pasteboard types, *not* file extensions (you can't register for specific file extensions). For example, the following registers a view as accepting files.

See also unregisterDraggedTypes.

releaseGState

```
- (void)releaseGState
```

Release the NSView's graphics state object. See also allocateGState.

removeCursorRect:cursor:

```
- (void)removeCursorRect:(NSRect)aRect cursor:(NSCursor *)anObject
```

Removes cursor rectangle `aRect` for cursor `anObject` from the view. `aRect` and `anObject` must match the values that were specified when the cursor rectangle was added (through `addCursorRect:cursor:`). See also `addCursorRect:cursor:`. You rarely need to use this method; it's usually easier to use `NSWindow`'s `invalidateCursorRectsForView:` method and let the `resetCursorRects` mechanism restore the cursor rectangles.

removeFromSuperview

```
- (void)removeFromSuperview
```

Unlinks the `NSView` from its superview and its `NSWindow`, removes it from the responder chain, and invalidates its cursor rectangles. See also `addSubview:`.

removeTrackingRect:

```
- (void)removeTrackingRect:(NSTrackingRectTag)tag
```

Removes the tracking rectangle identified by `tag` from the view. (`tag` is an unique identifier returned from the `addTractingRect:owner:assumeInside:` method). See also `acceptsFirstMouse:`.

renewGState

```
- (void)renewGState
```

Marks the `NSView`'s graphics state object as needing initialization. This method is lazy: the graphics state object isn't refreshed until the `NSView` is drawn. See also `allocateGState`.

replaceSubview:with:

```
- (void)replaceSubview:(NSView *)oldView with:(NSView *)newView
```

Replaces oldView with newView in the NSView's subview list. This method does nothing if oldView is not a subview of the NSView, if newView is not an NSView, or if oldView equals newView. This message should not be sent to an NSClipview object. Use NSClipview's setDocumentView: method instead. See also addSubview:.

resetCursorRects

```
- (void)resetCursorRects
```

This method should be implemented by subclasses to reset their cursor rectangles. Each NSView subclass that wants to include cursor rectangles—areas in which the cursor is changed—must implement this method. The implementation must contain invocations of addCursorRect:cursor:, the method that defines the cursor rectangles and associates them with particular NSCursor objects. The NSView must clip the cursor rectangles that it adds to ensure that they don't overlap the visible rectangle. For example:

```
- resetCursorRects
{
   NSRect visible = [self visibleRect]

   if (visible != NSZeroRect)
          [self addCursorRect:&visible cursor:theCursor];
  }
```

You never need to invoke this method directly; it's invoked automatically when the NSView's NSWindow frame changes, or when the NSWindow receives an invalidateCursorRectsForView: message. Note that this method isn't invoked when the NSView's frame changes unless it changed because its NSWindow was resized. If your application changes an NSView's frame programmatically, through setFrameSize: or setFrameOrigin:, for example, you should follow the frame-changing message with an invalidateCursorRectsForView: message, as shown below:

```
/* Change the NSView's frame. */
[aView setFrame:toNewRect];

/* Tell the NSWindow that the view's cursor rects may have changed.
*/
[[aView window] invalidateCursorRectsForView:aView];
```

```
/* Redisplay the Window. */
[[aView window] display];
```

Invocations of this method aren't cumulative; before a `resetCursorRects` message is sent to a particular `NSView`, the `NSView`'s existing cursor rectangles are automatically discarded. See also `addCursorRect:cursor:`.

resizeSubviewsWithOldSize:

```
- (void)resizeSubviewsWithOldSize:(NSSize)oldSize
```

Initiates `superviewSizeChanged:` messages to subviews. This method is invoked from the `setFrameSize:` method if the `NSView` has subviews and has received a `setAutoresizeSubviews:YES` message. By default, this method sends a `resizeWithOldSuperviewSize:` message to each subview. You should not invoke this method directly, but you may want to override it to define a specific retiling behavior. `oldSize` is the previous bounds rectangle size. See also `autoresizesSubviews`.

resizeWithOldSuperviewSize:

```
- (void)resizeWithOldSuperviewSize:(NSSize)oldSize
```

Informs the `NSView` that its superview's size has changed. This method is invoked when the `NSView`'s superview has received a `resizeSubviewsWithOldSize:` message. This method will automatically resize the `NSView` according to the parameters set by the `setAutosizingMask:` message. You may want to override this method to provide specific resizing behavior. `oldSize` is the previous bounds rectangle size of the receiving `NSView`'s superview.

rotateByAngle:

```
- (void)rotateByAngle:(float)angle
```

Rotates the `NSView`'s frame rectangle by `angle` from its current angle of orientation. Positive values indicate counterclockwise rotation; negative values indicate clockwise rotation. The position of the coordinate origin, (0.0, 0.0), remains unchanged; it's at the center of the rotation. This method posts the `NSViewFocusDidChangeNotification` notification with the receiving object to the default notification center. See also `frameRotation`.

scaleUnitSquareToSize:

```
- (void)scaleUnitSquareToSize:(NSSize)newSize
```

Scales the NSView's coordinate system unit size to newSize. Unit lengths along the x and y axes will be equal to those given in newSize. This method posts the notification NSViewFocusDidChangeNotification with the receiving object to the default notification center. See the "Notifications" section of the Application Kit's "Types and Constants" chapter for more information on notifications. See also boundsRotation.

scrollClipView:toPoint:

```
- (void)scrollClipView:(NSClipView *)aClipView
toPoint:(NSPoint)aPoint
```

Scrolls the clip view aClipView to aPoint. See also scrollPoint:.

scrollPoint:

```
- (void)scrollPoint:(NSPoint)aPoint
```

Aligns aPoint with an NSClipView-derived document view's origin. aPoint is given in the receiving view's coordinates. After scrolling, aPoint will be coincident with the document view's lower left corner, or its upper left corner if the receiving view is flipped. See also adjustScroll:, autoscroll:, reflectScrolledClipView:, scrollClipView:toPoint:, scrollRect:by:, scrollRectToVisible:.

scrollRect:by:

```
- (void)scrollRect:(NSRect)aRect by:(NSSize)delta
```

Shifts the rectangle aRect, which is in the NSView's drawing coordinates, by delta. Only those bits which are visible before and after scrolling are moved. This method works for all NSViews and does not require that the NSView's immediate ancestor be an NSClipView or NSScrollView. See also scrollPoint:.

scrollRectToVisible:

```
- (BOOL)scrollRectToVisible:(NSRect)aRect
```

Scrolls aRect so that it becomes visible within the NSView's parent NSClipView. The receiving NSView must be a NSClipView's content view. This method will scroll the NSClipView the minimum amount necessary to make aRect visible. aRect is a rectangle in the receiving NSView's coordinates. Returns YES if scrolling actually occurs; otherwise returns NO. See also scrollPoint:.

setAutoresizesSubviews:

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

Sets whether to notify subviews of resizing. This method determines whether the resizeSubviewsWithOldSize: message will be sent to the NSView upon receipt of a setFrameSize: message. By default, automatic resizing of subviews is disabled. See also autoresizesSubviews.

setAutoresizingMask:

```
- (void)setAutoresizingMask:(unsigned int)mask
```

Determines how the receiving NSView's frame rectangle will change when its superview's size changes. Create mask by logically ORing the following together:

Table 1-25 Autoresizing Masks

Flag	Meaning
NSViewNotSizeable	NSView does not resize with its superview.
NSViewMinXMargin	Left margin between NSViews can stretch.
NSViewWidthSizable	NSView's width can stretch.
NSViewMaxXMargin	Right margin between NSViews can stretch.
NSViewMinYMargin	Top margin between NSViews can stretch.
NSViewHeightsSizable	NSView's height can stretch.
NSViewMaxYMargin	Bottom margin between NSViews can stretch.

See also autoresizesSubviews.

setBounds:

```
- (void)setBounds:(NSRect)aRect
```

Sets the view's bounds origin and size to `aRect` by sending the `setBoundsOrigin:`, and `setBoundsSize:` messages. See also `bounds`, `boundsRotation`.

setBoundsOrigin:

```
- (void)setBoundsOrigin:(NSPoint)newOrigin
```

Sets the view's drawing origin to `newOrigin`. This method posts the `NSViewFocusDidChangeNotification` notification with the receiving object to the default notification center. See also `setBounds:`, `boundsRotation`.

setBoundsRotation:

```
- (void)setBoundsRotation:(float)angle
```

Rotates the `NSView`'s coordinate system to *angle*. This method posts the `NSViewFocusDidChangeNotification` notification with the receiving object to the default notification center. See also `setBounds:`, `boundsRotation`.

setBoundsSize:

```
- (void)setBoundsSize:(NSSize)newSize
```

Resizes the `NSView`'s coordinate system to `newSize`. This method posts the `NSViewFocusDidChangeNotification` notification with the receiving object to the default notification center. See the "Notifcations" section of the Application Kit's "Types and Constants" chapter for more information on notifications. See also `boundsRotation`.

setFrame:

```
- (void)setFrame:(NSRect)frameRect
```

Assigns the view a new frame rectangle (size and origin) by sending the view `setFrameSize:` and `setFrameOrigin:` messages. See also `frame`.

setFrameOrigin:

```
- (void)setFrameOrigin:(NSPoint)newOrigin
```

Sets the origin of the view's frame to `newOrigin`. This method posts the `NSViewFrameDidChangeNotification` and `NSViewFocusDidChangeNotification` notifications with the receiving object to the default notification center. See the "Notifications" section of the Application Kit's "Types and Constants" chapter for more information on notifications. See also `setFrame:`, `frame`.

setFrameRotation:

```
- (void)setFrameRotation:(float)angle
```

Rotates the view's frame to `angle` . This method posts the `NSViewFocusDidChangeNotification` notification with the receiving object to the default notification center. See the "Notifications" section of the Application Kit's "Types and Constants" chapter for more information on notifications. See also `frame`.

setFrameSize:

```
- (void)setFrameSize:(NSSize)newSize
```

Resizes the view's frame to `newSize`, in its superview's coordinates. This method posts the `NSViewFrameDidChangeNotification` and `NSViewFocusDidChangeNotification` notifications with the receiving object to the default notification center. See also `setFrame:`, `frame`.

setNeedsDisplay:

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

This method sets a flag indicating whether the `NSView` needs to be displayed. If `flag` is `YES`, the view is marked as changed and requiring redisplay. This method sends the `setNeedsDisplayInRect:` message with the current bounding rectangle for the view to do the work, thereby marking the whole view as needing redisplay. See also `display`.

setNeedsDisplayInRect:

```
- (void)setNeedsDisplayInRect:(NSRect)invalidRect
```

Marks the NSView as changed and requiring redisplay within rectangle invalidRect. This rectangle is added to a list of any other "dirty" rectangles within the view needing updating. See also setNeedsDisplay:, display.

setPostsBoundsChangedNotifications:

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

Sets whether the view posts bounds changed notifications whenever the view's bounds are translated, scaled, or rotated. See also postsBoundsChangedNotifications.

setPostsFrameChangedNotifications:

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

Sets whether to activate ancestor notifications. If flag is YES, the receiving NSView will inform its ancestors in the view hierarchy whenever its frame changes in size or location. If flag is NO, the ancestors are not informed of any frame size or location changes. See also postsFrameChangedNotifications.

setUpGState

```
- (void)setUpGState
```

Sets up the NSView's graphics state object. See also allocateGState, gState.

shouldDelayWindowOrderingForEvent:

```
- (BOOL)shouldDelayWindowOrderingForEvent:(NSEvent *)anEvent
```

Returns YES if the normal NSWindow ordering and activation mechanism should be delayed until the next mouse-up event. You never inovke this method directly; it's invoked automatically for each mouse-down that's directed at the NSView. The default implementation returns NO.

An NSView subclass that contains draggable images should implement this to return YES (perhaps predicating the decision on the data in anEvent, the event record for the mouse-down itself). This allows the user to click on a draggable image without bringing the NSView's NSWindow to the front or making its application active. Note that this method doesn't prevent this ordering and activation from occurring, it simply puts it off until the user releases the mouse. To cause the ordering and activation to be skipped when the mouse is released, the NSView should send a preventWindowOrdering message to the NSApplication object from within its implementation of mouseDown:. The preventWindowOrdering message is sent automatically by NSView's dragImage:... method—in other words, ordering and activation is prevented if the user actually drags the clicked-on item.

shouldDrawColor

```
- (BOOL)shouldDrawColor
```

Returns whether the view should be drawn in color. If the NSView is being drawn to a window that can render color, then YES is returned; otherwise NO is returned. See also display.

sortSubviewsUsingFunction:context:

```
- (void)sortSubviewsUsingFunction:(int (*)(id ,id ,void *))compare
    context:(void *)context
```

Sorts the receiving view's subviews using the sorting function compare and the context context. The first two arguments of the function are the views to be compared. See also addSubview:.

subviews

```
- (NSArray *)subviews
```

Returns a mutable array containing the receiving NSView's subviews. You can use this array to send messages to each NSView in the NSView hierarchy. You never modify this array directly; use addSubview: and removeFromSuperview to add and remove NSViews from the NSView hierarchy. If the NSView has no subviews an empty array is returned. See also addSubview:, superview.

superview

```
- (NSView *)superview
```

Returns the NSView's superview. If the NSView hasn't a superview, nil is returned. When applying this method recursively, you should check the return value against the content NSView of the NSView's NSWindow to avoid flying off the top of the NSView hierarchy. See also subviews.

tag

```
- (int)tag
```

Returns the NSView's tag, which is an integer that you can use to identify objects in your application. By default, NSView returns -1. You can override this method to identify certain NSViews. For example, your application could take special action when an NSView with a given tag receives a mouse event. See also viewWithTag:.

translateOriginToPoint:

```
- (void)translateOriginToPoint:(NSPoint)point
```

Shifts the NSView's coordinate system to point. This method posts the NSViewFocusDidChangeNotification notification with the receiving object to the default notification center. See the Notifications section of the Application Kit's Types and Constants chapter for information on notifications. See also boundsRotation.

unlockFocus

```
- (void)unlockFocus
```

Unfocuses the receiving view. Balances an earlier lockFocus message to the same NSView. If the lockFocus method saved the previous graphics state, this method restores it.

unregisterDraggedTypes

```
- (void)unregisterDraggedTypes
```

Unregisters the window as a recipient of dragged images. See also `registerForDraggedTypes:`, `dragFile:fromRect:slideBack:event:`.

viewWillMoveToSuperview:

```
- (void)viewWillMoveToSuperview:(NSView *)newSuperview
```

Changes the receiving view's superview to `newSuperview`. See also `viewWillMoveToWindow:`.

viewWillMoveToWindow:

```
- (void)viewWillMoveToWindow:(NSWindow *)newWindow
```

Notifies the view that it will move to a new window. See also `window`, `viewWillMoveToSuperview:`.

viewWithTag:

```
- (id)viewWithTag:(int)aTag
```

Returns the subview (including `self`) with `aTag` as its tag, or `nil` if no view is found with that tag. See also `tag`.

visibleRect

```
- (NSRect)visibleRect
```

Returns the visible portion of the `NSView`. If no portion of the `NSView` is visible, an empty rectangle (`NSZeroRect`) is returned. Visibility is determined by intersecting the `NSView`'s frame rectangle against the frame rectangles of each of its ancestors in the view hierarchy, after appropriate coordinate transformations. Only those portions of the `NSView` that lie within the frame rectangles of all its ancestors can be visible. This method does not take into account any siblings of the receiving view or siblings of its ancestors. If the `NSView` is being printed, this method returns the portion of the `NSView` that is visible on the page being imaged. See also `display`.

widthAdjustLimit

```
- (float)widthAdjustLimit
```

Returns the fraction (between 0.0 and 1.0) of the page that can be pushed onto the next page during automatic pagination to prevent items from being cut in half. This limit applies to horizontal pagination. This method is invoked by `print:` and `fax:`. By default, this method returns 0.2. See also `heightAdjustLimit`, `adjustPageHeightNew:top:bottom:limit:`, `adjustPageWidthNew:left:right:limit:`.

window

```
- (NSWindow *)window
```

Returns the `NSWindow` in which the view is displayed. See also `addSubview:`.

writeEPSInsideRect:toPasteboard:

```
- (void)writeEPSInsideRect:(NSRect)rect
    toPasteboard:(NSPasteboard *)pasteboard
```

Places PostScript code for the rectangle `rect` on the `pasteboard`. See also `dataWithEPSInsideRect:`.

NSWindow

Inherits From:	NSResponder : NSObject
Conforms To:	NSCoding (NSResponder) NSObject (NSObject)
Declared In:	AppKit / NSWindow.h

Class Description

The `NSWindow` class defines objects that manage and coordinate the windows that an application displays on the screen. A single `NSWindow` object corresponds to, at most, one window. The two principle functions of an `NSWindow` are to provide an area in which views can be placed, and to accept and distribute, to the appropriate `NSView`s, events that the user instigates by manipulating the mouse and keyboard.

Rectangles, Views, and the View Hierarchy

An NSWindow is defined by a *frame rectangle* that encloses the entire window, including its title bar, resize bar, and border, and by a *content rectangle* that encloses just its content area. Both rectangles are specified in the screen coordinate system. The frame rectangle establishes the NSWindow's *base coordinate system*. This coordinate system is always aligned with and is measured in the same increments as the screen coordinate system (in other words, the base coordinate system can't be rotated or scaled). The origin of a base coordinate system is the bottom left corner of the window's frame rectangle.

You create an NSWindow through one of the `init:...` methods by specifying, among other attributes, the size and location of its content rectangle. The frame rectangle is derived from the dimensions of the content rectangle.

When it's created, an NSWindow automatically creates two NSViews: an opaque *frame view* and a transparent *content view* that fills the content area. The frame view is a private object that your application can't access directly. The content view is the "highest" accessible view in the window; you can replace the content view with an NSView of your own creation through NSWindow's `setContentView:` method.

You add other views to the window by declaring each to be a subview of the content view, or a subview of one of the content view's subviews, and so on, through NSView's `addSubview:` method. This tree of views is called the window's *view hierarchy*. When an NSWindow is told to display itself, it does so by sending view-displaying messages to each object in its view hierarchy. Because displaying is carried out in a determined order, the content view (which is drawn first) may be wholly or partially obscured by its subviews, and these subviews may be obscured by their subviews (and so on).

Event Handling

The window system and the NSApplication object forward mouse and keyboard events to the appropriate NSWindow object. The NSWindow that's currently designated to receive keyboard events is known as the *key window*. If the mouse or keyboard event affects the window directly—resizing or moving it, for example—the NSWindow performs the appropriate operation itself and

sends messages to its delegate informing it of its intentions, thus allowing your application to intercede. Events that are directed at specific views within the window are forwarded by the NSWindow to the NSView.

The NSWindow keeps track of the object that was last selected to handle keyboard events as its *first responder*. The first responder is typically the NSView that displays the current selection. In addition to keyboard events, the first responder is sent action messages that have a user-selected target (a nil target in program code). The NSWindow continually updates the first responder in response to the user's mouse actions.

Each NSWindow provides a *field editor*, an NSText object that handles small-scale text-editing chores. The field editor can be used by the NSWindow's first responder to edit the text that it displays. The fieldEditor:forObject: method returns the NSWindow's field editor. (You can make this method instead return an alternative NSText object, appropriate for the object specified the second argument, by implementing the delegate method windowWillReturnFieldEditor:toObject:.)

Method Types

Activity	Class Method
Initializing and getting a new NSWindow object	– initWithContentRect:styleMask:backing:defer: – initWithContentRect:styleMask:backing:defer: screen:
Computing frame and content rectangles	+ contentRectForFrameRect:styleMask: + frameRectForContentRect:styleMask: + minFrameWidthWithTitle:styleMask:
Accessing the content view	– contentView – setContentView:
Window graphics	– backgroundColor – representedFilename – setBackgroundColor: – setRepresentedFilename: – setTitle: – setTitleWithRepresentedFilename: – styleMask – title

Activity	Class Method
Window device attributes	– backingType – deviceDescription – gState – isOneShot – setBackingType: – setOneShot: – windowNumber
The miniwindow	– miniwindowImage – miniwindowTitle – setMiniwindowImage: – setMiniwindowTitle:
The field editor	– endEditingFor: – fieldEditor:forObject:
Window status and ordering	– becomeKeyWindow – becomeMainWindow – canBecomeKeyWindow – canBecomeMainWindow – hidesOnDeactivate – isKeyWindow – isMainWindow – isMiniaturized – isVisible – level – makeKeyAndOrderFront: – makeKeyWindow – makeMainWindow – orderBack: – orderFront: – orderFrontRegardless – orderOut: – orderWindow:relativeTo: – resignKeyWindow – resignMainWindow – setHidesOnDeactivate: – setLevel:

Activity	Class Method
Moving and resizing the window	– cascadeTopLeftFromPoint: – center – constrainFrameRect:toScreen: – frame – minSize – maxSize – setContentSize: – setFrame:display: – setFrameOrigin: – setFrameTopLeftPoint: – setMinSize: – setMaxSize:
Converting coordinates	– convertBaseToScreen: – convertScreenToBase:
Managing the display	– display – disableFlushWindow – displayIfNeeded – enableFlushWindow – flushWindow – flushWindowIfNeeded – isAutodisplay – isFlushWindowDisabled – setAutodisplay: – setViewsNeedDisplay: – update – useOptimizedDrawing: – viewsNeedDisplay
Screen and window depths	+ defaultDepthLimit – canStoreColor – deepestScreen – depthLimit – hasDynamicDepthLimit – screen – setDepthLimit: – setDynamicDepthLimit:
Cusor management	– areCursorRectsEnabled – disableCursorRects – discardCursorRects – enableCursorRects – invalidateCursorRectsForView: – resetCursorRects

Activity	Class Method
Handling user actions and events	– close – deminiaturize: – isDocumentEdited – isReleasedWhenClosed – miniaturize: – performClose: – performMiniaturize: – resizeFlags – setDocumentEdited: – setReleasedWhenClosed:
Aiding event handling	– acceptsMouseMovedEvents – currentEvent – discardEventsMatchingMask:beforeEvent: – firstResponder – keyDown: – makeFirstResponder: – mouseLocationOutsideOfEventStream – nextEventMatchingMask: – nextEventMatchingMask:untilDate:inMode: dequeue: – postEvent:atStart: – setAcceptsMouseMovedEvents: – sendEvent: – tryToPerform:with: – worksWhenModal
Dragging	– dragImage:at:offset:event:pasteboard:source: slideBack: – registerForDraggedTypes: – unregisterDraggedTypes
Services and windows menu support	– isExcludedFromWindowsMenu – setExcludedFromWindowsMenu: – validRequestorForSendType:returnType:
Saving and restoring the frame	+ removeFrameUsingName: – frameAutosaveName – saveFrameUsingName: – setFrameAutosaveName: – setFrameFromString: – setFrameUsingName: – stringWithSavedFrame
Printing and PostScript	– dataWithEPSInsideRect: – fax: – print:

Activity	Class Method
Window image caching and restoring	– cacheImageInRect: – discardCachedImage – restoreCachedImage
Assigning a delegate	– delegate – setDelegate:
Methods Implemented by the Delegate	– windowDidBecomeKey: – windowDidBecomeMain: – windowDidChangeScreen: – windowDidDeminiaturize: – windowDidExpose: – windowDidMiniaturize: – windowDidMove: – windowDidResignKey: – windowDidResignMain: – windowDidResize: – windowDidUpdate: – windowShouldClose: – windowWillClose: – windowWillReturnFieldEditor:toObject:

Class Methods

contentRectForFrameRect:styleMask:

```
+ (NSRect)contentRectForFrameRect:(NSRect)aRect
    styleMask:(unsigned int)aStyle
```

Calculates and returns the content rectangle for an NSWindow with frame rectangle aRect and window type style aStyle. Both are in screen coordinates. The acceptable style masks are

- NSBorderlessWindowMask
- NSTitledWindowMask
- NSClosableWindowMask
- NSMiniturizableWindowMask
- NSResizableWindowMask

See also frameRectForContentRect:styleMask:, and the NSWindow enums in the Application Kit "Types and Constants" section.

defaultDepthLimit

```
+ (NSWindowDepth)defaultDepthLimit
```

Returns the default depth limit for all windows. Do not send this message before the NSApplication object is running. Do not send this messge until your application object is created, and a DPS context is created. See also depthLimit, setDepthLimit:, hasDynamicDepthLimit, setDynamicDepthLimit:.

frameRectForContentRect:styleMask:

```
+ (NSRect)frameRectForContentRect:(NSRect)aRect
    styleMask:(unsigned int)aStyle
```

Calculates and returns the frame rectangle for an NSWindow with the given content rectangle (aRect) and style (aStyle). Both are in screen coordinates. See the style method for a list of acceptable style values. The acceptable style masks are

- NSBorderlessWindowMask
- NSTitledWindowMask
- NSClosableWindowMask

- NSMiniturizableWindowMask
- NSResizableWindowMask

See also contentRectForFrameRect:styleMask:.

minFrameWidthWithTitle:styleMask:

```
+ (float)minFrameWidthWithTitle:(NSString *)aTitle
    styleMask:(unsigned int)aStyle
```

Returns the minimum frame width that an NSWindow's frame rectangle must have for it to display all of aTitle, given the specified style. See the style method for a list of acceptable style mask values.

removeFrameUsingName:

```
+ (void)removeFrameUsingName:(NSString *)name
```

Removes frame rectangle name from the system defaults. See also saveFrameUsingName:, setFrameFromString:, setFrameUsingName:, stringWithSavedFrame, frameAutosaveName, setFrameAutosaveName:.

Instance Methods

acceptsMouseMovedEvents

```
- (BOOL)acceptsMouseMovedEvents
```

Returns YES if the NSWindow accepts mouse-moved events, and NO otherwise. See also setAcceptsMouseMovedEvents:.

areCursorRectsEnabled

```
- (BOOL)areCursorRectsEnabled
```

Returns YES if cursor rectangles are enabled, NO otherwise. See also disableCursorRects, enableCursorRects, discardCursorRects, invalidateCursorRectsForView:, resetCursorRects.

backgroundColor

```
- (NSColor *)backgroundColor
```

Returns the NSwindow's background color. See also NSColor, setBackgroundColor:.

backingType

```
- (NSBackingStoreType)backingType
```

Returns the window device's backing store type, which is one of the following values:

- NSBackingStoreRetained
- NSBackingStoreNonretained
- NSBackingStoreBuffered

See also NSBackingStoreType (Display Postscript "Types and Constants" chapter).

becomeKeyWindow

```
- (void)becomeKeyWindow
```

Records the window's new status as the key window. This method posts the notification NSWindowDidBecomeKeyNotification with the receiving object to the default notification center, and sends a becomeKeyWindow message to this window's first responder (unless this window is the first responder, or none exists). It is not necessary to invoke this method; it is invoked automatically when the NSWindow becomes the key window. See also canBecomeKeyWindow, isKeyWindow, makeKeyWindow, makeKeyAndOrderFront:, resignKeyWindow.

becomeMainWindow

```
- (void)becomeMainWindow
```

Records the window's new status as the main window. This method posts the notification NSWindowDidBecomeMainNotification with the receiving object to the default notification center. See also canBecomeMainWindow, isMainWindow, makeMainWindow, resignMainWindow.

cacheImageInRect:

```
- (void)cacheImageInRect:(NSRect)aRect
```

This method saves a window's current "bits". The sender can then draw into the window (animation, etc.), after which they can quickly clear their drawing by restoring the area(s) they have drawn over. This used to be accomplished with instance drawing.See also `restoreCachedImage`, `discardCachedImage`.

canBecomeKeyWindow

```
– (BOOL)canBecomeKeyWindow
```

Returns `YES` if the receiving `NSWindow` object can become the key window, `NO` otherwise. See also `becomeKeyWindow`.

canBecomeMainWindow

```
– (BOOL)canBecomeMainWindow
```

Returns `YES` if the receiving `NSWindow` object can become the main window, `NO` otherwise. See also `becomeMainWindow`.

canStoreColor

```
– (BOOL)canStoreColor
```

Returns `YES` if the `NSWindow` has a depth limit large enough to store color values, and `NO` otherwise. See also `depthLimit`.

cascadeTopLeftFromPoint:

```
– (NSPoint)cascadeTopLeftFromPoint:(NSPoint)topLeftPoint
```

When successively invoked, tiles windows by offsetting them slightly to the right and down from the previous window. Returns the top left point of the placed window, which is typically used for `topLeftPoint` in the next invocation. If you specify (0,0) for the top left point, this method places the window as is, and returns its top left point.

center

```
- (void)center
```

Moves the NSWindow to the center of the screen: dead-center horizontally and slightly above center vertically. Use this method to place an NSWindow—most likely an attention panel—where the user can't miss it. This method is invoked automatically when a panel is placed on the screen by NSApplication's beginModalSessionForWindow: method.

close

```
- (void)close
```

Closes the window. When this method begins, it posts the notification NSWindowWillCloseNotification with the receiving object to the default notification center. See also isReleasedWhenClosed, setReleasedWhenClosed:, performClose:.

constrainFrameRect:toScreen:

```
- (NSRect)constrainFrameRect:(NSRect)frameRect
    toScreen:(NSScreen *)screen
```

Constrains the window's frame rectangle frameRect to fit on screen. Returns the constrained frame rectangle. theFrame is modified so that its top edge lies on the given screen. If the NSWindow is resizable, the rectangle's height is adjusted to bring the bottom edge onto the screen as well. The rectangle's width and horizontal location are unaffected. You shouldn't need to invoke this method yourself; it is invoked automatically (and the modified frame is used to locate and set the size of the NSWindow) whenever a titled NSWindow is placed on-screen or resized. You can override this method to prevent a particular NSWindow from being constrained, or to constrain it differently. The unconstrained frame rectangle is pointed to by theFrame; the screen that it will be displayed on is pointed to by screen.

contentView

```
- (id)contentView
```

Returns the NSWindow's content view: the highest accessible NSView object in the NSWindow's view heirarchy. See also setContentView:.

convertBaseToScreen:

```
- (NSPoint)convertBaseToScreen:(NSPoint)aPoint
```

Converts `aPoint` from base to screen coordinates. See also `convertScreenToBase:`.

convertScreenToBase:

```
- (NSPoint)convertScreenToBase:(NSPoint)aPoint
```

Converts `aPoint` from screen to base coordinates. See also `convertBaseToScreen:`.

currentEvent

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

Returns the last event object retrieved from the even queue by the `NSApplication`. See also `NSApplication`, `NSEvent`.

dataWithEPSInsideRect:

```
- (NSData *)dataWithEPSInsideRect:(NSRect)rect
```

Returns the encapsulated PostScript inside `rect` as a data object. See also `EPSOperationWithView:insideRect:toData:` (`NSPrintOperation`), `NSData` (Foundation Kit “Types and Constants” chapter).

deepestScreen

```
- (NSScreen *)deepestScreen
```

Returns the deepest screen that the `NSWindow` is on, or `nil` if the `NSWindow` is off the screen. See also `screen`, `depthLimit`.

delegate

```
- (id)delegate
```

Returns the `NSWindow`’s delegate, or `nil` if none exists.

deminiaturize:

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

Hides the miniwindow and redisplays the NSWindow. You rarely need to invoke this method; it's invoked automatically when an NSWindow is deminiaturized by the user (by double-clicking a miniwindow, or by choosing the Arrange in Front item in the NSWindow's menu). However, if you feel compelled to deminiaturize an NSWindow programmatically, you should note that the deminiaturize message is sent to the miniwindow, *not* the original NSWindow, and the value passed as sender is ignored.

depthLimit

```
- (NSWindowDepth)depthLimit
```

Returns the window's depth limit, which can be one of the following values:

- NSDefaultDepth
- NSTwoBitGrayDepth
- NSEightBitGrayDepth
- NSTwelveBitRGBDepth
- NSTwentyFourBitRGBDepth

If the return value is NSDefaultDepth, you can find the actual depth limit by sending the window class a defaultDepthLimit message. See also defaultDepthLimit, deepestScreen, hasDynamicDepthLimit, setDepthLimit:.

deviceDescription

```
- (NSDictionary *)deviceDescription
```

Returns the window device's attributes as key/value pairs. See also NSDictionary.

disableCursorRects

```
- (void)disableCursorRects
```

Disables all cursor rectangles within the NSWindow. Typically this method is used when you need to do some special cursor manipulation, and you don't want the Application Kit interfering. See also areCursorRectsEnabled.

disableFlushWindow

```
- (void)disableFlushWindow
```

Disables the `flushWindow` method for the `NSWindow`. If the `NSWindow` is a buffered window, drawing won't automatically be flushed to the screen by the display methods defined in the `NSView` class. This permits several `NSView`s to be displayed before the results are shown to the user. Flushing should be disabled only temporarily, while the `NSWindow`'s display is being updated. Each `disableFlushWindow` message should be paired with a subsequent `enableFlushWindow` message. Message pairs can be nested; flushing won't be reenabled until the last (unnested) `enableFlushWindow` message is sent. See also `flushWindow`.

discardCachedImage

```
- (void)discardCachedImage
```

This method releases the memory used to store a saved bit image. This method should be called after the final `restoreCachedImage` is called. See also `cacheImageInRect:`, `restoreCachedImage`.

discardCursorRects

```
- (void)discardCursorRects
```

Removes all cursor rectangles in the `NSWindow`. This method is invoked by `resetCursorRects` to remove existing cursor rectangles before resetting them. In general, you wouldn't invoke it in the code you write, but might want to override it to change its behavior. See also `areCursorRectsEnabled`.

discardEventsMatchingMask:beforeEvent:

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

Invokes the `NSApplication` method of the same name. Removes all events from the event queue matching `mask` that were generated before `lastEvent`. If `lastEvent` is `nil`, all events matching `mask` are removed from the queue.

display

```
- (void)display
```

Displays all the NSWindow's views, including the border, resize bar, and title bar. If displaying is disabled for the NSWindow, display enables it. See also displayIfNeeded, isAutodisplay, setAutodisplay:, setViewsNeedDisplay:, update.

displayIfNeeded

```
- (void)displayIfNeeded
```

Displays all the NSWindow's views that need to be redrawn. This method is useful when you want to disable displaying in the NSWindow, modify some number of NSViews, and then display only the ones that were modified Note that this method, unlike display, doesn't reenable display if it's currently disabled. See also display.

dragImage:at:offset:event:pasteboard:source:slideBack:

```
- (void)dragImage:(NSImage *)anImage at:(NSPoint)baseLocation
    offset:(NSSize)initialOffset event:(NSEvent *)event
    pasteboard:(NSPasteboard *)pboard source:(id)sourceObject
    slideBack:(BOOL)slideFlag
```

Instigates an image-dragging session. You never invoke this method directly from your application; it can only be invoked from within an NSView's implementation of the mouseDown: method. Furthermore, NSView also implements the dragImage:... method; you typically instigate an image-dragging session by sending this message to an NSView, rather than an NSWindow. The two methods are identical except for the interpretation of the baseLocation argument: In NSWindow's implementation, baseLocation is taken in the base coordinate system. See the description of this method in the NSView class for the meanings of the other arguments. See also dragImage:at:offset:event:pasteboard:source: slideBack: (NSView).

enableCursorRects

```
- (void)enableCursorRects
```

Enables cursor rectangles within the NSWindow. See also areCursorRectsEnabled.

enableFlushWindow

```
- (void)enableFlushWindow
```

Enables flushing for a buffered window. See also disableFlushWindow, flushWindow, flushWindowIfNeeded, isFlushWindowDisabled.

endEditingFor:

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

Ends the field editor's editing assignment for anObject. If the field editor is the first responder, it resigns that status, passing it to the NSWindow (even if the field editor refuses to resign). This forces a textDidEndEditing: message to be sent to the field editor's delegate. The field editor is then removed from the view hierarchy and its delegate is set to nil. See also fieldEditor:forObject:.

fax:

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

Prints the NSWindow (all the NSViews in its view hierarchy including the frame view) to a fax modem. This method provides users with an independent control for faxing an NSWindow. This method brings up a Fax panel before printing begins. Note that faxing is platform specific, therefore this method is not part of the OpenStep specification. See also print:, NSPrintOperation.

fieldEditor:forObject:

```
- (NSText *)fieldEditor:(BOOL)createFlag forObject:(id)anObject
```

Returns the NSWindow object's field editor for anObject. If the field editor does not exist and createFlag is YES, a field editor is created. The field editor is provided as a convenience and can be used however your application sees fit. Typically, the field editor is used by simple text-bearing objects—for example, a NSTextField object uses its NSWindow's field editor to display and manipulate text. The field editor can be shared by any number of objects

and so its state may be constantly changing. Therefore, it shouldn't be used to display text that demands sophisticated NSText object preparation. For this you should create a dedicated NSText object).

A newly created NSWindow doesn't have a field editor; the only way to create a field editor is to invoke this method with a flag value of YES. After a field editor has been created for an NSWindow, the flag argument is ignored.

The NSWindow's delegate can supply the object that this method returns as the return value of the windowWillReturnFieldEditor:toObject: delegate message (the NSWindow is passed as the first argument, anObject is passed as the second). However, note the following:

- If the NSWindow's delegate is anObject, the windowWillReturnFieldEditor:toObject: message isn't sent.
- The object returned by the delegate method doesn't become the NSWindow's field editor.

If this method returns a non-nil value, it should be followed by an invocation of NSWindow's endEditingFor: method before the field editor is actually used.

firstResponder

```
- (NSResponder *)firstResponder
```

Returns the first responder to user events. See also makeFirstResponder:,

flushWindow

```
- (void)flushWindow
```

If the NSWindow is buffered and flushing hasn't been disabled by disableFlushWindow, this flushes the off-screen buffer to the screen. This method is automatically invoked when you send a display message to an NSWindow or NSView. However, it has no effect if the display is being directed to a printer or other device, rather than to the screen. See also display, disableFlushWindow, enableFlushWindow, flushWindowIfNeeded, isFlushWindowDisabled.

flushWindowIfNeeded

```
- (void)flushWindowIfNeeded
```

Flushes the Window's off-screen buffer to the screen, provided that:

- The NSWindow is a buffered window.
- Flushing isn't currently disabled.
- Some previous flushWindow messages had no effect because flushing was disabled.

You should use this method, rather than flushWindow, to flush an NSWindow after flushing has been reenabled. See also flushWindow.

frame

```
- (NSRect)frame
```

Returns the window's frame rectangle. See also setFrame:display:, setFrameOrigin:, setFrameTopLeftPoint:, minSize, maxSize, setContentSize:, setMinSize:, setMaxSize:.

frameAutosaveName

```
- (NSString *)frameAutosaveName
```

Returns the name that's used to automatically save the NSWindow's frame rectangle data in the system defaults, as set through setFrameAutosaveName:. If the NSWindow has an autosave name, it's frame data is written as a default whenever the frame rectangle changes. See also removeFrameUsingName:, saveFrameUsingName:, setFrameAutosaveName:, setFrameFromString:, setFrameUsingName:, stringWithSavedFrame.

gState

```
- (int)gState
```

Returns the PostScript graphics-state object for the NSWindow.

hasDynamicDepthLimit

```
- (BOOL)hasDynamicDepthLimit
```

Returns YES if the NSWindow's depth limit can change to match the depth of the screen it is displayed on, and NO if it can't. See also setDynamicDepthLimit:, depthLimit.

hidesOnDeactivate

```
- (BOOL)hidesOnDeactivate
```

Returns YES if the NSWindow will be removed from the screen when its application is deactivated, and NO if it will remain on-screen. See also setHidesOnDeactivate:.

initWithContentRect:styleMask:backing:defer:

```
- (id)initWithContentRect:(NSRect)contentRect
    styleMask:(unsigned int)aStyle
    backing:(NSBackingStoreType)bufferingType defer:(BOOL)flag
```

Initializes a new NSWindow. contentRect specifies the location and size of the NSWindow's content area in screen coordinates. If a NULL pointer is passed for this argument, a default rectangle is used. aStyle is a bitmap mask, and specifies the NSWindow's style. Styles are

- NSBorderlessWindowMask
- NSTitledWindowMask
- NClosableWindowMask
- NSMiniturizableWindowMask
- NSResizableWindowMask

Titled and resizable NSWindows are by far the most common. The third argument, bufferingType, specifies how the drawing done in the NSWindow is buffered by the object's window device:

- NSBackingStoreRetained
- NSBackingStoreNonretained
- NSBackingStoreBuffered

Lastly, if flag is YES, window creation is deferred until the NSWindow is needed on-screen. All display messages sent to the NSWindow or its NSViews will be postponed until the window is created, just before it's moved on-screen. Deferring the creation of the window improves launch time and minimizes the virtual memory load on the Window Server.

The NSWindow creates an instance of NSView to be its default content view. You can replace it with your own object by using the setContentView: method. This method returns self. See also initWithContentRect:styleMask:backing:defer: screen:, NSBackingStoreType (Display Postscript "Types and Constants" chapter).

initWithContentRect:styleMask:backing:defer: screen:

```
- (id)initWithContentRect:(NSRect)contentRect
    styleMask:(unsigned int)aStyle
    backing:(NSBackingStoreType)bufferingType defer:(BOOL)flag
    screen:(NSScreen *)aScreen
```

Initializes a new NSWindow with a content rectangle location and size specified by contentRect, a window style and buttons as indicated in the bitmap mask aStyle, drawing buffering specified by bufferingType, and for the screen specified by aScreen. This method is equivalent to initContent:style:backing:buttonMask:defer:, except that the content rectangle is specified relative to the lower left corner of aScreen.

If aScreen is NULL, the content rectangle is interpreted relative to the lower left corner of the main screen. The main screen is the one that contains the current key window, or, if there is no key window, the one that contains the main menu. If there's neither a key window nor a main menu (if there's no active application), the main screen is the one where the origin of the screen coordinate system is located. If flag is YES, the window system defers creating the window until it's needed. See initWithContentRect:styleMask:backing:defer: for a further explanation of the arguments. See also NSScreen.

invalidateCursorRectsForView:

```
- (void)invalidateCursorRectsForView:(NSView *)aView
```

Marks cursor rectangles invalid for aView. See also areCursorRectsEnabled.

isAutodisplay

- (BOOL)isAutodisplay

Returns whether the window displays all views requiring redrawing when update is invoked. See also setAutodisplay:, update.

isDocumentEdited

- (BOOL)isDocumentEdited

Returns YES if the NSWindow's document has been edited, otherwise returns NO. See also setDocumentEdited:.

isExcludedFromWindowsMenu

- (BOOL)isExcludedFromWindowsMenu

Returns YES if the NSWindow is excluded from the application's Windows menu, and NO if it isn't. See also setExcludedFromWindowsMenu:.

isFlushWindowDisabled

- (BOOL)isFlushWindowDisabled

Returns YES if the NSWindow's flushing ability is disabled, otherwise returns NO. See also flushWindow.

isKeyWindow

- (BOOL)isKeyWindow

Returns YES if the NSWindow is the application's key window, otherwise returns NO. See also becomeKeyWindow, canBecomeKeyWindow, isMainWindow.

isMainWindow

- (BOOL)isMainWindow

Returns YES if the NSWindow is the main window for the application, and NO if it isn't. See also becomeMainWindow, canBecomeMainWindow, isKeyWindow.

isMiniaturized

```
- (BOOL)isMiniaturized
```

Returns YES if the NSWindow is hidden and the miniwindow displayed, and NO otherwise. See also isVisible.

isOneShot

```
- (BOOL)isOneShot
```

Returns YES if the backing-store memory for the NSWindow is freed when the NSWindow is ordered off-screen. See also setOneShot:.

isReleasedWhenClosed

```
- (BOOL)isReleasedWhenClosed
```

Returns YES if the NSWindow is released when it is closed, otherwise returns NO. See also setReleasedWhenClosed:, performClose:.

isVisible

```
- (BOOL)isVisible
```

Returns YES if the NSWindow is on-screen (even if it's obscured by other NSWindows), otherwise returns NO.

keyDown:

```
- (void)keyDown:(NSEvent *)theEvent
```

Responds to the key-down event passed as theEvent. NSWindow's version of keyDown: first checks to see if the message has been sent to an NSMenu object that is not visible, in which case an update message is sent to the NSMenu. Next, if theEvent is an NSKeyDown event sent along with some characters, then theEvent is sent to the content view object. The content view object

passes it along to any subviews until it reaches the first subview that accepts it. If no view responds, the default NSResponder keyDown: method is invoked. See also update, keyDown: (NSResponder.

level

```
- (int)level
```

Returns the current window level. The following values represent the NSWindow levels:

- NSNormalWindowLevel
- NSFloatingWindowLevel
- NSDockWindowLevel
- NSSubmenuWindowLevel
- NSMainMenuWindowLevel

For more information on window levels, see the "NSWindow" section of the Application Kit's "Types and Constants" chapter. See also setLevel:.

makeFirstResponder:

```
- (BOOL)makeFirstResponder:(NSResponder *)aResponder
```

Makes aResponder the first receiver of keyboard events and action messages sent to the NSWindow. If successful, YES is returned. If aResponder isn't already the NSWindow's first responder, this method asks the object that currently is first responder to resign. However, if the old first responder refuses to resign, no changes are made and NO is returned.

The Application Kit uses this method to alter the first responder in response to mouse-down events; you can also use it to explicitly set the first responder from within your program. aResponder should be a NSResponder object. Typically, it's an NSView in the NSWindow's view hierarchy. See also firstResponder.

makeKeyAndOrderFront:

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

Moves the NSWindow to the front of the screen list (within its tier) and makes it the key window. This method can be used in an action message. See also setLevel:, orderFront:, orderBack:, orderOut:, orderWindow:relativeTo:.

makeKeyWindow

```
- (void)makeKeyWindow
```

Makes the NSWindow the key window, that is, the window that accepts keyboard events. See also becomeKeyWindow, becomeMainWindow.

makeMainWindow

```
- (void)makeMainWindow
```

Makes the NSWindow the main window. See also becomeMainWindow, becomeKeyWindow.

maxSize

```
- (NSSize)maxSize
```

Returns the maximum size that an NSWindow's frame can be sized. See also minSize, setMaxSize:, setMinSize:.

minSize

```
- (NSSize)minSize
```

Returns the minimum size that an NSWindow's frame can be sized. See also maxSize, setMaxSize:, setMinSize:.

miniaturize:

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

Hides the window and displays its miniwindow. If the window doesn't have a miniwindow counterpart, one is created. When this method completes successfully, it posts NSWindowDidMiniaturizeNotification.

A `miniaturize:` message is generated when the user clicks the miniaturize button in the `NSWindow`'s title bar. This method has a `sender` argument so that it can be used in an action message from an `NSControl`. It ignores this argument. See also `deminiaturize:`.

miniwindowImage

```
- (NSImage *)miniwindowImage
```

Returns the image that's displayed in the miniwindow. See also `setMiniwindowImage:`, `miniwindowTitle`, `setMiniwindowTitle:`.

miniwindowTitle

```
- (NSString *)miniwindowTitle
```

Returns the title that's displayed in the miniwindow. See also `setMiniwindowTitle:`, `miniwindowImage`, `setMiniwindowImage:`.

mouseLocationOutsideOfEventStream

```
- (NSPoint)mouseLocationOutsideOfEventStream
```

Provides current location of the cursor, in base coordinates.

nextEventMatchingMask:

```
- (NSEvent *)nextEventMatchingMask:(unsigned int)mask
```

Returns the next event object for the application that matches the events indicated by the given event `mask`. See the Event Handling section of the Types and Constants chapter for a list of event masks. See also `nextEventMatchingMask:untilDate:inMode: dequeue:`.

nextEventMatchingMask:untilDate:inMode: dequeue:

```
- (NSEvent *)nextEventMatchingMask:(unsigned int)mask
    untilDate:(NSDate *)expiration inMode:(NSString *)mode
    dequeue:(BOOL)deqFlag
```

Returns the next event object for the application that matches the event `mask`, and that occurs before time `expiration`. Until `expiration`, the run loop runs in `mode`, which can be one of the following values:

- `NSEventTrackingRunLoopMode`
- `NSModalPaneRunLoopMode`

If `flag` is `YES`, the event is removed from the event queue. See also `nextEventMatchingMask:`.

orderBack:

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

Moves the `NSWindow` to the back of its tier in the screen list. This method may also change the key window and the main window. See also `orderFront:`, `makeKeyAndOrderFront:`.

orderFront:

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

Moves the `NSWindow` to the front of its tier in the screen list. This method may also change the key window and main window. See `orderBack:`, `makeKeyAndOrderFront:`.

orderFrontRegardless

```
- (void)orderFrontRegardless
```

Moves the `NSWindow` to the front of its tier, even if the `NSWindow`'s application isn't active. Normally an `NSWindow` can't be moved in front of the key window unless the `NSWindow` and the key window are in the same application. You should rarely need to invoke this method; it's designed to be used when applications are cooperating such that an active application (with the key window) is using another application to display data. If the `NSWindow` is currently miniaturized, this method posts the notification `NSWindowDidDeminiaturizeNotification` with the window object to the default notification center.

orderOut:

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

Removes the window object from the screen list. This method may change the key window and the main window. See also `makeKeyAndOrderFront:`.

orderWindow:relativeTo:

```
- (void)orderWindow:(NSWindowOrderingMode)place
    relativeTo:(int)otherWin
```

Repositions the `NSWindow` in position `place` relative to `otherWin` within the screen list. If the window is currently miniaturized, this method posts the `NSWindowDidDeminiaturizeNotification` notification to the default notification center. `place` can be on of the following values:

- `NSWindowAbove`
- `NSWindowBelow`
- `NSWindowOut`

See also `makeKeyAndOrderFront:`.

performClose:

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

Simulates the user clicking the close button by momentarily highlighting the button and then closing the `NSWindow`. If the `NSWindow`'s delegate or the `NSWindow` itself implements `windowWillClose:`, then that message is sent with the `NSWindow` as the argument (only one such message is sent; if both the delegate and the `NSWindow` implement the method, only the delegate will receive the message). If the `NSWindow` doesn't have a close button, then the method calls `NSBeep()`. See also `close`, `performClick:` (`NSButton`).

performMiniaturize:

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

Simulates the user clicking the miniaturize button by momentarily highlighting the button then miniaturizing the `NSWindow`. If the `NSWindow` doesn't have a miniaturize button, then this method calls `NSBeep()`. See also `performClick:` (`NSButton`).

postEvent:atStart:

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

Posts an event for the NSApplication. If atStart is YES, the event goes to the beginning of the event queue. See also postEvent:atStart: (NSApplication).

print:

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

Prints the NSWindow (all the NSViews in its view hierarchy including the frame view). This method brings up a Print panel before printing begins. See also fax:, NSPrintOperation.

registerForDraggedTypes:

```
- (void)registerForDraggedTypes:(NSArray *)newTypes
```

Registers the NSPasteboard types (newTypes) that the NSWindow accepts in an image-dragging session. Argument values are NSPasteboard types, not file extensions (you can't register for specific file extensions). For example, the following registers an NSWindow as accepting files:

```
NSArray *fileType = [NSArray arrayWithObjects:
NSFilenamesPboardType, nil];
[aWindow registerForDraggedTypes:fileType ];
```

Note – Registering a window for dragged types automatically makes it a candidate destination object during a dragging session. As such, it must implement some or all of the NSDraggingDestination protocol methods. As a convenience, NSWindow provides default implementations of these methods (in general, the dragging destination methods are forwarded to the NSWindow's delegate). See the NSDraggingDestination protocol description for details.

See also dragImage:at:offset:event:pasteboard:source: slideBack:, unregisterDraggedTypes.

representedFilename

```
- (NSString *)representedFilename
```

Returns the filename associated with this NSWindow (regardless of the title string). See also setRepresentedFilename:, setTitleWithRepresentedFilename:.

resetCursorRects

```
- (void)resetCursorRects
```

Removes all existing cursor rectangles from the NSWindow, then recreates the cursor rectangles by sending a resetCursorRects message to every NSView in the NSWindow's view hierarchy. This method is typically invoked by the NSApplication object when it detects that the key window's cursor rectangles are invalid. In program code, it's more efficient to invoke invalidateCursorRectsForView:, rather than this method, to fix invalid cursor rectangles. See also areCursorRectsEnabled, resetCursorRects (NSView).

resignKeyWindow

```
- (void)resignKeyWindow
```

Records that the NSWindow object is no longer the key window. This method posts the notification NSWindowDidResignKeyNotification with the receiving object to the default notification center. See the "Notifications" section of the Application Kit's "Types and Constants" chapter for more information on notifications. You never need to invoke this method; it's invoked automatically when the NSWindow resigns key window status. The method sends resignKeyWindow to the NSWindow's first responder, and sends windowDidResignKey: to the NSWindow's delegate (if the respective objects can respond). See also becomeKeyWindow.

resignMainWindow

```
- (void)resignMainWindow
```

Records that the NSWindow is no longer the main window. This method posts the notification NSWindowDidResignMainNotification with the receiving object to the default notification center. See the Notifications section of the Application Kit Types and Constants chapter for more information on notifications.

resizeFlags

```
– (int)resizeFlags
```

Valid only while the NSWindow is being resized, this method returns the flags field of the event record for the mouse-down event that initiated the resizing session. The integer encodes, as a mask, information such as the modifier key that was held down when the event occurred. Because of its limited validity, this method should only be invoked from within an implementation of the delegate method windowDidResize:. Note that the default implementation of this method returns 0.

resizeIncrements

```
- (NSSize)resizeIncrements
```

Returns the increment used in window resizing.

restoreCachedImage

```
- (void)restoreCachedImage
```

This method will redraw the image cached by cacheImageInRect: back into the window, erasing whatever was previously there. Note that this method will not flush the window. For the image restoration to appear in the window, you may need to call flushWindowIfNeeded. Only call flushWindowIfNeeded if you are not already calling flushWindow following your restoreCachedImage call, or you may get some flicker. See also cacheImageInRect:, discardCachedImage.

saveFrameUsingName:

```
– (void)saveFrameUsingName:(NSString *)name
```

Saves the NSWindow's frame rectangle as a system default. With the companion method setFrameUsingName:, you can save and reset an NSWindow's frame over various launchings of an application. The default is owned by the application, filed under the name "NSWindow Frame *name*". See also removeFrameUsingName: (class method), setFrameUsingName:, frameAutosaveName, setFrameAutosaveName:, setFrameFromString:, stringWithSavedFrame.

screen

```
- (NSScreen *)screen
```

Returns the screen that the NSWindow is on. If the NSWindow is partly on one screen and partly on another, the screen where most of it lies is the one returned. See also deepestScreen.

sendEvent:

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

Dispatches mouse and keyboard events. If this method is dispatching a window exposed event, it posts the NSWindowDidExposeNotification notification with the receiving object and, in the notification's dictionary, a rectangle describing the exposed area (with the key NSExposedRect) to the default notification center. If this method is dispatching a screen changed event, it posts the NSWindowDidChangeScreenNotification with the receiving object. If this method is dispatching a window moved event, it posts NSWindowDidMoveNotification. See the Notifications section of the Applications Kit's Types and Constants chapter for more information on notifications.

setAcceptsMouseMovedEvents:

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

If flag is YES, the NSWindow accepts mouse-moved events. If flag is NO, the NSWindow does not accept mouse-moved events.

setAutodisplay:

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

If flag is YES, the NSWindow displays all views requiring redrawing when update is invoked. See also isAutodisplay.

setBackgroundColor:

```
- (void)setBackgroundColor:(NSColor *)color
```

Sets the color that fills the NSWindow's content area. See also backgroundColor.

setBackingType:

```
(void)setBackingType:(NSBackingStoreType)type
```

Sets the type of backing used by the NSWindow's window device.

- NSBackingStoreRetained
- NSBackingStoreNonretained
- NSBackingStoreBuffered

This method can only be used to switch a buffered NSWindow to retained or vice versa; you can't change the backing type of a nonretained NSWindow (a PostScript error is generated if you attempt to do so). See also backingType, NSBackingStoreType (Display Postscript "Types and Constants" chapter).

setContentSize:

```
- (void)setContentSize:(NSSize)aSize
```

Resizes the window's content area to aSize. This method calls frameRectForContentRect:styleMask: with the new content rectangle size, and the current style.

setContentView:

```
- (void)setContentView:(NSView *)aView
```

Makes aView the NSWindow's content view. The previous content view is removed from the NSWindow's view hierarchy. aView is resized to fit precisely within the content area of the NSWindow. You can transform the content view's coordinate system, but you can't alter its size or location directly. See also contentView.

setDelegate:

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

Makes anObject the NSWindow's delegate, and returns self. An NSWindow's delegate is given a chance to respond to action messages that work their way up the responder chain to the NSWindow through NSApplication's sendAction:to:from: method. It can also respond to notification messages sent by the NSWindow. See also delegate.

setDepthLimit:

```
- (void)setDepthLimit:(NSWindowDepth)limit
```

Sets the window's depth limit to limit which can be one of the following values:

- NSDefaultDepth
- NSTwoBitGrayDepth
- NSEightBitGrayDepth
- NSTwelveBitRGBDepth
- NSTwentyFourBitRGBDepth

See also depthLimit.

setDocumentEdited:

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

Sets whether the NSWindow's document has been edited. If flag is YES, the NSWindow's close button will display a broken "X" to indicate that the document needs to be saved. If flag is NO, the close button will be shown with a solid "X". The default is NO.

setDynamicDepthLimit:

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

Sets whether the NSWindow's depth limit should change to match the depth of the display device that it's on. If flag is YES, the depth limit will depend on which screen the NSWindow is on. If flag is NO, the NSWindow will have the default depth limit. A different, and nondynamic, depth limit can be set with the setDepthLimit: method. See also depthLimit.

setExcludedFromWindowsMenu:

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

Sets whether the receiving window object is omitted from the NSWindow's menu. If flag is YES, it won't be listed in the menu. If flag is NO, it will be listed when the NSWindow, or its miniwindow, is on-screen. The default is NO. See also isExcludedFromWindowsMenu.

setFrame:display:

```
- (void)setFrame:(NSRect)frameRect display:(BOOL)flag
```

Moves and/or resizes the NSWindow frame to frameRect. If flag is YES, the NSWindow is displayed; otherwise the NSWindow is not displayed. This method posts the NSWindowDidResizeNotification notification with the receiving object to the default notification center. See the Notifications section of the Application Kit's Types and Constants chapter for more information on notifications. See also setFrameOrigin:, setFrameTopLeftPoint:.

setFrameAutosaveName:

```
- (BOOL)setFrameAutosaveName:(NSString *)name
```

Sets the name that's used to automatically save the NSWindow's frame rectangle in the system defaults. If name isn't NULL, the NSWindow's frame is saved as a default (as described in saveFrameUsingName:) under the given name each time the frame changes. Passing NULL as an argument turns off this automation. An NSWindow can have only one frame autosave name at a time; if the NSWindow already has an autosave name, the old one is replaced. If name is already being used as an autosave name by an NSWindow in this application, the name isn't set and this method returns NO; otherwise returns YES. See also frameAutosaveName, saveFrameUsingName:, removeFrameUsingName: (class method).

setFrameFromString:

```
- (void)setFrameFromString:(NSString *)string
```

Sets the frame rectangle from `string`, which encodes the position and dimensions of the frame rectangle and the position and dimensions of the screen. See also `stringWithSavedFrame`, `setFrameUsingName:`, `removeFrameUsingName:` (class method).

setFrameOrigin:

```
- (void)setFrameOrigin:(NSPoint)aPoint
```

Moves the window by changing its frame origin (lower left corner) to `aPoint`. See also `setFrameTopLeftPoint:`, `frame`.

setFrameTopLeftPoint:

```
- (void)setFrameTopLeftPoint:(NSPoint)aPoint
```

Moves the window by changing its top-left corner to `aPoint`. See also `frame`.

setFrameUsingName:

```
- (BOOL)setFrameUsingName:(NSString *)name
```

Sets the frame rectangle from the named default. This method returns `YES` if name exists in the system defaults; otherwise the frame rectangle isn't set, and `NO` is returned. See also `frameAutosaveName`, `saveFrameUsingName:`, `setFrameFromString:`, `removeFrameUsingName:` (class method).

setHidesOnDeactivate:

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

Determines whether the `NSWindow` will disappear when the application is inactive. If flag is `YES`, the `NSWindow` is hidden (taken out of the screen list) when the application stops being the active application. If flag is `NO`, the `NSWindow` stays on-screen. The default for `NSWindow`s is `NO`; the default for `NSPanel`s and `NSMenu`s is `YES`.

setLevel:

```
- (void)setLevel:(int)newLevel
```

Resets the window level to `newLevel`. The following values represent the `NSWindow` levels:

- `NSNormalWindowLevel`
- `NSFloatingWindowLevel`
- `NSDockWindowLevel`
- `NSSubmenuWindowLevel`
- `NSMainMenuWindowLevel`

For more information on window levels, see the `NSWindow` section of the Application Kit's "Types and Constants" chapter. See also `level`.

setMaxSize:

```
- (void)setMaxSize:(NSSize)aSize
```

Sets the `NSWindow`'s maximum size to `aSize`. See also `maxSize`, `setMinSize:`.

setMinSize:

```
- (void)setMinSize:(NSSize)aSize
```

Sets the `NSWindow`'s minimum size to `aSize`. See also `minSize`, `setMaxSize:`.

setMiniwindowImage:

```
- (void)setMiniwindowImage:(NSImage *)image
```

Sets the `image` that's displayed in the `NSWindow`'s miniwindow.

setMiniwindowTitle:

```
- (void)setMiniwindowTitle:(NSString *)title
```

Sets the `title` that's displayed in the miniwindow. Normally, the miniwindow's title is taken, often abbreviated, from that of the `NSWindow`. This method is guaranteed to work only if the miniwindow is currently visible. In the latter case, the miniwindow's title is automatically redisplayed. Note that setting the `NSWindow`'s title (through `setTitle:` or `setTitleWithRepresentedFilename:`) will automatically reset the miniwindow's title to that of the `NSWindow`.

setOneShot:

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

Sets whether the backing-store memory that the NSWindow object manages should be freed when the NSWindow is removed from the screen. This is convenient for NSWindows used once or twice but not displayed continually. The default is NO. See also isOneShot.

setReleasedWhenClosed:

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

If flag is YES the NSWindow object is released upon closing; if flag is NO, the object is retained. See also close.

setRepresentedFilename:

```
- (void)setRepresentedFilename:(NSString *)aString
```

Alters aString by formatting it as a path and file name, then sets the internal file name associated with this window to the result. The format of the file name associate with the NSWindow is the file name, followed by a dash, followed by the path, with the dash surrounded by two space. For example:

```
MyFile - /Net/sever/group/home
```

If aString doesn't include a path to the file, the current working directory is used. This method doesn't affect the title string. See setTitleWithRepresentedFilename:.

setResizeIncrements:

```
- (void)setResizeIncrements:(NSSize)increments
```

Sets the increment used in window resizing.

setTitle:

```
- (void)setTitle:(NSString *)aString
```

Makes aString the NSWindow's title.

setTitleWithRepresentedFilename:

```
- (void)setTitleWithRepresentedFilename:(NSString *)aString
```

Invokes `setRepresentedFilename:` and makes the resulting string the `NSWindow`'s title.

setViewsNeedDisplay:

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

If `flag` is `YES`, then some of the `NSWindow`'s views need to be redrawn. If `flag` is `NO`, then no redrawing is necessary. See also `viewsNeedDisplay`.

stringWithSavedFrame

```
- (NSString *)stringWithSavedFrame
```

Returns a string encoding the position and dimensions of the frame rectangle and the position and dimensions of the screen. See also `saveFrameUsingName:`, `setFrameFromString:`, `removeFrameUsingName:` (class method).

styleMask

```
- (unsigned int)styleMask
```

Returns the `NSWindow`'s style mask, which can be one of the following values:

- `NSBorderlessWindowMask`
- `NSTitledWindowMask`
- `NSClosableWindowMask`
- `NSMiniaturizableWindowMask`
- `NSResizableWindowMask`

title

```
- (NSString *)title
```

Returns the `NSWindow`'s title string. See also `setTitle:`.

tryToPerform:with:

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

Aids in dispatching action messages (`anAction`) to `anObject`. This method gives the `NSWindow`'s delegate a chance to respond to the action message before passing the message up the responder chain. If a receiver for `anAction` is found, this method returns `YES`. Otherwise, it returns `NO`. See also `delegate`.

unregisterDraggedTypes

```
- (void)unregisterDraggedTypes
```

Unregisters the `NSWindow` as a recipient of dragged images. See also `registerForDraggedTypes:`.

update

```
- (void)update
```

Update's the `NSWindow`'s display and cursor rectangles. This method is invoked after every event. The default implementation of this method does nothing more than post the `NSWindowDidUpdateNotification` notification. A subclass can reimplement this method to perform specialized operations, but should send an `update` message to `super` just before returning. For example, the `NSMenu` class implements this method to disable and enable menu commands as appropriate.

A window is automatically sent an `update` message before it's ordered into the screen list. If the `NSApplication` object has received a `setWindowsNeedUpdate:YES` message, each visible `NSWindow` in the application is sent an `update` message after every event in the main event loop.

You can manually cause an `update` message to be sent to all visible `NSWindows` through `NSApplication`'s `updateWindows` method. See the "Notifications" section of the Application Kit's "Types and Constants" chapter for more information on notifications.

useOptimizedDrawing:

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

Informs the NSWindow whether to optimize focusing and drawing when NSViews are displayed. The optimizations may prevent sibling subviews from being displayed in the correct order—this matters only if the subviews overlap. You should always set flag to YES if there are no overlapping subviews within the NSWindow. The default is NO.

validRequestorForSendType:returnType:

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

Returns whether the NSWindow can respond to a service with send and receive types sendType and returnType. This message is passed to the NSWindow's delegate, if the delegate can respond and isn't an NSResponder with its own next responder. If the delegate can't respond or returns nil, this method passes the message to the NSApplication object. If the NSApplication object returns nil, this method also returns nil, indicating that no object was found that could supply sendType data for a remote message from the Services menu and accept back returnType data. If such an object was found, it is returned.

viewsNeedDisplay

```
- (BOOL)viewsNeedDisplay
```

Returns YES if some of the receiving NSWindow's views need redrawing; returns NO otherwise. See also setViewsNeedDisplay:.

windowNumber

```
- (int)windowNumber
```

Returns the window number of the NSWindow's window device. Each window device in an application is given a unique window number—note that this isn't the same as the global window number assigned by the Window Server. You use this number as the second argument of orderWindow:relativeTo:. If the NSWindow doesn't have a window device, the return value will be equal to or less than 0.

worksWhenModal

```
- (BOOL)worksWhenModal
```

This method should be overridden to return YES if the NSWindow can receive keyboard and mouse events when there's a modal panel (an attention panel) on-screen. The default implementation returns NO. Only NSPanel objects should change this default.

Methods Implemented by the Delegate

windowDidBecomeKey:

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

Sent by the default notification center to notify the delegate that the window is the key window. aNotification is always NSWindowDidBecomeKeyNotification. If the delegate implements this method, it's automatically registered to receive this notification. See the "Notifications" section of the "Types and Constants" chapter.

windowDidBecomeMain:

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

Sent by the default notification center to notify the delegate that the window is the main window. aNotification is always NSWindowDidBecomeMainNotification. If the delegate implements this method, it's automatically registered to receive this notification. See the "Notifications" section of the "Types and Constants" chapter.

windowDidChangeScreen:

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

Sent by the default notification center to notify the delegate that the window changed screens. aNotification is always NSWindowDidChangeScreenNotification. If the delegate implements this method, it's automatically registered to receive this notification. See the "Notifications" section of the "Types and Constants" chapter.

windowDidDeminiaturize:

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

Sent by the default notification center to notify the delegate that the window was restored to screen. `aNotification` is always `NSWindowDidDeminiaturizeNotification`. If the delegate implements this method, it's automatically registered to receive this notification. See the "Notifications" section of the "Types and Constants" chapter.

windowDidExpose:

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

Sent by the default notification center to notify the delegate that the window was exposed. `aNotification` is always `NSWindowDidExposeNotification`. If the delegate implements this method, it's automatically registered to receive this notification. See the "Notifications" section of the "Types and Constants" chapter.

windowDidMiniaturize:

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

Sent by the default notification center to notify the delegate that the window was miniaturized. `aNotification` is always `NSWindowDidMiniaturizeNotification`. If the delegate implements this method, it's automatically registered to receive this notification. See the "Notifications" section of the "Types and Constants" chapter.

windowDidMove:

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

Sent by the default notification center to notify the delegate that the window did move. `aNotification` is always `NSWindowDidMoveNotification`. If the delegate implements this method, it's automatically registered to receive this notification. See the "Notifications" section of the "Types and Constants" chapter.

windowDidResignKey:

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

Sent by the default notification center to notify the delegate that the window isn't the key window. `aNotification` is always `NSWindowDidResignKeyNotification`. If the delegate implements this method, it's automatically registered to receive this notification. See the "Notifications" section of the "Types and Constants" chapter.

windowDidResignMain:

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

Sent by the default notification center to notify the delegate that the window isn't the main window. `aNotification` is always `NSWindowDidResignMainNotification`. If the delegate implements this method, it's automatically registered to receive this notification. See the "Notifications" section of the "Types and Constants" chapter.

windowDidResize:

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

Sent by the default notification center to notify the delegate that the window was resized. `aNotification` is always `NSWindowDidResizeNotification`. If the delegate implements this method, it's automatically registered to receive this notification. See the "Notifications" section of the "Types and Constants" chapter.

windowDidUpdate:

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

Sent by the default notification center to notify the delegate that the window was updated. `aNotification` is always `NSWindowDidUpdateNotification`. If the delegate implements this method, it's automatically registered to receive this notification. See the "Notifications" section of the "Types and Constants" chapter.

windowShouldClose:

```
- (BOOL)windowShouldClose:(id)sender
```

Notifies delegate that the window is about to close.

windowWillClose:

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

Sent by the default notification center to notify the delegate that the window will close. `aNotification` is always `NSWindowWillCloseNotification`. If the delegate implements this method, it's automatically registered to receive this notification. See the "Notifications" section of the "Types and Constants" chapter.

windowWillReturnFieldEditor:toObject:

```
- (id)windowWillReturnFieldEditor:(NSWindow *)sender
    toObject:(id)client
```

Lets the delegate provide another text object for the field editor.

NSWorkspace

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

Class Description

An `NSWorkspace` object responds to application requests to perform a variety of services:

- Opening, manipulating, and obtaining information about files and devices
- Tracking changes to the file system, devices, and the user database
- Launching applications
- Miscellaneous services such as animating an image and requesting additional time before power off

An `NSWorkspace` object is made available through the `sharedWorkspace` context method. For example, the following statement uses an `NSWorkspace` object to request that a file be opened in the Edit application:

```
[[NSWorkspace sharedWorkspace] openFile:@"/Myfiles/README"
    withApplication:@"Edit"];
```

Method Types

Activity	Class Method
Creating a workspace	+ sharedWorkspace
Opening files	– openFile: – openFile:fromImage:at:inView: – openFile:withApplication: – openFile:withApplication:andDeactivate: – openTempFile:
Manipulating files	– performFileOperation:source:destination: files:tag: – selectFile:inFileViewerRootedAtPath:
Requesting information about files	– fullPathForApplication: – getFileSystemInfoForPath:isRemovable: isWritable:isUnmountable:description: – getInfoForFile:application:type: – iconForFile: – iconForFiles: – iconForFileType:
Tracking changes to the file system	– fileSystemChanged – noteFileSystemChanged
Updating registered services and file types	– findApplications
Launching and manipulating applications	– hideOtherApplications – launchApplication: – launchApplication:showIcon:autolaunch:
Unmounting a device	– unmountAndEjectDeviceAtPath:
Tracking status changes for devices	– checkForRemovableMedia – mountNewRemovableMedia – mountedRemovableMedia
Notification center	– notificationCenter

Activity	Class Method
Tracking changes to the user defaults database	– noteUserDefaultsChanged – userDefaultsChanged
Animating an image	– slideImage:from:to:
Requesting additional time before power off or logout	– extendPowerOffBy:

Class Methods

sharedWorkspace

```
+ (NSWorkspace *)sharedWorkspace
```

Returns a shared workspace.

Instance Methods

checkForRemovableMedia

```
- (void)checkForRemovableMedia
```

Causes the Workspace Manager to poll the system's drives for any disks that have been inserted but not yet mounted. Asks the Workspace Manager to mount the disk asynchronously and returns immediately. See also `mountNewRemovableMedia`, `mountedRemovableMedia`.

extendPowerOffBy:

```
- (int)extendPowerOffBy:(int)requested
```

Requests more time before the power goes off or the user logs out. Returns the granted number of additional milliseconds.

fileSystemChanged

```
- (BOOL)fileSystemChanged
```

Returns whether a change to the file system has been registered with a `noteFileSystemChanged` message since the last `fileSystemChanged` message.

findApplications

```
- (void)findApplications
```

Instructs Workspace Manager to examine all applications in the normal places and update its records of registered services and file types. See also `launchApplication:`.

fullPathForApplication:

```
- (NSString *)fullPathForApplication:(NSString *)appName
```

Returns the full path for the application `appName`, and returns `nil` if it isn't found. See also `getFileSystemInfoForPath:isRemovable:isWritable:isUnmountable:description:`, `getInfoForFile:application:type:`, `iconForFile:`, `iconForFiles:`, `iconForFileType:`.

getFileSystemInfoForPath:isRemovable:isWritable:isUnmountable:description:

```
- (BOOL)getFileSystemInfoForPath:(NSString *)fullPath
    isRemovable:(BOOL *)removableFlag
    isWritable:(BOOL *)writableFlag
    isUnmountable:(BOOL *)unmountableFlag
    description:(NSString **)description
    type:(NSString **)fileSystemType
```

Describes the file system at `fullPath` in `description` and `fileSystemType`, sets the flags appropriately, and returns `YES` if `fullPath` is a file system mount point, or `NO` if it isn't. See also `getInfoForFile:application:type:`, `fullPathForApplication:`.

getInfoForFile:application:type:

```
- (BOOL)getInfoForFile:(NSString *)fullPath
    application:(NSString **)appName type:(NSString **)type
```

Retrieves information about the file specified by `fullPath`, sets `appName` to the application the Workspace Manager would use to open `fullPath`, sets `type` to a value or file name extension indicating the file's type, and returns `YES` upon success and `NO` otherwise. See also `fullPathForApplication:`.

hideOtherApplications

```
- (void)hideOtherApplications
```

Hides all applications other than the sender. See also `launchApplication:`.

iconForFile:

```
- (NSImage *)iconForFile:(NSString *)fullPath
```

Returns an NSImage with the icon for the single file specified by fullPath. See also iconForFiles:, iconForFileType:, fullPathForApplication:.

iconForFiles:

```
- (NSImage *)iconForFiles:(NSArray *)pathArray
```

Returns an NSImage with the icon for the files specified in pathArray, an array of NSStrings. If pathArray specifies one file, its icon is returned. If pathArray specifies more than one file, an icon representing the multiple selection is returned. See also iconForFile:, iconForFileType:, fullPathForApplication:.

iconForFileType:

```
- (NSImage *)iconForFileType:(NSString *)fileType
```

Returns an NSImage the icon for the file type specified by fileType. See also iconForFile:, iconForFiles:, fullPathForApplication:.

launchApplication:

```
- (BOOL)launchApplication:(NSString *)appName
```

Instructs Workspace Manager to launch the application appName and returns YES if the application was successfully launched and NO otherwise. See also launchApplication:showIcon:autolaunch:, hideOtherApplications.

launchApplication:showIcon:autolaunch:

```
- (BOOL)launchApplication:(NSString *)appName
    showIcon:(BOOL)showIcon autolaunch:(BOOL)autolaunch
```

Instructs Workspace Manager to launch the application `appName` displaying the application's icon if `showIcon` is `YES` and using the dock autolaunching defaults if `autolaunch` is `YES`. Returns `YES` if the application was successfully launched and `NO` otherwise. See also `launchApplication:`, `hideOtherApplications`.

mountNewRemovableMedia

```
- (NSArray *)mountNewRemovableMedia
```

Causes the Workspace Manager to poll the system's drives for any disks that have been inserted but not yet mounted, waits until the new disks have been mounted, and returns a list of full path names to all newly mounted disks. See also `mountedRemovableMedia`, `checkForRemovableMedia`, `unmountAndEjectDeviceAtPath:`.

mountedRemovableMedia

```
- (NSArray *)mountedRemovableMedia
```

Returns a list of the pathnames of all currently mounted removable disks. See also `mountNewRemovableMedia`, `checkForRemovableMedia`, `unmountAndEjectDeviceAtPath:`.

noteFileSystemChanged

```
- (void)noteFileSystemChanged
```

Informs Workspace Manager that the file system has changed. See also `fileSystemChanged`.

noteUserDefaultsChanged

```
- (void)noteUserDefaultsChanged
```

Informs Workspace Manager that the defaults database has changed. See also `userDefaultsChanged`.

notificationCenter

```
- (NSNotificationCenter *)notificationCenter
```

Returns the notification center for Workspace notifications. See also `NSNotificationCenter` (Foundation Kit "Classes" chapter).

openFile:

```
- (BOOL)openFile:(NSString *)fullPath
```

Instructs Workspace Manager to open the file specified by `fullPath` using the default application for its type; returns `YES` if file was successfully opened and `NO` otherwise. See also `openFile:fromImage:at:inView:`, `openFile:withApplication:`, `openFile:withApplication:andDeactivate:`, `openTempFile:`.

openFile:fromImage:at:inView:

```
- (BOOL)openFile:(NSString *)fullPath fromImage:(NSImage *)anImage
    at:(NSPoint)point inView:(NSView *)aView
```

Instructs Workspace Manager to open the file specified by `fullPath` using the default application for its type. To provide animation prior to opening, `anImage` should contain the file's icon, and its image should be displayed at `point`, using `aView`'s coordinates. Returns `YES` if the file was successfully opened and `NO` otherwise. See also `openFile:`.

openFile:withApplication:

```
- (BOOL)openFile:(NSString *)fullPath
    withApplication:(NSString *)appName
```

Instructs Workspace Manager to open the file specified by `fullPath` using the `appName` application. Returns `YES` if the file was successfully opened and `NO` otherwise. See also `openFile:`.

openFile:withApplication:andDeactivate:

```
- (BOOL)openFile:(NSString *)fullPath
    withApplication:(NSString *)appName andDeactivate:(BOOL)flag
```

Instructs Workspace Manager to open the file specified by `fullPath` using the `appName` application, where `flag` indicates if the sending application should be deactivated before the request is sent. Returns `YES` if the file was successfully opened and `NO` otherwise. See also `openFile:`.

openTempFile:

```
- (BOOL)openTempFile:(NSString *)fullPath
```

Instructs Workspace Manager to open the temporary file specified by `fullPath` using the default application for its type. Returns `YES` if file was successfully opened and `NO` otherwise. See also `openFile:`.

performFileOperation:source:destination: files:tag:

```
- (BOOL)performFileOperation:(NSString *)operation
    source:(NSString *)source destination:(NSString *)destination
    files:(NSArray *)files tag:(int *)tag
```

Requests the Workspace Manager to perform a file `operation` on a set of `files` in the `source` directory specifying the `destination` directory if needed, using `tag` as an identifier for asynchronous operations. Returns `YES` if the operation succeeded and `NO` otherwise. See also `selectFile:inFileViewerRootedAtPath:`.

selectFile:inFileViewerRootedAtPath:

```
- (BOOL)selectFile:(NSString *)fullPath
    inFileViewerRootedAtPath:(NSString *)rootFullpath
```

Instructs Workspace Manager to select the file specified by `fullPath` opening a new file viewer if a path is specified by `rootFullpath`. Returns `YES` if the file was successfully selected and `NO` otherwise. See also `performFileOperation:source:destination: files:tag:`.

slideImage:from:to:

```
- (void)slideImage:(NSImage *)image from:(NSPoint)fromPoint
    to:(NSPoint)toPoint
```

Instructs Workspace Manager to animate a sliding image of `image` from `fromPoint` to `toPoint`, specified in screen coordinates.

unmountAndEjectDeviceAtPath:

```
- (BOOL)unmountAndEjectDeviceAtPath:(NSString *)path
```

Unmounts and ejects the device at `path`. Returns `YES` if the unmount succeeded and `NO` otherwise. See also `checkForRemovableMedia`.

userDefaultsChanged

```
- (BOOL)userDefaultsChanged
```

Returns whether a change to the defaults database has been registered with a `noteUserDefaultsChanged` message since the last `userDefaultsChanged` message.

NSChangeSpelling

Adopted by:	NSText
Declared In:	AppKit/NSSpellProtocol.h

Protocol Description

An object in the responder chain that can correct a misspelled word implements this protocol. See the description of the `NSSpellChecker` class for more information.

Instance Methods

changeSpelling:

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

Implement to replace the selected word in the receiver with a corrected version from the Spelling panel. This message is sent by the `NSSpellChecker` instance to the object whose text is being checked. To get the corrected spelling, the receiver asks the sender for the string value of its selected cell.

NSColorPickingCustom

Adopted by:	NSColorPicker
Declared In:	AppKit/NSColorPicking.h

Protocol Description

Together with the `NSColorPickingDefault` protocol, `NSColorPickingCustom` provides a way to add color pickers—custom user interfaces for color selection—to an application's `NSColorPanel`. The `NSColorPickingDefault` protocol provides basic behavior for a color picker. The `NSColorPicker` class adopts the `NSColorPickingDefault` protocol. The easiest way to implement a color picker is to create a subclass of `NSColorPicker` and use it as a base upon which to add the `NSColorPickingCustom` protocol. See also `NSColorPickingDefault`, `NSColorPicker`.

Method Types

Activity	ClassMethod
Getting the Mode	– currentMode – supportsMode:
Gettting the View	– provideNewView:
Setting the Current Color	– setColor:

Instance Methods

currentMode

```
- (int)currentMode
```

Returns the color picker's current mode (or submode, if applicable). The returned value should be unique to your color picker. The return value can be one of the following values:

- `NSGrayModeColorPanel`
- `NSRGBModeColorPanel`
- `NSCMYKModeColorPanel`

- NSHSBModeColorPanel
- NSCustomPaletteModeColorPanel
- NSColorListModeColorPanel
- NSWheelModeColorPanel

provideNewView:

```
- (NSView *)provideNewView:(BOOL)firstRequest
```

Returns the view containing the color picker's user interface. This message is sent to the color picker whenever the color panel attempts to display it. The argument indicates whether this is the first time the message has been sent; if `firstRequest` is YES, the method should perform any initialization required (such as lazily loading a nib file), or any custom initialization required for your color picker.

setColor:

```
- (void)setColor:(NSColor *)aColor
```

Adjusts the color picker to make `aColor` the currently selected color. This method is invoked on the current color picker each time `NSColorPanel`'s `setColor:` method is invoked. If `aColor` is actually different from the color picker's color (as it would be if, for example, the user dragged a color into the color panel's color well) this method could be used to update the color picker's color to reflect the change. See also `setColor:` (`NSColorPanel`).

supportsMode:

```
- (BOOL)supportsMode:(int)mode
```

Returns YES if the receiver supports the specified picking `mode`. This method attempts to restore the user's previously selected mode, and is invoked when the `NSColorPanel` is first initialized. It is also invoked by `NSColorPanel`'s `setMode:` to find the color picker that supports a particular mode. See also `currentMode` for a list of current modes.

NSColorPickingDefault

Adopted by:	NSColorPicker
Declared In:	AppKit/NSColorPicking.h

Protocol Description

The `NSColorPickingDefault` protocol, together with the `NSColorPickingCustom` protocol, provides an interface for adding color pickers—custom user interfaces for color selection—to an application's `NSColorPanel`. The `NSColorPickingDefault` protocol provides basic behavior for a color picker. The `NSColorPickingCustom` protocol provides implementation-specific behavior.

The `NSColorPicker` class implements the `NSColorPickingDefault` protocol. The simplest way to implement your own color picker is to create a subclass of `NSColorPicker`, implementing the `NSColorPickingCustom` protocol in that subclass. However, it's possible to create a subclass of another class, such as `NSView`, and use it as a base upon which to add the methods of both `NSColorPickingDefault` and `NSColorPickingCustom`.

Color Picker Bundles

A class that implements the `NSColorPickingDefault` and `NSColorPickingCustom` protocols needs to be compiled and linked in an application's object file. However, your application need not explicitly create an instance of this class. Instead, your application's file package should include a directory named `ColorPickers`; within this directory you should place a directory `MyPickerClass.bundle` for each custom color picker your application implements. This bundle should contain all resources required for your color picker: nib files, TIFF files, and so on.

`NSColorPanel` will allocate and initialize an instance of each class for which a bundle is found in the `ColorPickers` directory. The class name is assumed to be the bundle directory name minus the `.bundle` extension.

Color Picker Buttons

`NSColorPanel` lets the user select a color picker from a matrix of `NSButtonCell`s. This protocol includes methods for providing and manipulating the image that gets displayed on the button. See also `NSColorPickingCustom`, `NSColorPicker`, `NSColorPanel`.

Method Types

Activity	ClassMethod
Initializing a Color Picker	– initWithPickerMask:colorPanel:
Adding Button Images	– insertNewButtonImage:in: – provideNewButtonImage
Setting the Mode	– setMode:
Using Color Lists	– attachColorList: – detachColorList:
Showing Opacity Controls	– alphaControlAddedOrRemoved:
Responding to a Resize View	– viewSizeChanged:

Instance Methods

`alphaControlAddedOrRemoved:`

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

Sent by the color panel when the opacity controls have been hidden or displayed. If the color picker has its own opacity controls, it should hide or display them, depending on whether the sender's `showsAlpha` method returns `NO` or `YES`.

`attachColorList:`

```
- (void)attachColorList:(NSColorList *)aColorList
```

Attaches the given color list to the receiver, if it isn't already displaying the list. You never need to inovke this method; it is invoked automatically by the `NSColorPanel` when its `attachColorList:` method is invoked. Since

NSColorPanel's list mode manages NSColorLists, this method need only be implemented by a custom color picker that manages NSColorLists itself. See also detachColorList:.

detachColorList:

```
- (void)detachColorList:(NSColorList *)aColorList
```

Removes the given color list from the receiver, unless the receiver isn't displaying the list. You never need to invoke this method; it is invoked automatically by the NSColorPanel when its detachColorList: method is invoked. Since NSColorPanel's list mode manages NSColorLists, this method need only be implemented by a custom color picker that manages NSColorLists itself. See also attachColorList:.

initWithPickerMask:colorPanel:

```
- (id)initWithPickerMask:(int)mask
    colorPanel:(NSColorPanel *)colorPanel
```

Initializes the receiver for the specified mask and colorPanel. This method is sent by the NSColorPanel to all implementors of the color picking protocols when the application's color panel is first initialized. If the color picker responds to any of the modes represented in mask, it should perform its initialization (if desired) and return self; otherwise it should do nothing and return nil. However, a custom color picker can instead delay initialization until it receives a provideNewView: message. In order for your color picker to receive this message, it must have a bundle in your application's "ColorPickers" directory (described in the Color Picker Bundles of the Protocol Description above).

mask is determined by the argument to the NSColorPanel method setPickerMask:. If no mask has been set, mask is NSColorPanelAllModesMask. If your color picker supports any additional modes, you should invoke the setPickerMask: method when your application intializes to notify the NSColorPanel class. This method should examine the mask and determine whether it supports any of the modes included there. You may also check the value in mask to enable or disable any subpickers or optional controls implemented by your color picker. Your color picker may also retain colorPanel in an instance variable for future

communication with the color panel. See the Color section of the Application Kit's Types and Constants chapter for more information on color-picker modes and mask.

insertNewButtonImage:in:

```
- (void)insertNewButtonImage:(NSImage *)newImage
    in:(NSButtonCell *)newButtonCell
```

Sets `newImage` as `newButtonCell`'s image. `newButtonCell` is the `NSButtonCell` object that lets the user choose the picker from the color panel. This method should perform application-specific manipulation of the image before it's inserted and displayed by the button cell. See also `provideNewButtonImage`.

provideNewButtonImage

```
- (NSImage *)provideNewButtonImage
```

Returns the image for the mode button that the user uses to select this picker in the color panel. This is the same image that the color panel uses as an argument when sending the `insertNewButtonImage:in:` message.

setMode:

```
- (void)setMode:(int)mode
```

Sets the color picker's `mode`. This method is invoked by `NSColorPanel`'s `setMode:` method to ensure that the color picker reflects the current mode. Most color pickers have only one mode, and thus don't need to do any work in this method. Others, like the standard sliders picker, have multiple modes. The standard mode values are

- `NSGrayModeColorPanel`
- `NSRGBModeColorPanel`
- `NSCMYKModeColorPanel`
- `NSHSBModeColorPanel`
- `NSCustomPaletteModeColorPanel`
- `NSColorListModeColorPanel`
- `NSWheelModeColorPanel`

viewSizeChanged:

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

Sent when the color picker's superview has been resized in a way that might affect the color picker. `sender` is the `NSColorPanel` that contains the color picker. Use this method to perform special preparation when resizing the color picker's view.

NSDraggingDestination (Informal Protocol)

Category Of:	NSObject
Declared In:	AppKit/NSDragging.h

Protocol Description

The `NSDraggingDestination` protocol declares methods that the destination (or recipient) of a dragged image must implement. The destination automatically receives `NSDraggingDestination` messages as an image enters, moves around inside, and then exits or is released within the destination's boundaries.

Note – Within this text the term *dragging session* means the entire process during which an image is selected, dragged, released, and is absorbed or rejected by the destination. A *dragging operation* is the action that the destination takes in absorbing the image when it's released. The *dragging source* is the object that "owns" the image that's being dragged. It's specified as an argument to the `dragImage:...` message, sent to a `NSWindow` or `NSView`, that instigated the dragging session.

The Dragged Image

The image that's dragged in an image-dragging session is an `NSImage` object that represents data that's put on the pasteboard. Although a dragging destination can access the `NSImage` (through a method described in the `NSDraggingInfo` protocol), its primary concern is with the pasteboard data that the `NSImage` represents—the dragging operation that a destination ultimately performs is on the pasteboard data, not on the image itself.

Valid Destinations

Dragging is a visual phenomenon. To be an image-dragging destination, an object must represent a portion of screen real estate; thus, only `NSWindow`s and `NSView`s can be destinations. Furthermore, you must announce the destination-candidacy of an `NSWindow` or `NSView` by sending it a `registerForDraggedTypes:` message. This method, defined in both classes, registers the pasteboard types that the object will accept. During a dragging session, a candidate destination will only receive `NSDraggingDestination` messages if the pasteboard types for which it is registered matches a type that's represented by the image that's being dragged.

Although `NSDraggingDestination` is declared as a protocol, the `NSView` and `NSWindow` subclasses that you create to adopt the protocol need only implement those methods that are pertinent. (The `NSView` and `NSWindow` classes provide private implementations for all of the methods.) In addition, an `NSWindow` or its delegate may implement these methods; the delegate's implementation takes precedent.

The Sender of Destination Messages

Each of the `NSDraggingDestination` methods sports a single argument: `sender`, the object that invoked the method. Within its implementations of the `NSDraggingDestination` methods, the destination can send `NSDraggingInfo` messages to `sender` to get more information on the current dragging session.

The Order of Destination Messages

The six `NSDraggingDestination` methods are invoked in a distinct order:

- As the image is dragged into the destination's boundaries, the destination is sent a `draggingEntered:` message.
- While the image remains within the destination, a series of `draggingUpdated:` messages are sent.
- If the image is dragged out of the destination, `draggingExited:` is sent and the sequence of `NSDraggingDestination` messages stops. If it re-enters, the sequence begins again (with a new `draggingEntered:` message).

- When the image is released, it either slides back to its source (and breaks the sequence) or a `prepareForDragOperation:` message is sent to the destination, depending on the value that was returned by the most recent invocation of `draggingEntered:` or `draggingUpdated:`.
- If the `prepareForDragOperation:` message returned `YES`, a `performDragOperation:` message is sent.
- Finally, if `performDragOperation:` returned `YES`, `concludeDragOperation:` is sent.

Method Types

Activity	ClassMethod
Before the Image is Released	– draggingEntered: – draggingExited: – draggingUpdated:
After the Image is Released	– concludeDragOperation: – performDragOperation: – prepareForDragOperation:

Instance Methods

`concludeDragOperation:`

```
- (void)concludeDragOperation:(id <NSDraggingInfo>)sender
```

Invoked when the dragging operation is complete (but only if the previous `performDragOperation:` returned `YES`). The destination implements this method to perform any tidying up that it needs to do. This is the last message that's sent from `sender` to the destination during a dragging session. See also `prepareForDragOperation:`, `performDragOperation:`.

`draggingEntered:`

```
- (unsigned int)draggingEntered:(id <NSDraggingInfo>)sender
```

Invoked when the dragged image enters the destination. Specifically, the message is sent when the hot spot on the cursor that's dragging the image enters any portion of the destination's bounds rectangle (if it's an `NSView`) or its frame rectangle (if it's an `NSWindow`).

This method must return a single value that indicates which dragging operation the destination will perform when the image is released. It should be one of the operations specified in the value returned by `sender`'s `draggingSourceOperationMask` method (see the `NSDragInfo` protocol). If none of the operations are appropriate, this method should return `NSDragOperationNone` (this is the default response if the method isn't implemented by the destination). See the Drag Operation section of the Application Kit's Types and Constants chapter for a list of dragging operation constants. See also `draggingExited:`, `draggingUpdated:`.

draggingExited:

```
- (void)draggingExited:(id <NSDraggingInfo>)sender
```

Invoked when the dragged image exits the destination (following, inversely, the geometric specification given in the description of `draggingEntered:`). See also `draggingEntered:`, `draggingUpdated:`.

draggingUpdated:

```
- (unsigned int)draggingUpdated:(id <NSDraggingInfo>)sender
```

Invoked periodically while the image is over the destination. The messages continue until the image is either released or exits. The return value follows the same rules as that for the `draggingEntered:` method. The default return value (if this method isn't implemented by the destination) is the value returned by the previous `draggingEntered:` message.

Only one destination at at time receives a sequence of `draggingUpdated:` messages. For example, if the cursor is within the bounds of two overlapping `NSView`s that are both valid destinations, the uppermost `NSView` receives these messages until the image is either released or exits. See also `draggingEntered:`, `draggingExited:`.

performDragOperation:

```
- (BOOL)performDragOperation:(id <NSDraggingInfo>)sender
```

Gives the destination an opportunity to perform the dragging operation. This method is invoked after the released image has been removed from the screen (but only if the previous `prepareForDragOperation:` message returned `YES`). The destination should implement this method to do the real work of importing the data represented by the image. If the destination accepts the data, it returns `YES`, otherwise it returns `NO`. The default (if the destination doesn't implement the method) is to return `NO`. See also `prepareForDragOperation:`, `concludeDragOperation:`.

`prepareForDragOperation:`

```
- (BOOL)prepareForDragOperation:(id <NSDraggingInfo>)sender
```

Invoked when the image is released (but only if the most recent `draggingEntered:` or `draggingUpdated:` message returned an acceptable drag-operation value). This method returns `YES` if it will perform the drag operation and `NO` if not. See also `performDragOperation:`, `concludeDragOperation:`.

NSDraggingInfo

Adopted by:	No OpenStep classes
Declared In:	AppKit/NSDragging.h

Protocol Description

The `NSDraggingInfo` protocol declares methods that supply information about a dragging session (see the `NSDraggingDestination` protocol, an informal protocol of `NSObject`, for definitions of dragging terms). A view or window first registers dragging types; it may then send `NSDraggingInfo` protocol messages while dragging occurs to get details about that dragging session.

`NSDraggingInfo` methods are designed to be invoked from within an object's implementation of the `NSDraggingDestination` protocol methods. An object that conforms to `NSDraggingInfo` is passed as the argument to each of the methods defined by `NSDraggingDestination`; `NSDraggingInfo` messages

should be sent to this conforming object. The Application Kit supplies an `NSDraggingInfo` object automatically so that you never need to create a class that implements this protocol.

Method Types

Activity	ClassMethod
Dragging-Session Information	– draggingDestinationWindow – draggingLocation – draggingPasteboard – draggingSequenceNumber – draggingSource – draggingSourceOperationMask
Image Information	– draggedImage – draggedImageLocation
Sliding the Image	– slideDraggedImageTo:

Instance Methods

`draggedImage`

```
– (NSImage *)draggedImage
```

Returns the image object that's being dragged. Don't invoke this method after the user has released the image, and don't release the object that this method returns. See also `draggedImageLocation`.

`draggedImageLocation`

```
– (NSPoint)draggedImageLocation
```

Returns the current location of the dragged image's origin. The image moves in lockstep with the cursor (the position of which is given by `draggingLocation`) but may be positioned at some offset. The point that's returned is reckoned in the base coordinate system of the destination object's `NSWindow`. See also `draggedImage`.

draggingDestinationWindow

```
- (NSWindow *)draggingDestinationWindow
```

Returns the destination's NSWindow. See also draggingLocation, draggingPasteboard, draggingSequenceNumber, draggingSource, draggingSourceOperationMask.

draggingLocation

```
- (NSPoint)draggingLocation
```

Returns the current location of the cursor's hot spot, reckoned in the base coordinate system of the destination object's NSWindow. See also draggingDestinationWindow.

draggingPasteboard

```
- (NSPasteboard *)draggingPasteboard
```

Returns the pasteboard that holds the dragged data. See also draggingDestinationWindow.

draggingSequenceNumber

```
- (int)draggingSequenceNumber
```

Returns a number that uniquely identifies the dragging session. See also draggingDestinationWindow.

draggingSource

```
- (id)draggingSource
```

Returns the source, or "owner," of the dragged image. Returns nil if the source isn't in the same application as the destination. See also draggingDestinationWindow.

draggingSourceOperationMask

```
- (unsigned int)draggingSourceOperationMask
```

Returns the dragging operation mask declared by the dragging source's `draggingSourceOperationMaskForLocal:` method. The elements in the mask will be one or more of the following:

- `NSDragOperationCopy`
- `NSDragOperationLink`
- `NSDragOperationGeneric`
- `NSDragOperationPrivate`

If the user is holding down a modifier key during the drag, the value that corresponds to the key (as shown in the table below) is AND 'ed with the source's mask.

Table 2-1

Modifier Key	Value
Control	`NSDragOperationLink`
Alternate	`NSDragOperationCopy`
Command	`NSDragOperationGeneric`

See also `draggingSourceOperationMaskForLocal:` (`NSDraggingSource` protocol).

`slideDraggedImageTo:`

```
- (void)slideDraggedImageTo:(NSPoint)screenPoint
```

Slides the image to the given location in the screen coordinate system. This method should only be invoked from within the destination's implementation of `prepareForDragOperation:`—in other words, after the user has released the image but before it's removed from the screen.

NSDraggingSource (Informal Protocol)

Category Of:	NSObject
Declared In:	AppKit/NSDragging.h

Protocol Description

`NSDraggingSource` declares methods that can (or must) be implemented by the source object in a dragging session. (See the `NSDraggingDestination` protocol for definitions of dragging terms.) This *dragging source* is specified as an argument to the `dragImage:...` message, sent to an `NSWindow` or `NSView`, that instigated the dragging session.

Of the methods declared below, only the `draggingSourceOperationMaskForLocal:` method *must* be implemented. The other methods are invoked only if the dragging source implements them. All four methods are invoked automatically during a dragging session—you never send an `NSDraggingSource` message directly to an object.

Method Types

Activity	ClassMethod
Querying the Source	– draggingSourceOperationMaskForLocal: – ignoreModifierKeysWhileDragging
Informing the Source	– draggedImage:beganAt: – draggedImage:endedAt:deposited:

Instance Methods

draggedImage:beganAt:

```
- (void)draggedImage:(NSImage *)image beganAt:(NSPoint)screenPoint
```

Invoked when the dragged `image` is displayed, but before it starts following the mouse. `screenPoint` is the origin of the image in screen coordinates.

draggedImage:endedAt:deposited:

```
– (void)draggedImage:(NSImage *)image endedAt:(NSPoint)screenPoint
    deposited:(BOOL)didDeposit
```

Invoked after the dragged `image` has been released and the dragging destination has been given a chance to operate on the data it represents. `screenPoint` is the location of image's origin when it was released reckoned in screen coordinates. `deposited` indicates whether the destination accepted the image.

draggingSourceOperationMaskForLocal:

```
-(unsigned int)draggingSourceOperationMaskForLocal:
    (BOOL)isLocal
```

Returns a mask giving the operations that can be performed on the dragged image's data. This is the only `NSDraggingSource` method that must be implemented by the source object. isLocal indicates whether the candidate destination object (the window or view over which the dragged image is poised) is in the same application as the source. This method should return a mask, built by OR'ing together applicable combinations of the following constants:

Drag Operation	Meaning
`NSDragOperationNone`	No operation possible
`NSDragOperationCopy`	The data represented by the image can be copied
`NSDragOperationLink`	The data can be shared
`NSDragOperationGeneric`	The operation can be defined by the destination
`NSDragOperationPrivate`	Private source / destination negotiation
`NSDragOperationAll`	Combines all the above

ignoreModifierKeysWhileDragging

```
(BOOL)ignoreModifierKeysWhileDragging
```

Returns `YES` if modifier keys should have no effect on the type of operation performed, and returns `NO` otherwise.

NSIgnoreMisspelledWords

Adopted by:	NSText
Declared In:	AppKit/NSSpellProtocol.h

Protocol Description

Implement this protocol to have the Ignore button in the Spelling panel function properly. The Ignore button allows the user to accept a word that the spelling checker believes is misspelled. In order for this action to update the "ignored words" list for the document being checked, the `NSIgnoreMisspelledWords` protocol must be implemented.

This protocol is necessary because a list of ignored words is useful only if it pertains to the entire document being checked, but the spelling checker (`NSSpellChecker` object) does not check the entire document for spelling at once. The spelling checker returns as soon as it finds a misspelled word. Thus, it checks only a subset of the document at any one time. The user usually wants to check the entire document, and so usually several spelling checks are run in succession until no misspelled words are found. This protocol allows the list of ignored words to be maintained per-document, even though the spelling checks are not run per-document.

The `NSIgnoreMisspelledWords` protocol specifies a method, `ignoreSpelling:`, which should be implemented like this:

```
- (void)ignoreSpelling:(id)sender
{
    [[NSSpellChecker sharedSpellChecker]
        ignoreWord:[[sender selectedCell] stringValue]
        inSpellDocumentWithTag:myDocumentTag];
}
```

The second argument to the `NSSpellChecker` method `ignoreWord:inSpellDocumentWithTag:` is a tag that the `NSSpellChecker` can use to distinguish the documents being checked. (See the discussion of "Matching a List of Ignored Words With the Document It Belongs To" in the description of the `NSSpellChecker` class.) Once the `NSSpellChecker` has a way to distinguish the various documents, it can append new ignored words to the appropriate list.

To make the ignored words feature useful, the application must store a document's ignored words list with the document. See the NSSpellChecker class description for more information.

Instance Methods

ignoreSpelling:

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

Implement to allow an application to ignore misspelled words on a document-by-document basis. This message is sent by the NSSpellChecker instance to the object whose text is being checked. To inform the NSSpellChecker that a particular spelling should be ignored, the receiver asks the NSSpellChecker for the string value of its selected cell. It then sends the NSSpellChecker an ignoreWord:inSpellDocumentWithTag: message.

NSMenuActionResponder (Informal Protocol)

Category Of:	NSObject
Declared In:	AppKit/NSMenu.h

Protocol Description

This informal protocol allows your application to update the enabled or disabled status of an NSMenuItem. It declares only one method, validateItem:. By default, every time a user event occurs, NSMenu automatically enables and disables each visible menu item based on criteria described later in this specification. Implement validateItem: in cases where you want to override NSMenu's default enabling scheme. This is described in more detail later.

There are two ways that NSMenuItems can be enabled or disabled: Explicitly, by sending the setEnabled: message, or automatically, as described below. NSMenuItems are updated automatically unless you send the message setAutoenablesItems:NO to the NSMenu object. You should never mix the two. That is, never use setEnabled: unless you have disabled the automatic updating.

Automatic Updating of NSMenuItems

Whenever a user event occurs, the NSMenu object updates the status of every visible menu item. To update the status of a menu item, NSMenu tries to find the object that responds to the NSMenuItem's action message. It searches the following objects in the following order until it finds one that responds to the action message.

- The NSMenuItem's target
- The key window's first responder
- The key window's delegate
- The main window's first responder
- The main window's delegate
- The NSApplication object
- The NSApplication's delegate
- The NSMenu's delegate

If none of these objects responds to the action message, the menu item is disabled. If NSMenu finds an object that responds to the action message, it then checks to see if that object responds to the validateItem: message (the method defined in this informal protocol). If validateItem: is not implemented in that object, the menu item is enabled. If it is implemented, the return value of validateItem: indicates whether the menu item should be enabled or disabled.

For example, the NSText object implements the copy: method. If your application has a Copy menu item that sends the copy: action message to the first responder, that menu item is automatically enabled any time an NSText object is the first responder of the key or main window. If you have an object that might become the first responder and that object could allow users to select something that they aren't allowed to copy, you can implement the validateItem: method in that object. validateItem: can return NO if the forbidden items are selected and YES if they aren't. By implementing validateItem:, you can have the Copy menu item disabled even though its target object implements the copy: method. If instead your object *never* permits copying, then you would simply not implement copy: in that object, and the item would be disabled automatically whenever the object is first responder.

If you send a setEnabled: message to enable or disable a menu item when the automatic updating is turned on, other objects might reverse what you have done after another user event occurs. Using setEnabled:, you can never

be sure that a menu item is enabled or disabled or will remain that way. If your application must use `setEnabled:`, you must turn off the automatic enabling of menu items (by sending `setAutoEnablesItems:NO` to `NSMenu`) in order to get predictable results.

Instance Methods

`validateItem:`

`- (BOOL)validateItem:(id <NSMenuItem>)aItem`

Implemented to override the default action of updating an `NSMenuItem`. Returns `YES` to enable the `NSMenuItem`, and returns `NO` to disable it.

NSMenuItem

Adopted By:	NSMenuCell
Declared In:	AppKit/NSMenuItem.h

Protocol Description

`NSMenuItem` defines objects that are used as command items in menus. How these items appear depends on the host system's user interface. See the `NSMenu` class specification for more information on menus.

Methods

`action`

`-(SEL)action`

Returns the menu item's action method selector.

`hasSubmenu`

`- (BOOL)hasSubmenu`

Returns `YES` if the menu item has a submenu.

isEnabled

```
-(BOOL)isEnabled
```

Returns YES if the menu item is enabled, and NO if not.

keyEquivalent

```
-(NSString*)keyEquivalent
```

Returns the receiver's basic keyboard equivalent.

setAction:

```
-(void)setAction:(SEL)aSelector
```

Sets the menu item's action method selector to aSelector.

setEnabled:

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

Enables the menu item if flag is YES, and disables the menu item if flag is NO.

setKeyEquivalent:

```
-(void)setKeyEquivalent:(NSString*)aString
```

Sets the menu item's basic key equivalent to aString.

setTag:

```
- (void)setTag:(unsigned int)anInt
```

Sets the menu item's tag to anInt.

setTarget:

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

Sets the menu item's target to anObject.

setTitle:

```
- (void)setTitle:(NSString*)aString
```

Sets the receiving menu item's title to `aString`.

tag

```
-(unsignedint)tag
```

Returns the menu item's tag.

target

```
-(id)target
```

Returns the menu item's target.

title

```
-(NSString*)title
```

Returns the menu item's title.

NSNibAwaking (Informal Protocol)

Category Of:	NSObject
Declared In:	AppKit/NSNibLoading.h

Protocol Description

This informal protocol consists of a single method, `awakeFromNib`. It's implemented to receive a notification message that's sent after objects have been loaded from an Interface Builder archive.

When `loadNibFile:owner:` or a related method loads an Interface Builder archive into an application, each custom object from the archive is first initialized with an `init` message (`initFrame:` if the object is a kind of `NSView`). Outlets are initialized via any `set`*Variable*`:` methods that are available, where *Variable* is the name of an instance variable. (These methods

are optional; the Objective C run time system automatically initializes outlets.) Finally, after all the objects are fully initialized, they each receive an `awakeFromNib` message.

The order in which objects are loaded from the archive is not guaranteed. Therefore, it's possible for a `set`*Variable*`:` message to be sent to an object before its companion objects have been unarchived. For this reason, `set`*Variable*`:` methods should not send messages to other objects in the archive. However, messages to other objects can safely be sent from within `awakeFromNib`—by this point it's assured that all the objects are unarchived and fully initialized.

Typically, `awakeFromNib` is implemented for only one object in the archive, the controlling or "owner" object for the other objects that are archived with it. For example, suppose that a nib file contained two views that must be positioned relative to each other at run time. Trying to position them when either one of the views is initialized (in a `set`*Variable*`:` method) might fail, since the other view might not be unarchived and initialized yet. However, it can be done in an `awakeFromNib` method:

```
- (void)awakeFromNib
{
    NSRect viewFrame;

    [firstView getFrame:&viewFrame];
    [secondView moveTo:viewFrame.origin.x +
                  someVariable :viewFrame.origin.y];
}
```

There's no default `awakeFromNib` method; an `awakeFromNib` message is only sent if an object implements it. The Application Kit declares a prototype for this method, but doesn't implement it.

Instance Methods

awakeFromNib

```
– (void)awakeFromNib
```

Implemented to prepare an object for service after it has been loaded from an Interface Builder archive—a so-called "nib file". An `awakeFromNib` message is sent to each object loaded from the archive, but only if it can respond to the message, and only after all the objects in the archive have been loaded and

initialized. When an object receives an `awakeFromNib` message, it's already guaranteed to have all its outlet instance variables set. There is no default `awakeFromNib` method.

NSServicesRequests (Informal Protocol)

Category Of:	NSObject
Declared In:	AppKit/NSApplication.h

Protocol Description

This informal protocol consists of two methods, `writeSelectionToPasteboard:types:` and `readSelectionFromPasteboard:`. The first is implemented to provide data to a remote service, and the second to receive any data the remote service might send back. Both respond to messages that are generated when the user chooses a command from the Services menu.

Instance Methods

`readSelectionFromPasteboard:`

```
- (BOOL)readSelectionFromPasteboard:(NSPasteboard *)pboard
```

Implemented to replace the current selection (that is, the text or objects that are currently selected) with data from `pboard`. The data would have been placed in the pasteboard by another application in response to a remote message from the Services menu. A `readSelectionFromPasteboard:` message is sent to the same object that previously received a `writeSelectionToPasteboard:types:` message. There is no default `readSelectionFromPasteboard:` method.

`writeSelectionToPasteboard:types:`

```
- (BOOL)writeSelectionToPasteboard:(NSPasteboard *)pboard
    types:(NSArray *)types
```

Implemented to write the current selection to `pboard`. The selection should be written as one or more the the data types listed in `types`. After writing the data, this method should return `YES`. If for any reason it can't write the data, it should return `NO`. A `writeSelectionToPasteboards:types:` message is sent to the first responder when the user chooses a command from the Services menu, but only if the receiver didn't return `nil` to a previous `validRequestorForSendType:andReturnType:` message (`NSResponder`, `NSApplication`). After this method writes the data to the pasteboard, a remote message is sent to the application that provides the service that the user requested. If the service provider supplies return data to replace the selection, the first responder will then receive a `readSelectionFromPasteboard:` message. There is no default version of this method.

NSTableDataSource (Informal Protocol)

Characteristic	Description
Category Of:	NSObject
Declared In:	AppKit / NSTableView.h

Protocol Description

`NSTableDataSource` declares the methods that an `NSTableView` uses to access the contents of its data source object. It determines how many rows to display by sending a `numberOfRowsInTableView:` message, and accesses individual values with the `tableView:objectValueForTableColumn:row:` and `tableView:setObjectValue:forTableColumn:row:` methods. A data source must implement the first two methods to work with an `NSTableView`, but if it doesn't implement the third. The `NSTableView` simply provides read-only access to its contents.

The `NSTableView` treats objects provided by its data source as values to be displayed in `NSCell` objects. If these objects aren't of common value classes such as `NSString`, `NSNumber`, and so on you'll need to create a custom `NSFormatter` to display them. See the `NSFormatter` class specification for more information.

Suppose that an NSTableView's column identifiers are set up as NSStrings containing the names of attributes for the column, such as *Last Name:, *City:, and so on, and that the data source stores its records as an NSArray, called records, of NSDictionary objects using those names as keys. In such a case, this implementation of tableView:objectValueForTableColumn:row: suffices to retrieve values for the table view:

```
- (id)tableView:(NSTableView
    objectValueForTableColumn:(NSTableColumn
    row:(int)rowIndex
{
    id theRecord, theValue;

    NSParameterAssert(rowIndex >= 0 && rowIndex < [records count]);
    theRecord = [records objectAtIndex:rowIndex];
    theValue = [theRecord objectForKey:[aTableColumn identifier]];
    return theValue;
}
```

Here's the corresponding method for setting values:

```
- (void)tableView:(NSTableView *) tableView
    setObjectValue:anObject
    forTableColumn:(NSTableColumn *) aTableColumn
    row:(int)rowIndex
{
id theRecord;
NSParameterAssert(rowIndex >= 0 && rowIndex < [records count]);
theRecord = [records objectAtIndex:rowIndex];
[theRecord setObject:anObject forKey:[aTableColumn identifier]];
return;
}
```

Finally, for completeness' sake, numberOfRowsInTableView: simply returns the count of the NSArray:

```
- (int)numberOfRowsInTableView:(NSTableView *)aTableView
{
    return [records count];
}
```

In each case, the NSTableView that sends the message is provided as aTableView. A data source object that manages several sets of data can choose the appropriate set based on which NSTableView sends the message.

Instance Methods

numberOfRowsInTableView:

```
-numberOfRowsInTableView:(NSTableView *)TableView
```

Returns the number of records managed for `aTableView` by the data source object. An `NSTableView` uses this method to determine how many rows it should create and display.

tableView:objectValueForTableColumn:row:

```
- (id)tableView:(NSTableView *)tableView
    objectValueForTableColumn:(NSTableColumn *)tableColumn
    row:(int)row
```

Returns an attribute value for the record in `aTableView` at `rowIndex`. `aTableColumn` contains the identifier for the attribute, which you get by using `NSTableColumn`'s identifier method. For example, if `aTableColumn` stands for the city that an employee lives in and `rowIndex` specifies the record for an employee who lives in Portland, this method returns an object with a string value of *Portland:. See the category description for an example.

tableView:setObjectValue:forTableColumn:row:

```
- (void)tableView:(NSTableView *)tableView
    setObjectValue:(id)object
    forTableColumn:(NSTableColumn *)tableColumn
    row:(int)row
```

Sets an attribute value for the record in `aTableView` at `rowIndex`. `anObject` is the new value, and `aTableColumn` contains the identifier for the attribute, which you get by using `NSTableColumn`'s identifier method. See the category description for an example.

Functions 3

Rectangle Drawing Functions

Optimize Drawing

NSEraseRect()

```
void NSEraseRect(NSRect aRect)
```

Erases the rectangle `aRect` by filling it with white. (This does not alter the current drawing color.)

NSHighlightRect()

```
void NSHighlightRect(NSRect aRect)
```

Highlights or unhighlights `aRect` by switching light gray for white and vice versa, when drawing on the screen. If not drawing to the screen, the rectangle is filled with light gray.

NSRectClip()

```
void NSRectClip(NSRect aRect)
```

Intersects the current clipping path with the rectangle `aRect`, to determine a new clipping path.

NSRectClipList()

```
void NSRectClipList(const NSRect *rects, int count)
```

Takes an array of `count` number of rectangles and intersects the current clipping path with each of them. The new clipping path is the graphic intersection of all the rectangles and the original clipping path.

NSRectFill()

```
void NSRectFill(NSRect aRect)
```

Fills the rectangle referred to by `aRect` with the current color.

NSRectFillList()

```
void NSRectFillList(const NSRect *rects, int count)
```

Fills an array of `count` rectangles with the current color.

NSRectFillListWithGrays()

```
void NSRectFillListWithGrays(const NSRect *rects,
    const float *grays, int count)
```

Fills each rectangle in the array `rects` with the gray whose value is stored at the corresponding location in the array `grays`. Both arrays must be `count` elements long. Avoid rectangles that overlap, because the order in which they'll be filled can't be guaranteed.

Draw a Bordered Rectangle

NSDrawButton()

```
void NSDrawButton(NSRect aRect, NSRect clipRect)
```

Draws the bordered light gray rectangle whose appearance signifies a button in the OpenStep user interface. `aRect` is the bounds for the button, but only the area where `aRect` intersects `clipRect` is drawn.

NSDrawGrayBezel()

```
void NSDrawGrayBezel(NSRect aRect, NSRect clipRect)
```

Draws a bordered light gray rectangle, `aRect`, with the appearance of a pushed-in button, clipped by intersecting with `clipRect`.

NSDrawGroove()

```
void NSDrawGroove(NSRect aRect, NSRect clipRect)
```

Draws a light gray rectangle `aRect`, clipped by intersecting with `clipRect`, whose border is a groove, giving the appearance of a typical box in the OpenStep user interface.

NSDrawTiledRects()

```
NSRect NSDrawTiledRects(NSRect boundsRect, NSRect clipRect,
const NSRectEdge *sides, const float *grays, int count)
```

Draws unfilled rectangle `boundsRect`, clipped by `clipRect`, whose border is defined by the parallel arrays `sides` and `grays`, both of length `count`. Each element of `sides` specifies an edge of the rectangle, which is drawn with a width of 1.0 using the corresponding gray level from `grays`. If the `edges` array contains recurrences of the same edge, each is inset within the previous edge.

NSDrawWhiteBezel()

```
void NSDrawWhiteBezel(NSRect aRect, NSRect clipRect)
```

Draws a white rectangle with a bezeled border. Only the area that intersects `clipRect` is drawn.

NSFrameRect()

```
void NSFrameRect(NSRect aRect)
```

Draws a frame of width 1.0 around the inside of `aRect`, using the current color.

NSFrameRectWithWidth()

```
void NSFrameRectWithWidth(NSRect aRect, float frameWidth)
```

Draws a frame of width `frameWidth` around the inside of `aRect`, using the current color.

Color Functions

Get Information About Color Space and Window Depth

NSAvailableWindowDepths()

```
const NSWindowDepth *NSAvailableWindowDepths(void)
```

Returns a zero-terminated list of available window depths. Available window depths are

- `NSTwoBitGrayDepth`
- `NSEightBitGrayDepth`
- `NSEightBitRGBDepth`
- `NSTwelveBitRGBDepth`
- `NSTwentyFourBitRGBDepth`

NSBestDepth()

```
NSWindowDepth NSBestDepth(NSString *colorSpace, int bitsPerSample,
    int bitsPerPixel, BOOL planar, BOOL *exactMatch)
```

Returns a window depth deep enough for the given number of colors, bits per sample, bits per pixel, and if planar. Upon return, the variable pointed to by `exactMatch` is `YES` if the window depth can accommodate all of the values given for all of the parameters, and `NO` if not. See also `NSAvailableWindowDepths()`.

NSBitsPerPixelFromDepth()

```
int NSBitsPerPixelFromDepth(NSWindowDepth depth)
```

Returns the number of bits per pixel for the given window `depth`. See `NSAvailableWindowDepths()` for a list of available window depths.

NSBitsPerSampleFromDepth()

```
int NSBitsPerSampleFromDepth(NSWindowDepth depth)
```

Returns the number of bits per sample (bits per pixel in each color component) for the given window `depth`. See `NSAvailableWindowDepths()` for a list of window depths.

NSColorSpaceFromDepth()

```
NSString *NSColorSpaceFromDepth(NSWindowDepth depth)
```

Returns the name of the color space that matches the given window `depth`. See the Graphics section of the Application Kit's Types and Constants chapter for more information on color-space names. See `NSAvailableWindowDepths()` for a list of available window depths.

NSNumberOfColorComponents()

```
int NSNumberOfColorComponents(NSString *colorSpaceName)
```

Returns the number of color components in the named color space. The return value will be 1 for `NSCalibratedWhiteColorSpace`, `NSCalibratedBlackColorSpace`, `NSDeviceWhiteColorSpace`, and `NSDeviceBlackColorSpace`; the return value will be 3 for `NSCalibratedRGBColorSpace`, and `NSDeviceRGBColorSpace`; the return value will be 4 for `NSDeviceCMYKColorSpace`; and the return value will be 0 for a incorrect `colorSpaceName`.

NSPlanarFromDepth()

```
BOOL NSPlanarFromDepth(NSWindowDepth depth)
```

Returns `YES` if the given window depth is planar, `NO` if not.

Read the Color at a Screen Position

NSReadPixel()

```
NSColor *NSReadPixel(NSPoint location)
```

Returns the color of the pixel at the given `location`, which must be specified in the current view's coordinate system.

Text Functions

Filter Characters Entered into a Text Object

`NSEditorFilter()`

```
unsigned short NSEditorFilter(unsigned short theChar, int flags,
    NSStringEncoding theEncoding)
```

Identical to `NSFieldFilter()` except that it passes on values corresponding to Return, Tab, and Shift-Tab directly to the `NSText` object.

`NSFieldFilter()`

```
unsigned short NSFieldFilter(unsigned short theChar, int flags,
    NSStringEncoding theEncoding)
```

Checks each character the user types into an `NSText` object's text, allowing the user to move the selection among text fields by pressing Return, Tab, or Shift-Tab. Alphanumeric characters are passed to the `NSText` object for display. The function returns either the ASCII value of the character typed, 0 (for illegal characters or ones entered while a Command key is held down), or a constant that the Text object interprets as a movement command. See also `NSEditorFilter()`.

Calculate or Draw a Line of Text (in Text Object)

`NSDrawALine()`

```
int NSDrawALine(id self, NSLayInfo *layInfo)
```

Draws a line of text, using the global variables set by `NSScanALine()`. The return value has no significance.

NSScanALine()

```
int NSScanALine(id self, NSLayInfo *layInfo)
```

Determines the placement of characters in a line of text. `self` refers to the text object calling the function, and `layInfo` contains the line information. The function returns 1 if a word's length exceeds the width of a line and the text object's `charWrap` method returns `NO`. Otherwise, it returns 0. See the Text section of the Application Kit's Types and Constants chapter for the `NSLayInfo` definition.

Calculate Font Ascender, Descender, and Line Height (in Text Object)

NSTextFontInfo()

```
void NSTextFontInfo(id fid, float *ascender, float *descender,
    float *lineHeight)
```

Calculates, and returns by reference, the ascender, descender, and line height values for the font object given by `font`.

Access Text Object's Word Tables

NSDataWithWordTable()

```
NSData * NSDataWithWordTable(const unsigned char *smartLeft
    const unsigned char *smartRight,
    const unsigned char *charClasses, const NSFSM *wrapBreaks,
    int wrapBreaksCount, const NSFSM *clickBreaks,
    int clickBreaksCount, BOOL charWrap)
```

Given pointers to word table structures, records the structures in the returned `NSData` object. The arguments are similar to those of `NSReadWordTable()`.

NSReadWordTable()

```
void NSReadWordTable(NSZone *zone, NSData *data,
    unsigned char **smartLeft, unsigned char **smartRight,
    unsigned char **charClasses, NSFSM **wrapBreaks,
    int *wrapBreaksCount, NSFSM **clickBreaks,
    int *clickBreaksCount, BOOL *charWrap)
```

Given `data`, creates word tables in the memory zone specified by `zone`, returning (in the subsequent arguments) pointers to the various tables. The integer pointer arguments return the length of the preceding array, and `charWrap` indicates whether words whose length exceeds the text object's line length should be wrapped on a character-by-character basis.

Array Allocation Functions for Use by the NSText Class

NSChunkCopy()

```
NSTextChunk *NSChunkCopy(NSTextChunk *pc, NSTextChunk *dpc)
```

Copies the array `pc` to the array `dpc` and returns a pointer to the copy. See the Text section of the Application Kit's Types and Constants chapter for a description of the `NSTextChunk` structure. See also `NSChunkZoneCopy()`.

NSChunkGrow()

```
NSTextChunk *NSChunkGrow(NSTextChunk *pc, int newUsed)
```

Increases the array identified by the pointer `pc` to a size of `newUsed` bytes. See the Text section of the Application Kit's Types and Constants chapter for a description of the `NSTextChunk` structure. See also `NSChunkZoneGrow()`.

NSChunkMalloc()

```
NSTextChunk *NSChunkMalloc(int growBy, int initUsed)
```

Allocates initial memory for a structure whose first field is an `NSTextChunk` structure and whose subsequent field is a variable-sized array. The amount of memory allocated is equal to `initUsed`. If `initUsed` is 0, `growBy` bytes are allocated. `growBy` specifies how much memory should be allocated when the chunk grows. See the Text section of the Application Kit's Types and Constants chapter for a description of the `NSTextChunk` structure. See also `NSChunkRealloc()`, `NSChunkZoneMalloc()`.

NSChunkRealloc()

```
NSTextChunk *NSChunkRealloc(NSTextChunk *pc)
```

Increases the amount of memory available for the array identified by the pointer pc, as determined by the array's NSTextChunk. See the Text section of the Application Kit's Types and Constants chapter for a description of the NSTextChunk structure. See also NSChunkZoneRealloc().

NSChunkZoneCopy()

```
NSTextChunk *NSChunkZoneCopy(NSTextChunk *pc, NSTextChunk *dpc,
    NSZone *zone)
```

Similar to NSChunkCopy(), but uses the specified zone of memory.

NSChunkZoneGrow()

```
NSTextChunk *NSChunkZoneGrow(NSTextChunk *pc, int newUsed,
    NSZone *zone)
```

Similar to NSChunkGrow(), but uses the specified zone of memory.

NSChunkZoneMalloc()

```
NSTextChunk *NSChunkZoneMalloc(int growBy, int initUsed,
    NSZone *zone)
```

Similar to NSChunkMalloc(), but uses the specified zone of memory.

NSChunkZoneRealloc()

```
NSTextChunk *NSChunkZoneRealloc(NSTextChunk *pc, NSZone *zone)
```

Similar NSChunkRealloc(), but uses the specified zone of memory.

3

Imaging Functions

Copy an Image

NSCopyBitmapFromGState()

```
void NSCopyBitmapFromGState(int srcGstate, NSRect srcRect,
    NSRect destRect)
```

Copies the pixels in the rectangle `srcRect` to the rectangle `destRect`. The source rectangle is defined in the graphics state designated by `srcGstate`, and the destination is defined in the current graphics state.

NSCopyBits()

```
void NSCopyBits(int srcGstate, NSRect srcRect, NSPoint destPoint)
```

Copies the pixels in the rectangle `srcRect` to the location `destPoint`. The source rectangle is defined in the current graphics state if `srcGstate` is `NSNullObject`; otherwise, in the graphics state designated by `srcGstate`. The `destPoint` destination is defined in the current graphics state.

Render Bitmap Images

NSDrawBitmap()

```
void NSDrawBitmap(NSRect rect, int pixelsWide, int pixelsHigh,
    int bitsPerSample, int samplesPerPixel, int bitsPerPixel,
    int bytesPerRow, BOOL isPlanar, BOOL hasAlpha,
    NSString *colorSpaceName, const unsigned char *const data[5])
```

Renders an image from a bitmap. `rect` is the rectangle in which the image is drawn, and `data` is the bitmap data, stored in up to 5 channels unless `isPlanar` is `NO` (in which case the channels are interleaved in a single array).

Attention Panel Functions

Create an Attention Panel without Running It Yet

NSGetAlertPanel()

```
id NSGetAlertPanel(NSString *title, NSString *msg,
    NSString *defaultButton, NSString *alternateButton,
    NSString *otherButton, ...)
```

Returns an `NSPanel` object that you can use in a modal session. Unlike `NSRunAlertPanel()`, no button is displayed if `defaultButton` is `NULL`. `NSGetAlertPanel()` doesn't set up a modal event loop; instead, it returns a panel that can be used to set up a modal session. A modal session is useful for allowing the user to interrupt the program. During a modal session, you can perform activities while the panel is displayed and check at various points in your program whether the user has clicked one of the panel's buttons. See also `NSRunAlertPanel()`.

Create and Run an Attention Panel

NSRunAlertPanel()

```
int NSRunAlertPanel(NSString *title, NSString *msg,
    NSString *defaultButton, NSString *alternateButton,
    NSString *otherButton, ...)
```

Creates an attention panel that alerts the user to some consequence of a requested action, and runs the panel in a modal event loop. `title` is the panel's title (by default, "Alert"); `msg` is the `printf()`-style message that's displayed in the panel; `defaultButton` (by default, "OK") is the title for the main button, also activated by the Return button; `alternateButton` and `otherButton` give two more choices, which are displayed only if the corresponding argument isn't `NULL`. The trailing arguments are a variable number of `printf()`-style arguments to `msg`. Return values are either `NSAlertDefaultReturn`, `NSAlertAlternateReturn`, `NSAlertOtherReturn`, depending on which button is pushed; or `NSAlertErrorReturn` if an error occurs when creating the panel.

Release an Attention Panel

NSReleaseAlertPanel()

```
void NSReleaseAlertPanel(id panel)
```

Releases the specified alert panel. See also `NSGetAlertPanel()`.

Services Menu Functions

Registering Services Provider Applications

NSRegisterServicesProvider()

```
void NSRegisterServicesProvider(id provider, NSString *name)
```

Registers the given lightweight (that is, does not create an `NSApplication` object) services `provider`, under the given `name`. See also `NSUnregisterServicesProvider()`, `servicesProvider` (`NSApplication`).

NSUnregisterServicesProvider()

```
void NSUnregisterServicesProvider(NSString *name)
```

Unregisters the lightweight (that is, does not create an `NSApplication` object) services provider under the given `name`. See also `NSRegisterServicesProvider()`, `servicesProvider` (`NSApplication`).

Determine Whether an Item Is Included in Services Menus

NSSetShowsServicesMenuItem()

```
int NSSetShowsServicesMenuItem(NSString *item, BOOL showService)
```

Determines (based on the value of `showService`) whether the `item` command will be included in other applications' Services menus. `item` describes a service provided by this application, and should be the same string entered in the "Menu Item:" field of the services file. This function returns 0 upon success.

NSShowsServicesMenuItem()

```
BOOL NSShowsServicesMenuItem(NSString *item)
```

Returns `YES` if `item` is currently shown in Services menus.

Programmatically Invoke a Service

NSPerformService()

```
BOOL NSPerformService(NSString *item, NSPasteboard *pboard)
```

Invokes a service found in the application's Services menu. `item` is the name of a Services menu item, in any language; a slash in this name represents a submenu. `pboard` must contain the data required by the service, and when the function returns, `pboard` will contain the data supplied by the service provider.

Force Services Menu to Update Based on New Services

NSUpdateDynamicServices()

```
void NSUpdateDynamicServices(void)
```

Re-registers the services the application is willing to provide, by reading the file with the extension `.service` in the application path or in the standard path for services.

X-Windows Convenience Functions

This section lists convenience functions used to access window system facilities when running OpenStep in the X11 environment. In general these routines are designed to duplicate functions that Openstep programmers might have used when programming OpenStep applications on other window system platforms. Note that these routines do not represent elements of the OpenStep specification.

NSWindowCurrentMouse()

```
void NSWindowCurrentMouse(NSWindow* self, float* rx, float* ry)
```

NSWindowStillDown()

```
BOOL NSWindowStillDown(NSWindow* self, NSEvent* nsEvent)
```

NSSetWindowLevel()

```
void NSSetWindowLevel(NSWindow* self, unsigned int level)
```

NSMouseScreenLocation()

```
NSPoint NSMouseScreenLocation(NSWindow* window)
```

NSHideAppsExcept()

```
void NSHideAppsExcept(unsigned long ctxtid)
```

NSActivateContextNumber()

```
void NSActivateContextNumber(unsigned long ctxtid)
```

NSActivateNextApp()

```
void NSActivateNextApp()
```

Other Application Kit Functions

Application Main Function

NSApplicationMain()

```
void NSApplicationMain(int argc, char *argv[])
```

The main function for OpenStep applications. Called from within `main()`. For example:

```
void main(int argc, char *argv[]) {
    NSApplicationMain(argc, argv);
}
```

:creates an autorelease pool, loads the main nib file, finds the principal class, and creates the shared appliction.

Play the System Beep

NSBeep()

```
void NSBeep(void)
```

Plays the system beep.

Return File-related Pasteboard Types

NSCreateFileContentsPboardType()

```
NSString *NSCreateFileContentsPboardType(NSString *fileType)
```

Returns a string naming a pasteboard type that represents a file's contents, based on the supplied string `fileType`. `fileType` should generally be the extension part of a file name. The conversion from a named file type to a pasteboard type is simple; no mapping to standard pasteboard types is attempted.

NSCreateFilenamePboardType()

```
NSString *NSCreateFilenamePboardType(NSString *filename)
```

Returns a string naming a pasteboard type that represents a a file name, based on the supplied string `filename`.

NSGetFileType()

```
NSString *NSGetFileType(NSString *pboardType)
```

Returns the extension or file name from which the pasteboard type `pboardType` was derived. `nil` is returned if `pboardType` isn't a pasteboard type created by `NSCreateFileContentsPboardType()` or `NSCreateFilenamePboardType()`.

NSGetFileTypes()

```
NSArray *NSGetFileTypes(NSArray *pboardTypes)
```

Accepts an array of pasteboard types and returns an array of the unique extensions and file names from the file-content and file-name types found in `pboardTypes`. It returns `nil` if the input array contains no file-content or file-name types.

Draw a Distinctive Outline Around Linked Data

NSFrameLinkRect()

```
void NSFrameLinkRect(NSRect aRect, BOOL isDestination)
```

Draws a distinctive link outline just outside the rectangle `aRect`. To draw an outline around a destination link, `isDestination` should be `YES`, otherwise `NO`.

NSLinkFrameThickness()

```
float NSLinkFrameThickness(void)
```

Returns the thickness of the link outline so that the outline can be properly erased by the application, or for other purposes.

Convert an Event Mask Type to a Mask

NSEventMaskFromType()

```
unsigned int NSEventMaskFromType(NSEventType type)
```

Returns the event mask corresponding to `type` (an enumeration constant). The returned mask equals 1 left-shifted by `type` bits. See the Event Handling section of the Application Kit's Types and Constants chapter for a list of `NSEventType` enumeration constants.

Types and Constants 4

Applications

Application Instance Identifier

```
id NSApp;
```

Represents the application's `NSApplication` object.

Modal Session Information

```
typedef struct _NSModalSession *NSModalSession;
```

This structure stores information used by the system during a modal session.

Run Loop Status

```
enum {
    NSRunStoppedResponse,
    NSRunAbortedResponse,
    NSRunContinuesResponse
};
```

Predefined return values for `runModalFor:` and `runModalSession:`.

Run Loop Modes

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

Input-filter modes passed to NSRunLoop.

Boxes

Box Title Position

```
typedef enum _NSTitlePosition {
    NSNoTitle,
    NSAboveTop,
    NSAtTop,
    NSBelowTop,
    NSAboveBottom,
    NSAtBottom,
    NSBelowBottom
} NSTitlePosition;
```

This type's constants represent the locations where an `NSBox`'s title is placed in relation to the border (`setTitlePosition:` and `titlePosition`).

Buttons

Button Types

```
typedef enum _NSButtonType {
    NSMomentaryPushButton,
    NSPushOnPushOffButton,
    NSToggleButton,
    NSSwitchButton,
    NSRadioButton,
    NSMomentaryChangeButton,
    NSOnOffButton,
    NSMomentaryLightButton
} NSButtonType;
```

These constants indicate the way `NSButton`s and `NSButtonCell`s behave when pressed, and how they display their state. They are used by `NSButton`'s `setType:` method.

Cells and Button Cells

Cell Types

```
typedef enum _NSCellType {
    NSNullCellType,
    NSTextCellType,
    NSImageCellType
} NSCellType;
```

Represent different types of `NSCell` objects. `NSNullCellType` means the cell does not display. `NSTextCellType` displays text, and `NSImageCellType` displays an image.These values are set and returned by `NSCell`'s `setType:` and `type` methods.

Cell Image Position

```
typedef enum _NSCellImagePosition {
    NSNoImage,
    NSImageOnly,
    NSImageLeft,
    NSImageRight,
    NSImageBelow,
    NSImageAbove,
    NSImageOverlaps
} NSCellImagePosition;
```

Represent the position of an `NSButtonCell` relative to its title. These values are returned by `NSButtonCell`'s `imagePosition` and `setImagePosition:` methods.

Cell Attributes

```
typedef enum _NSCellAttribute {
    NSCellDisabled,
    NSCellState,
    NSPushInCell,
    NSCellEditable,
    NSChangeGrayCell,
    NSCellHighlighted,
    NSCellLightsByContents,
    NSCellLightsByGray,
    NSChangeBackgroundCell,
```

```
    NSCellLightsByBackground,
    NSCellIsBordered,
    NSCellHasOverlappingImage,
    NSCellHasImageHorizontal,
    NSCellHasImageOnLeftOrBottom,
    NSCellChangesContents,
    NSCellIsInsetButton
} NSCellAttribute;
```

These constant values represent parameters that you can set and access through `NSCell`'s and `NSButtonCell`'s `setCellAttribute:to:` and `cellAttribute:` methods. Only the first five constants are used by `NSCell`; the others apply to `NSButtonCell`s only.

Cell Entry Types

```
enum {
    NSAnyType,
    NSIntType,
    NSPositiveIntType,
    NSFloatType,
    NSPositiveFloatType,
    NSDoubleType,
    NSPositiveDoubleType
};
```

These constants represent numeric types that an `NSCell` can accept from the user. These values are set and returned by `NSCell`'s `setEntryType:` and `entryType` methods.

Button Cell Masks

```
enum {
    NSNoCellMask,
    NSContentsCellMask,
    NSPushInCellMask,
    NSChangeGrayCellMask,
    NSChangeBackgroundCellMask
};
```

`NSButtonCell` uses these values to determine how to highlight a button cell or show an ON state. These values are used by `NSButtonCell`'s `showsStateBy`, `setShowsStateBy:`, `highlightsBy`, and `setHighlightsBy:` methods.

Colors

Color Panel Modes

```
enum {
    NSGrayModeColorPanel,
    NSRGBModeColorPanel,
    NSCMYKModeColorPanel,
    NSHSBModeColorPanel,
    NSCustomPaletteModeColorPanel,
    NSColorListModeColorPanel,
    NSWheelModeColorPanel
};
```

These constants are tags that identify mode (or views) in the color panel.

Color Panel Mode Masks

```
enum {
    NSColorPanelGrayModeMask,
    NSColorPanelRGBModeMask,
    NSColorPanelCMYKModeMask,
    NSColorPanelHSBModeMask,
    NSColorPanelCustomPaletteModeMask,
    NSColorPanelColorListModeMask,
    NSColorPanelWheelModeMask,
    NSColorPanelAllModesMask
};
```

These bit masks determine the current mode (or view) of the color panel.

Data Links

Note that these data link types are not part of the OpenStep specification.

Data Link Number

```
typedef int NSDataLinkNumber;
```

Returned by `NSDataLink`'s `linkNumber` method as a persistent identifier of a destination link.

Data Link Disposition

```
typedef enum _NSDataLinkDisposition {
     NSLinkInDestination,
     NSLinkInSource,
     NSLinkBroken
} NSDataLinkDisposition;
```

Returned by `NSDataLink`'s disposition method to identify a link as a destination link, a source link, or a broken link.

Data Link Update Mode

```
typedef enum _NSDataLinkUpdateMode {
    NSUpdateContinuously,
    NSUpdateWhenSourceSaved,
    NSUpdateManually,
    NSUpdateNever
} NSDataLinkUpdateMode;
```

Identifies when a link's data is to be updated. Set by `NSDataLink`'s `setUpdateMode:` method, and returned by the `updateMode` method.

Drag Operations

Drag Operations

```
typedef enum _NSDragOperation {
    NSDragOperationNone,
    NSDragOperationCopy,
    NSDragOperationLink,
```

```
    NSDragOperationGeneric,
    NSDragOperationPrivate,
    NSDragOperationAll
};
```

These constants identify different kinds of dragging operations. The following table gives each constants meaning.

Drag Operation	Meaning
`NSDragOperationNone`	No operation possible (rejection)
`NSDragOperationCopy`	The data represented by the image can be copied
`NSDragOperationLink`	The data can be shared
`NSDragOperationGeneric`	The operation can be defined by the destination
`NSDragOperationPrivate`	Private source/destination negotiation. The system leaves the cursor alone until exit.
`NSDragOperationAll`	Combines all the above

Event Handling

Event Types

```
typedef enum _NSEventType {
    NSLeftMouseDown,
    NSLeftMouseUp,
    NSRightMouseDown,
    NSRightMouseUp,
    NSMouseMoved,
    NSLeftMouseDragged,
    NSRightMouseDragged,
    NSMouseEntered,
    NSMouseExited,
    NSKeyDown,
    NSKeyUp,
    NSFlagsChanged,
    NSPeriodic,
    NSCursorUpdate
} NSEventType;
```

Each constant of `NSEventType` identifies an event type. See the `NSEvent` class for more information.

Function Key Codes

```
enum {
    NSUpArrowFunctionKey = 0xF700,
    NSDownArrowFunctionKey = 0xF701,
    NSLeftArrowFunctionKey = 0xF702,
    NSRightArrowFunctionKey = 0xF703,
    NSF1FunctionKey  = 0xF704,
    NSF2FunctionKey  = 0xF705,
    NSF3FunctionKey  = 0xF706,
    NSF4FunctionKey  = 0xF707,
    NSF5FunctionKey  = 0xF708,
    NSF6FunctionKey  = 0xF709,
    NSF7FunctionKey  = 0xF70A,
    NSF8FunctionKey  = 0xF70B,
    NSF9FunctionKey  = 0xF70C,
    NSF10FunctionKey = 0xF70D,
    NSF11FunctionKey = 0xF70E,
    NSF12FunctionKey = 0xF70F,
    NSF13FunctionKey = 0xF710,
    NSF14FunctionKey = 0xF711,
    NSF15FunctionKey = 0xF712,
    NSF16FunctionKey = 0xF713,
    NSF17FunctionKey = 0xF714,
    NSF18FunctionKey = 0xF715,
    NSF19FunctionKey = 0xF716,
    NSF20FunctionKey = 0xF717,
    NSF21FunctionKey = 0xF718,
    NSF22FunctionKey = 0xF719,
    NSF23FunctionKey = 0xF71A,
    NSF24FunctionKey = 0xF71B,
    NSF25FunctionKey = 0xF71C,
    NSF26FunctionKey = 0xF71D,
    NSF27FunctionKey = 0xF71E,
    NSF28FunctionKey = 0xF71F,
    NSF29FunctionKey = 0xF720,
    NSF30FunctionKey = 0xF721,
    NSF31FunctionKey = 0xF722,
    NSF32FunctionKey = 0xF723,
    NSF33FunctionKey = 0xF724,
    NSF34FunctionKey = 0xF725,
    NSF35FunctionKey = 0xF726,
    NSInsertFunctionKey = 0xF727,
    NSDeleteFunctionKey = 0xF728,
    NSHomeFunctionKey = 0xF729,
    NSBeginFunctionKey = 0xF72A,
```

```
    NSEndFunctionKey = 0xF72B,
    NSPageUpFunctionKey = 0xF72C,
    NSPageDownFunctionKey = 0xF72D,
    NSPrintScreenFunctionKey = 0xF72E,
    NSScrollLockFunctionKey = 0xF72F,
    NSPauseFunctionKey = 0xF730,
    NSSysReqFunctionKey = 0xF731,
    NSBreakFunctionKey = 0xF732,
    NSResetFunctionKey = 0xF733,
    NSStopFunctionKey = 0xF734,
    NSMenuFunctionKey = 0xF735,
    NSUserFunctionKey = 0xF736,
    NSSystemFunctionKey = 0xF737,
    NSPrintFunctionKey = 0xF738,
    NSClearLineFunctionKey = 0xF739,
    NSClearDisplayFunctionKey = 0xF73A,
    NSInsertLineFunctionKey = 0xF73B,
    NSDeleteLineFunctionKey = 0xF73C,
    NSInsertCharFunctionKey = 0xF73D,
    NSDeleteCharFunctionKey = 0xF73E,
    NSPrevFunctionKey = 0xF73F,
    NSNextFunctionKey = 0xF740,
    NSSelectFunctionKey = 0xF741,
    NSExecuteFunctionKey = 0xF742,
    NSUndoFunctionKey = 0xF743,
    NSRedoFunctionKey = 0xF744,
    NSFindFunctionKey = 0xF745,
    NSHelpFunctionKey = 0xF746,
    NSModeSwitchFunctionKey = 0xF747
};
```

Unicodes that identify function keys on the keyboard. OpenStep reserves the range 0xF700-0xF8FF for this purpose. The availability of some keys is system-dependent.

Function Key Mask

```
enum {
    NSAlphaShiftKeyMask,
    NSShiftKeyMask,
    NSControlKeyMask,
    NSAlternateKeyMask,
    NSCommandKeyMask,
```

```
    NSNumericPadKeyMask,
    NSHelpKeyMask,
    NSFunctionKeyMask
};
```

Device-independent bit masks for evaluating event-modifier flags to determine which modifier key (if any) was pressed.

Event Masks

```
enum {
    NSLeftMouseDownMask,
    NSLeftMouseUpMask,
    NSRightMouseDownMask,
    NSRightMouseUpMask,
    NSMouseMovedMask,
    NSLeftMouseDraggedMask,
    NSRightMouseDraggedMask,
    NSMouseEnteredMask,
    NSMouseExitedMask,
    NSKeyDownMask,
    NSKeyUpMask,
    NSFlagsChangedMask,
    NSPeriodicMask,
    NSCursorUpdateMask,
    NSAnyEventMask
};
```

Bit masks for determining event types.

Exceptions

Global Exception Strings

```
NSString *NSAbortModalException;
NSString *NSAbortPrintingException;
NSString *NSAppKitIgnoredException;
NSString *NSAppKitVirtualMemoryException;
NSString *NSBadBitmapParametersException;
NSString *NSBadComparisonException;
NSString *NSBadRTFColorTableException;
NSString *NSBadRTFDirectiveException;
NSString *NSBadRTFFontTableException;
```

```
NSString *NSBadRTFStyleSheetException;
NSString *NSBrowserIllegalDelegateException;
NSString *NSColorListIOException;
NSString *NSColorListNotEditableException;
NSString *NSDraggingException;
NSString *NSFontUnavailableException;
NSString *NSIllegalSelectorException;
NSString *NSImageCacheException;
NSString *NSNibLoadingException;
NSString *NSPPDIncludeNotFoundException;
NSString *NSPPDIncludeStackOverflowException;
NSString *NSPPDIncludeStackUnderflowException;
NSString *NSPPDParseException;
NSString *NSPasteboardCommunicationException;
NSString *NSPrintOperationExistsException;/*NSPrintOperation.h */
NSString *NSPrintPackageException;
NSString *NSPrintingCommunicationException;
NSString *NSRTFPropertyStackOverflowException;
NSString *NSTIFFException;
NSString *NSTextLineTooLongException;
NSString *NSTextNoSelectionException;
NSString *NSTextReadException;
NSString *NSTextWriteException;
NSString *NSTypedStreamVersionException;
NSString *NSWindowServerCommunicationException;
NSString *NSWordTablesReadException;
NSString *NSWordTablesWriteException;
```

These global strings identify the exceptions returned by various operations in the Application Kit. They are defined in `NSErrors.h`.

Fonts

Font Trait Masks

```
typedef unsigned int NSFontTraitMask;
```

Characterizes one or more of a font's traits. It's used as an argument type for several of the methods in the `NSFontManager` class. You build a mask by OR'ing together the following enumeration constants:

```
enum {
    NSItalicFontMask,
    NSBoldFontMask,
```

```
    NSUnboldFontMask,
    NSNonStandardCharacterSetFontMask,
    NSNarrowFontMask,
    NSExpandedFontMask,
    NSCondensedFontMask,
    NSSmallCapsFontMask,
    NSPosterFontMask,
    NSCompressedFontMask,
    NSUnitalicFontMask
   NSFixedPitchFontMask
};
```

These values are used by `NSFontManager` to identify font traits.

Glyphs

```
typedef unsigned int NSGlyph;
```

A type definition for numbers identifying font glyphs. It's used as the argument type for several of the methods in `NSFont`.

Font Panel Views

```
enum {
    NSFPPreviewButton,
    NSFPRevertButton,
    NSFPSetButton,
    NSFPPreviewField,
    NSFPSizeField,
    NSFPSizeTitle,
    NSFPCurrentField
};
```

Tags identifying views in the font panel.

Font Identity Matrix

```
const float *NSFontIdentityMatrix;
```

Identifies a font matrix that's used for fonts displayed in an `NSView` object that has an unflipped coordinate system.

Font Manager Dictionary Keys

```
NSString *NSAFMAscender;
NSString *NSAFMCapHeight;
NSString *NSAFMCharacterSet;
NSString *NSAFMDescender;
NSString *NSAFMEncodingScheme;
NSString *NSAFMFamilyName;
NSString *NSAFMFontName;
NSString *NSAFMFormatVersion;
NSString *NSAFMFullName;
NSString *NSAFMItalicAngle;
NSString *NSAFMMappingScheme;
NSString *NSAFMNotice;
NSString *NSAFMUnderlinePosition;
NSString *NSAFMUnderlineThickness;
NSString *NSAFMVersion;
NSString *NSAFMWeight;
NSString *NSAFMXHeight;
```

Global keys to access the values available in the Adobe Font Manager (AFM) dictionary. You can convert the appropriate values (e.g., ascender, cap height) to floating point values by using `NSString`'s `floatValue` method.

Font Manager Tags

```
typedef enum _NSFontAction
    NSNoFontChangeAction
    NSViaPanelFontAction
    NSAddTraitFontAction
    NSSizeUpFontAction
    NSSizeDownFontAction
    NSHeavierFontAction
    NSLighterFontAction
    NSRemoveTraitFontAction
} NSFontAction
```

These tags represent font trait actions initiated by the Font Manager.

Graphics

NSWindowDepth

```
typedef int NSWindowDepth;
```

This type gives the window-depth limit. Use the `NSAvailableWindowDepths()` function to get a list of available window depths. Use the functions `NSBitsPerSampleFromDepth()`, `NSBitsPerPixelFromDepth()`, `NSPlanarFromDepth()`, and `NSColorSpaceFromDepth()` to extract information from a window depth. The `NSWindowDepth` type is also used as an argument type for methods in the `NSScreen` and `NSWindow` classes.

NSTIFFCompression

```
typedef enum _NSTIFFCompression {
    NSTIFFCompressionNone  = 1,
    NSTIFFCompressionCCITTFAX3  = 3,
    NSTIFFCompressionCCITTFAX4  = 4,
    NSTIFFCompressionLZW  = 5,
    NSTIFFCompressionJPEG  = 6,
    NSTIFFCompressionNEXT  = 32766,
    NSTIFFCompressionPackBits  = 32773,
    NSTIFFCompressionOldJPEG  = 32865
} NSTIFFCompression;
```

The constants defined in this type represent the various TIFF (tag image file format) data compression schemes. They are defined in the `NSBitMapImageRep` class and used in several methods of that class as well as in the TIFFRepresentationUsingCompression:factor: method of `NSImage`.

Device Matching

```
enum {
    NSImageRepMatchesDevice
};
```

`NSImageRepMatchesDevice` indicates that the value varies according to the output device. It can be passed in (or received back) as the value of `NSImageRep`'s `bitsPerSample`, `pixelsWide`, and `pixelsHigh`.

Colorspace Names

```
NSString *NSCalibratedWhiteColorSpace;
NSString *NSCalibratedBlackColorSpace;
NSString *NSCalibratedRGBColorSpace;
NSString *NSDeviceWhiteColorSpace;
NSString *NSDeviceBlackColorSpace;
NSString *NSDeviceRGBColorSpace;
NSString *NSDeviceCMYKColorSpace;
NSString *NSNamedColorSpace;
NSString *NSCustomColorSpace;
```

Predefined colorspace names. These strings are used as arguments in `NSDrawBitMap()` and `NSNumberOfColorComponents()`, and are values returned from NSColorSpaceFromDepth().

Gray Values

```
const float NSBlack;
const float NSDarkGray;
const float NSWhite;
const float NSLightGray;
```

Standard gray values for the 2-bit deep grayscale colorspace.

Device Dictionary Keys

```
NSString *NSDeviceResolution;
NSString *NSDeviceColorSpaceName;
NSString *NSDeviceBitsPerSample;
NSString *NSDeviceIsScreen;
NSString *NSDeviceIsPrinter;
NSString *NSDeviceSize;
```

Keys to get designated values from device dictionaries.

Matrices

Matrix Modes

```
typedef enum _NSMatrixMode {
    NSRadioModeMatrix,
    NSHighlightModeMatrix,
```

```
    NSListModeMatrix,
    NSTrackModeMatrix
} NSMatrixMode;
```

These constants represent `NSMatrix` operation modes. See the `NSMatrix` class description for more information.

Notifications

Notifications are posted to all interested observers of a specific condition to alert them that the condition has occurred. Global strings contain the actual text of the notification. In the Application Kit, these are defined per class. See the Foundation's `NSNotification` and `NSNotificationCenter` for more information.

Application

```
NSString *NSApplicationDidBecomeActiveNotification;
NSString *NSApplicationDidFinishLaunchingNotification;
NSString *NSApplicationDidHideNotification;
NSString *NSApplicationDidResignActiveNotification;
NSString *NSApplicationDidUnhideNotification;
NSString *NSApplicationDidUpdateNotification;
NSString *NSApplicationWillBecomeActiveNotification;
NSString *NSApplicationWillFinishLaunchingNotification;
NSString *NSApplicationWillHideNotification;
NSString *NSApplicationWillResignActiveNotification;
NSString *NSApplicationWillTerminateNotification
NSString *NSApplicationWillUnhideNotification;
NSString *NSApplicationWillUpdateNotification;
```

Color List

```
NSString *NSColorListDidChangNotification;
```

Color Panel

```
NSString *NSColorPanelColorDidChangeNotification;
```

Controls

```
NSString *NSControlTextDidBeginEditingNotification;
NSString *NSControlTextDidEndEditingNotification;
NSString *NSControlTextDidChangeNotification;
```

Image Representations

```
NSString *NSImageRepRegistryDidChangeNotification;
```

Split Views

```
NSString *NSSplitViewDidResizeSubviewsNotification;
NSString *NSSplitViewWillResizeSubviewsNotification;
```

Text

```
NSString *NSTextDidBeginEditingNotification;
NSString *NSTextDidEndEditingNotification;
NSString *NSTextDidChangeNotification;
```

Views

```
/* NSViewBoundsDidChangeNotification is sent whenever the views
bounds change and the frame does not. That is, it is sent whenever
the view's bounds are translated, scaled or rotated, but NOT when
the bounds change as a result of, for example, setFrameSize:. */
NSString *NSViewBoundsDidChangeNotification
NSString *NSViewFrameDidChangeNotification;
NSString *NSViewFocusDidChangeNotification;
```

Windows

```
NSString *NSWindowDidBecomeKeyNotification;
NSString *NSWindowDidBecomeMainNotification;
NSString *NSWindowDidChangeScreenNotification;
NSString *NSWindowDidDeminiaturizeNotification;
NSString *NSWindowDidExposeNotification;
NSString *NSWindowDidMiniaturizeNotification;
NSString *NSWindowDidMoveNotification;
NSString *NSWindowDidResignKeyNotification;
NSString *NSWindowDidResignMainNotification;
```

```
NSString *NSWindowDidResizeNotification;
NSString *NSWindowDidUpdateNotification;
NSString *NSWindowWillCloseNotification;
```

Workspace

```
NSString *NSWorkspaceDidLaunchApplicationNotification;
NSString *NSWorkspaceDidMountNotification;
NSString *NSWorkspaceDidPerformFileOperationNotification;
NSString *NSWorkspaceDidTerminateApplicationNotification;
NSString *NSWorkspaceDidUnmountNotification;
NSString *NSWorkspaceWillLaunchApplicationNotification;
NSString *NSWorkspaceWillPowerOffNotification;
NSString *NSWorkspaceWillUnmountNotification;
```

Panels

Panel Buttons

```
enum {
    NSOKButton = 1,
    NSCancelButton = 0
};
```

Values returned by the standard panel buttons, OK and Cancel.

Alert Panel

```
enum {
    NSAlertDefaultReturn = 1,
    NSAlertAlternateReturn = 0,
    NSAlertOtherReturn = -1,
    NSAlertErrorReturn  = -2
};
```

Values returned by the `NSRunAlertPanel()` function and by `runModalSession:` when the modal session is run with a panel provided by `NSGetAlertPanel()`.

Page Layouts

```
enum {
    NSPLImageButton,
    NSPLTitleField,
    NSPLPaperNameButton,
    NSPLUnitsButton,
    NSPLWidthForm,
    NSPLHeightForm,
    NSPLOrientationMatrix,
    NSPLCancelButton,
    NSPLOKButton
};
```

Tags that identify buttons, fields, and other views of the Page Layout panel. Note that these tags are not part of the OpenStep specification.

Pasteboards

Pasteboard Type Globals

```
NSString *NSStringPboardType;
NSString *NSColorPboardType;
NSString *NSFileContentsPboardType;
NSString *NSFilenamesPboardType;
NSString *NSFontPboardType;
NSString *NSRulerPboardType;
NSString *NSPostScriptPboardType;
NSString *NSTabularTextPboardType;
NSString *NSRTFPboardType;
NSString *NSTIFFPboardType;
NSString *NSDataLinkPboardType; //Defined in NSDataLink.h
NSString *NSSelectionPboardType; //Defined in NSSelection.h
```

Identifies the standard pasteboard types. These are used in a variety of `NSPasteboard` methods and functions.

Pasteboard Name Globals

```
NSString *NSDragPboard;
NSString *NSFindPboard;
NSString *NSFontPboard;
NSString *NSGeneralPboard;
NSString *NSRulerPboard;
```

Identifies the standard pasteboard names. Used in class method `pasteboardWithName:` to get a pasteboard by name.

Printing

Print Table Status

```
typedef enum _NSPrinterTableStatus {
    NSPrinterTableOK,
    NSPrinterTableNotFound,
    NSPrinterTableError
} NSPrinterTableStatus;
```

These constants describe the state of a printer-information table stored by an `NSPrinter` object. It is the argument type of the return value of `statusForTable:`.

Page Orientation

```
typedef enum _NSPrintingOrientation {
    NSPortraitOrientation,
    NSLandscapeOrientation
} NSPrintingOrientation;
```

These constants represent the way a page is oriented for printing.

Page Order

```
typedef enum _NSPrintingPageOrder {
    NSDescendingPageOrder,
    NSSpecialPageOrder,
    NSAscendingPageOrder,
    NSUnknownPageOrder
} NSPrintingPageOrder;
```

These constants describe the order in which pages are pooled for printing. `NSSpecialPageOrder` tells the spooler not to rearrange pages. Set through `NSPrintOperation`'s `setPageOrder:` method and returned by its `pageOrder` method.

Pagination Mode

```
typedef enum _NSPrintingPaginationMode {
    NSAutoPagination,
    NSFitPagination,
    NSClipPagination
} NSPrintingPaginationMode;
```

These constants represent the different ways an image is divided into pages during pagination. Pagination can occur automatically, the image can be forced onto a page, or it can be clipped to a page.

Print Panel Layout

```
enum {
    NSPPSaveButton,
    NSPPPreviewButton,
    NSFaxButton,
    NSPPTitleField,
    NSPPImageButton,
    NSPPNameTitle,
    NSPPNameField,
    NSPPNoteTitle,
    NSPPNoteField,
    NSPPStatusTitle,
    NSPPStatusField,
    NSPPCopiesField,
    NSPPPageChoiceMatrix,
    NSPPPageRangeFrom,
    NSPPPageRangeTo,
    NSPPScaleField,
    NSPPOptionsButton,
    NSPPPaperFeedButton,
    NSPPLayoutButton
};
```

Tags that identify text fields, controls, and other views in the Print panel.

Printing Information Dictionary Keys

```
NSString *NSPrintAllPages;
NSString *NSPrintBottomMargin;
NSString *NSPrintCopies;
NSString *NSPrintFirstPage;
NSString *NSPrintHorizonalPagination;
NSString *NSPrintHorizontallyCentered;
NSString *NSPrintJobDisposition;
NSString *NSPrintJobFeatures;
NSString *NSPrintLastPage;
NSString *NSPrintLeftMargin;
NSString *NSPrintManualFeed;
NSString *NSPrintOrientation;
NSString *NSPrintPackageException;
NSString *NSPrintPagesPerSheet;
NSString *NSPrintPaperFeed;
NSString *NSPrintPaperName;
NSString *NSPrintPaperSize;
NSString *NSPrintPrinter;
NSString *NSPrintReversePageOrder;
NSString *NSPrintRightMargin;
NSString *NSPrintSavePath;
NSString *NSPrintScalingFactor;
NSString *NSPrintTopMargin;
NSString *NSPrintVerticalPagination;
NSString *NSPrintVerticallyCentered;
```

The keys in the mutable dictionary associated with `NSPrintInfo`. See `NSPrintInfo.h` for types and descriptions of values.

Print Job Disposition Values

```
NSString  *NSPrintCancelJob;
NSString  *NSPrintFaxJob;
NSString  *NSPrintPreviewJob;
NSString  *NSPrintSaveJob;
NSString  *NSPrintSpoolJob;
```

These global constants define the disposition of a print job. See `NSPrintInfo`'s `setJobDisposition:` and `jobDisposition`.

Fax Values (Platform Specific)

The following strings are not part of the OpenStep specification.

```
NSString *NSPrintFaxReceiverNames
NSString *NSPrintFaxReceiverNumbers
NSString *NSPrintFaxSendTime
NSString *NSPrintFaxUseCoverSheet
NSString *NSPrintFaxCoverSheetName
NSString *NSPrintFaxReturnReceipt
NSString *NSPrintFaxHighResolution
NSString *NSPrintFaxTrimPageEnds
NSString *NSPrintFaxModem
NSString *NSPrintFaxJob;
```

Save Panels

```
enum {
    NSFileHandlingPanelImageButton,
    NSFileHandlingPanelTitleField,
    NSFileHandlingPanelBrowser,
    NSFileHandlingPanelCancelButton,
    NSFileHandlingPanelOKButton,
    NSFileHandlingPanelForm,
    NSFileHandlingPanelHomeButton,
    NSFileHandlingPanelDiskButton,
    NSFileHandlingPanelDiskEjectButton
};
```

Tags that identify buttons, fields, and other views in the Save panel.

Scrollers

Scroller Arrow

```
typedef enum _NSScrollerArrow {
    NSScrollerIncrementArrow,
    NSScrollerDecrementArrow
} NSScrollerArrow;
```

These constants indicate the two types of scroller arrows. `NSScroller`'s `drawArrow:highlight:` method takes an `NSScrollerArrow` as the first argument.

Scroller Arrow Position

```
typedef enum _NSScrollArrowPosition {
    NSScrollerArrowsMaxEnd,
    NSScrollerArrowsMinEnd,
    NSScrollerArrowsNone
} NSScrollArrowPosition;
```

`NSScroller` uses these constants in its `setArrowPosition:` method to set the position of the arrows within the scroller.

Scroller Parts

```
typedef enum _NSScrollerPart {
    NSScrollerNoPart,
    NSScrollerDecrementPage,
    NSScrollerKnob,
    NSScrollerIncrementPage,
    NSScrollerDecrementLine,
    NSScrollerIncrementLine,
    NSScrollerKnobSlot
} NSScrollerPart;
```

`NSScroller` uses these constants in its `hitPart` method to identify the part of the scroller specified in a mouse event.

Usable Scroller Parts

```
typedef enum _NSScrollerUsablePart {
    NSNoScrollerParts,
    NSOnlyScrollerArrows,
    NSAllScrollerParts
} NSUsableScrollerParts;
```

These constants define the usable parts of an `NSScroller` object.

Scroller Width

```
const float NSScrollerWidth;
```

Identifies the default width of a vertical `NSScroller` object and the default height of a horizontal `NSScroller` object.

Text

Line Break Information

```
typedef struct _NSBreakArray {
    NSTextChunk chunk;
    NSLineDesc  breaks[1];
} NSBreakArray;
```

Holds line-break information for an `NSText` object. It's mainly an array of line descriptors.

Line Character Array

```
typedef struct _NSCharArray {
    NSTextChunk chunk;
    unsigned char text[1];
} NSCharArray;
```

Holds the character array for the current line in the `NSText` object.

Character Filter Function

```
typedef unsigned short (*NSCharFilterFunc) (
    unsigned short charCode,
    int flags,
    NSStringEncoding theEncoding);
```

The character filter function analyzes each character the user enters in the `NSText` object.

Finite-State Machine

```
typedef struct _NSFSM {
    const struct _NSFSM  *next;
    short   delta;
    short   token;
} NSFSM;
```

A word definition finite-state machine structure used by an `NSText` object.

Line Height Change Information

```
typedef struct _NSHeightChange {
    NSLineDesc  lineDesc;
    NSHeightInfo heightInfo;
} NSHeightChange;
```

Associates line descriptors and line-height information in an `NSText` object.

Line Height Information

```
typedef struct _NSHeightInfo {
    float newHeight;
    float oldHeight;
    NSLineDesc  lineDesc;
} NSHeightInfo;
```

Stores height information for each line of text in an `NSText` object.

Line Select and Draw Information

```
typedef struct _NSLay {
    float x;
    float y;
    short   offset;
    short   chars;
    id  font;
    void   *paraStyle;
    NSRun *run;
    NSLayFlags lFlags;
} NSLay;
```

Represents a single sequence of text in a line and records everything needed to select or draw that piece.

```
typedef struct _NSLayArray {
    NSTextChunk chunk;
    NSLay   lays[1];
} NSLayArray;
```

Holds the layout for the current line. Since the structure's first field is an `NSTextChunk` structure, `NSLayArray`s can be manipulated by the functions that manage variable-sized arrays of records.

```
typedef struct {
    unsigned int mustMove:1;
    unsigned int isMoveChar:1;
    unsigned int RESERVED:14;
} NSLayFlags;
```

Records whether a text lay in an NSText object needs special treatment (for example, because of non-printing characters).

```
typedef struct _NSLayInfo {
    NSRect rect;
    float descent;
    float width;
    float left;
    float right;
    float rightIndent;
    NSLayArray *lays;
    NSWidthArray *widths;
    NSCharArray *chars;
    NSTextCache cache;
    NSRect *textClipRect;
    struct _lFlags {
        unsigned int horizCanGrow:1;
        unsigned int vertCanGrow:1;
        unsigned int erase:1;
        unsigned int ping:1;
        unsigned int endsParagraph:1;
        unsigned int resetCache:1;
        unsigned int RESERVED:10;
    } lFlags;
} NSLayInfo;
```

NSText's scanning and drawing functions use this structure to communicate information about lines of text.

Line Descriptor

```
typedef short NSLineDesc;
```

Used to identify lines of text in the NSText object.

Paragraph Properties

```
typedef enum _NSParagraphProperty {
    NSLeftAlignedParagraph,
    NSRightAlignedParagraph,
    NSCenterAlignedParagraph,
    NSJustificationAlignedParagraph,
    NSFirstIndentParagraph,
    NSIndentParagraph,
    NSAddTabParagraph,
    NSRemoveTabParagraph,
    NSLeftMarginParagraph,
    NSRightMarginParagraph
} NSParagraphProperty;
```

The constants of this type identify specific paragraph properties for selected text. `NSCStringText`'s `setSelProp:to:` method takes this argument type.

Text Runs

```
typedef struct _NSRun {
    id  font;
    int chars;
    void   *paraStyle;
    int  textRGBColor;
    unsigned char superscript;
    unsigned char subscript;
    id info;
    NSRunFlags rFlags;
} NSRun;
```

In an `NSText` object, this structure represents a single sequence of text with a given format.

Text Run Array

```
typedef struct _NSRunArray {
    NSTextChunk chunk;
    NSRun   runs[1];
} NSRunArray;
```

This structure holds the array of text runs in an `NSText` object. Since the first field is an `NSTextChunk` structure you can manipulate the items in the array with the functions that manage variable-sized arrays of records.

Text Run Flags

```
typedef struct {
    unsigned int underline:1;
    unsigned int dummy:1;
    unsigned int subclassWantsRTF:1;
    unsigned int graphic:1;
    unsigned int forcedSymbol:1;
    unsigned int RESERVED:11;
} NSRunFlags;
```

The fields of this structure record whether a run in an `NSText` object contains graphics, is underlined, or if an alternate character forced the use of a symbol.

Selection Points

```
typedef struct _NSSelPt {

    int cp; //Character position
    int line; // Offset of NSLineDesc in break table
    float x; //x coordinate
    float y; //y coordinate
    int c1st; //Position of first char in line
    float ht; //Line height
} NSSelPt;
```

Represents one end of a selection in an `NSText` object.

Tab Stops

```
typedef struct _NSTabStop {
    short   kind;
    float x;
} NSTabStop;
```

This structure describes an `NSText` object's tab stops.

Text Blocks

```
typedef struct _NSTextBlock {
    struct _NSTextBlock *next;
    struct _NSTextBlock *prior;
    struct _tbFlags {
        unsigned int malloced:1;
```

```
        unsigned int PAD:15;
    } tbFlags;
    short   chars;
    unsigned char *text;
} NSTextBlock;
```

A structure holds text characters in blocks no bigger than `NSTextBlockSize` (see below). A linked list of these text blocks comprises the text for an `NSText` object.

Text Block Size

```
enum {
    NSTextBlockSize   = 512
};
```

The size, in bytes, of a text block.

Text Cache

```
typedef struct _NSTextCache {
    int curPos;
    NSRun *curRun;
    int runFirstPos;
    NSTextBlock *curBlock;
    int blockFirstPos;
} NSTextCache;
```

This structure describes the current text block and run, and the cursor position in the text.

Text Chunks

```
typedef struct _NSTextChunk {
    short growby;
    int allocated;
    int used;
} NSTextChunk;
```

Text objects use this structure to implement variable-sized arrays of records.

Text Filter Function

```
typedef char  *(*NSTextFilterFunc) (
    id self,
    unsigned char * insertText,
    int *insertLength,
    int position);
```

A text filter function implements autoindenting and other features in an `NSText` object.

Text Scanning and Drawing Functions

```
typedef int (*NSTextFunc) (
    id self,
    NSLayInfo *layInfo);
```

This is the type for an `NSText` object's scanning and drawing function, as set through `NSCStringText`'s `setScanFunc:` and `setDrawFunc:` methods.

NSTextAlignment

```
typedef enum _NSTextAlignment {
    NSLeftTextAlignment,
    NSRightTextAlignment,
    NSCenterTextAlignment,
    NSJustifiedTextAlignment,
    NSNaturalTextAlignment
} NSTextAlignment;
```

The constants of this type determine text alignment. Used by `NSCell`, `NSControl`, `NSForm`, `NSFormCell`, and `NSText` methods. `NSNaturalTextAlignment` indicates the default alignment for the text.

Text Style

```
typedef struct _NSTextStyle {
    float indent1st;
    float indent2nd;
    float lineHt;
    float descentLine;
    NSTextAlignment alignment;
```

```
    short numTabs;
    NSTabStop *tabs;
} NSTextStyle;
```

`NSText` uses this structure to describe text layout and tab stops.

Line Width Array

```
typedef struct _NSWidthArray {
    NSTextChunk chunk;
    float widths[1];
} NSWidthArray;
```

Holds the character widths for the current line. Since the first field is an `NSTextChunk` structure, you can manipulate the items in the array with the functions that manage variable-sized arrays of records.

Left Tab

```
enum {
    NSLeftTab
};
```

Used by the `NSText` object's tab functions.

Backspace, Carriage Return, Delete, and Backtab Key Codes

```
enum {
    NSBackspaceKey = 8,
    NSCarriageReturnKey = 13,
    NSDeleteKey = 0x7f,
    NSBacktabKey = 25
};
```

These character-code constants are used by the `NSText` object's character filter function.

Text Movement Key Codes

```
enum {
    NSIllegalTextMovement  = 0,
    NSReturnTextMovement  = 0x10,
    NSTabTextMovement   = 0x11,
    NSBacktabTextMovement  = 0x12,
```

```
    NSLeftTextMovement   = 0x13,
    NSRightTextMovement   = 0x14,
    NSUpTextMovement   = 0x15,
    NSDownTextMovement   = 0x16
};
```

Movement codes describing types of movement between text fields.

Break Tables

```
const NSFSM *NSCBreakTable;
int NSCBreakTableSize;
const NSFSM *NSEnglishBreakTable;
int NSEnglishBreakTableSize;
const NSFSM *NSEnglishNoBreakTable;
int NSEnglishNoBreakTableSize;
```

These tables (with their associated sizes) are finite-state machines that determine word wrapping in an `NSText` object.

Character Category Tables

```
const unsigned char *NSCCharCatTable;
const unsigned char *NSEnglishCharCatTable;
```

These tables define the character classes used in an `NSText` object's break and click tables.

Click Tables

```
const NSFSM *NSCClickTable;
int NSCClickTableSize;
const NSFSM *NSEnglishClickTable;
int NSEnglishClickTableSize;
```

`NSText` objects use these tables as finite-state machines that determine which characters are selected when the user double-clicks.

Smart Cut and Paste Tables

```
const unsigned char *NSCSmartLeftChars;
const unsigned char *NSCSmartRightChars;
const unsigned char *NSEnglishSmartLeftChars;
const unsigned char *NSEnglishSmartRightChars;
```

These tables are suitable as arguments for the `NSCStringText` methods `setPreSelSmartable:` and `setPostSelSmartTable:`. When users paste text into a text object, if the character to the left (right) side of the new word is not in the left (right) table, an extra space is added to that side.

NSCStringText Internal State Structure

This is the structure returned by the `cStringTextInternalState` method of `NSCStringText`, for use only by applications that need to access the internal state of an `NSCStringText` object.

```
typedef struct _NSCStringTextInternalState  {

//Pointer to state table that specifies word and line breaks.
const NSFSM *breakTable;

//Pointer to state table that defines word boundaries for a
//double-click selection.
const NSFSM *clickTable;

//Pointer to table that specifies which characters on the left
//end of a selection are treated as equivalent to a space.
const unsigned char *preSelSmartTable;

//Pointer to table that specifies which characters on the right
//end of a selection are treated as equivalent to a space.
const unsigned char *postSelSmartTable;

//Pointer to table that maps ASCII characters to character classes.
const unsigned char *charCategoryTable;

//Record of notification methods the delegate implements.
char delegateMethods;

//Function to check each character as it’s typed into the text.
NSCharFilterFunc charFilterFunc;

//Function to check text that’s being added to the NSCStringText
object.
NSTextFilterFunc textFilterFunc;

//Reserved for internal use
NSString *_string;

//Function that calculates the line of text.
```

```
NSTextFunc scanFunc;

//Function that draws the line of text.
NSTextFunc drawFunc;

//Object that's notified when the NSCStringText object is modified.
id delegate;

//Integer the delegate uses to identify the NSCStringText object.
int tag;

//Timed entry number for the vertical bar that marks the
//insertion point.
void *cursorTE;

//Pointer to first record in a linked list of text blocks.
NSTextBlock *firstTextBlock;

//Pointer to last record in a linked list of text blocks.
NSTextBlock *lastTextBlock;

//Pointer to array of format runs. By default, theRuns points
//to a single run of the default font.
NSRunArray  *theRuns;

//Format run to use for the next characters entered.
NSRun  typingRun;

//Pointer to the array of line breaks.
NSBreakArray *theBreaks;

//Line containing the end of the growing selection.
int growLine;

//Number of characters in the NSCStringText object.
int textLength;

//Bottom of the last line of text, relative to the origin of
bodyRect.
float maxY;

//Widest line of text. Only accurate after calcLine method is
invoked.
float maxX;
```

```
//Rectangle in which the NSCStringText object draws.
NSRect bodyRect;

//Reserved for internal use.
float borderWidth;

//Number of clicks that created the selection.
char clickCount;

//Starting position of the selection.
NSSelPt sp0;

//Ending position of the selection.
NSSelPt spN;

//Left anchor position.
NSSelPt anchorL;

//Right anchor position.
NSSelPt anchorR;

//Maximum size of the frame rectangle.
NSSize maxSize;

//Minimum size of the frame rectangle.
NSSize minSize;

struct _tFlags {
    #ifdef __BIG_ENDIAN__
    //Reserved for internal use.
    unsigned int _editMode:2;
    unsigned int _selectMode:2;
    unsigned int _caretState:2;

    //True if any changes have been made to the text since the
    //NSCStringText object became first responder
    unsigned int changeState:1;

    //True if the NSCStringText object wraps words whose length
    //exceeds the line length on a character basis. False if
    //such words are truncated at end of line.
    unsigned int charWrap:1;

    //True if the left mouse button (or any button if
    //button functions are not differentiated) is down.
    unsigned int haveDown:1;
```

```
    //True if the anchor's position is at sp0.
    unsigned int anchorIs0:1;

    //True if the NSCStringText object's width can grow or shrink.
    unsigned int horizResizable:1;

    //True if the NSCStringText object's height can grow or shrink
    unsigned int vertResizable:1;

    //Reserved for internal use.
    unsigned int overstrikeDiacriticals:1;

    //True if the NSCStringText object uses one font for all
    //its text.
    unsigned int monoFont:1;

    //True if the NSCStringText object doesn't update the font
    //panel automatically.
    unsigned int disableFontPanel:1;

    //True if the NSCStringText object is a subview of
    //an NSClipView.
    unsigned int inClipView:1;

    #else
    unsigned int inClipView:1;
    unsigned int disableFontPanel:1;
    unsigned int monoFont:1;
    unsigned int overstrikeDiacriticals:1;
    unsigned int vertResizable:1;
    unsigned int horizResizable:1;
    unsigned int anchorIs0:1;
    unsigned int haveDown:1;
    unsigned int charWrap:1;
    unsigned int changeState:1;
    unsigned int _caretState:2;
    unsigned int _selectMode:2;
    unsigned int _editMode:2;
    #endif
    } tFlags;

//Reserved for internal use.
void *_info;
void *_textStr;
}  NSCStringTextInternalState;
```

Tracking Rectangle Tag

```
typedef int NSTrackingRectTag;
```

A unique identifier of a tracking rectangle assigned by `NSView`. See `NSView`'s `addTrackingRectangle:owner:userData:assumeInside:` method.

Border Type

```
typedef enum _NSBorderType {
    NSNoBorder,
    NSLineBorder,
    NSBezelBorder,
    NSGrooveBorder
} NSBorderType;
```

Constants representing the four types of borders that can appear around `NSView` objects.

Autoresizing Constants

```
enum {
    NSViewNotSizable,
    NSViewMinXMargin,
    NSViewWidthSizable,
    NSViewMaxXMargin,
    NSViewMinYMargin,
    NSViewHeightSizable,
    NSViewMaxYMargin
};
```

`NSView` uses these autoresize constants to describe the parts of a view (or its margins) that are resized when the view's superview is resized.

Windows

Window Levels

```
enum {
    NSNormalWindowLevel   = 0,
    NSFloatingWindowLevel  = 3,
    NSDockWindowLevel   = 5,
    NSSubmenuWindowLevel  = 10,
    NSMainMenuWindowLevel  = 20
};
```

These constants list the window-device tiers that the Application Kit uses. Windows are ordered (or “layered”) within tiers: The uppermost window in one tier can still be obscured by the lowest window in the next higher tier.

Window Styles

```
enum {
    NSBorderlessWindowMask,
    NSTitledWindowMask,
    NSClosableWindowMask,
    NSMiniaturizableWindowMask,
    NSResizableWindowMask
};
```

Bitmap masks to determine window styles.

Size Globals

```
NSSize NSIconSize;
NSSize NSTokenSize;
```

These global constants give the dimensions of an icon and token.

Workspaces

Workspace File Type Globals

```
NSString *NSPlainFileType;
NSString *NSDirectoryFileType;
NSString *NSApplicationFileType;
NSString *NSFilesystemFileType;
NSString *NSShellCommandFileType;
```

Identifies the type of file queried by the method `getInfoForFile:application:type:`. The file type is passed back by reference in this method's last argument.

Workspace File Operation Globals

```
NSString *NSWorkspaceCompressOperation;
NSString *NSWorkspaceCopyOperation;
NSString *NSWorkspaceDecompressOperation;
NSString *NSWorkspaceDecryptOperation;
NSString *NSWorkspaceDestroyOperation;
NSString *NSWorkspaceDuplicateOperation;
NSString *NSWorkspaceEncryptOperation;
NSString *NSWorkspaceLinkOperation;
NSString *NSWorkspaceMoveOperation;
NSString *NSWorkspaceRecycleOperation;
```

Used as file-operation arguments in the `performFileOperation:source:destination:files:tag:` method (first argument).

Part 2 — Foundation Kit

Classes 5

The Foundation Kit contains the OpenStep root class `NSObject`, and other classes represent basic data types such as byte arrays, character sets, and strings; object collections such as sets, arrays, and dictionaries; and classes representing system information such as time and dates. These classes provide Application Kit support.

The following diagram shows the Foundation Kit classes and their inheritance relationships. After the diagram, the class descriptions are arranged in alphabetical order.

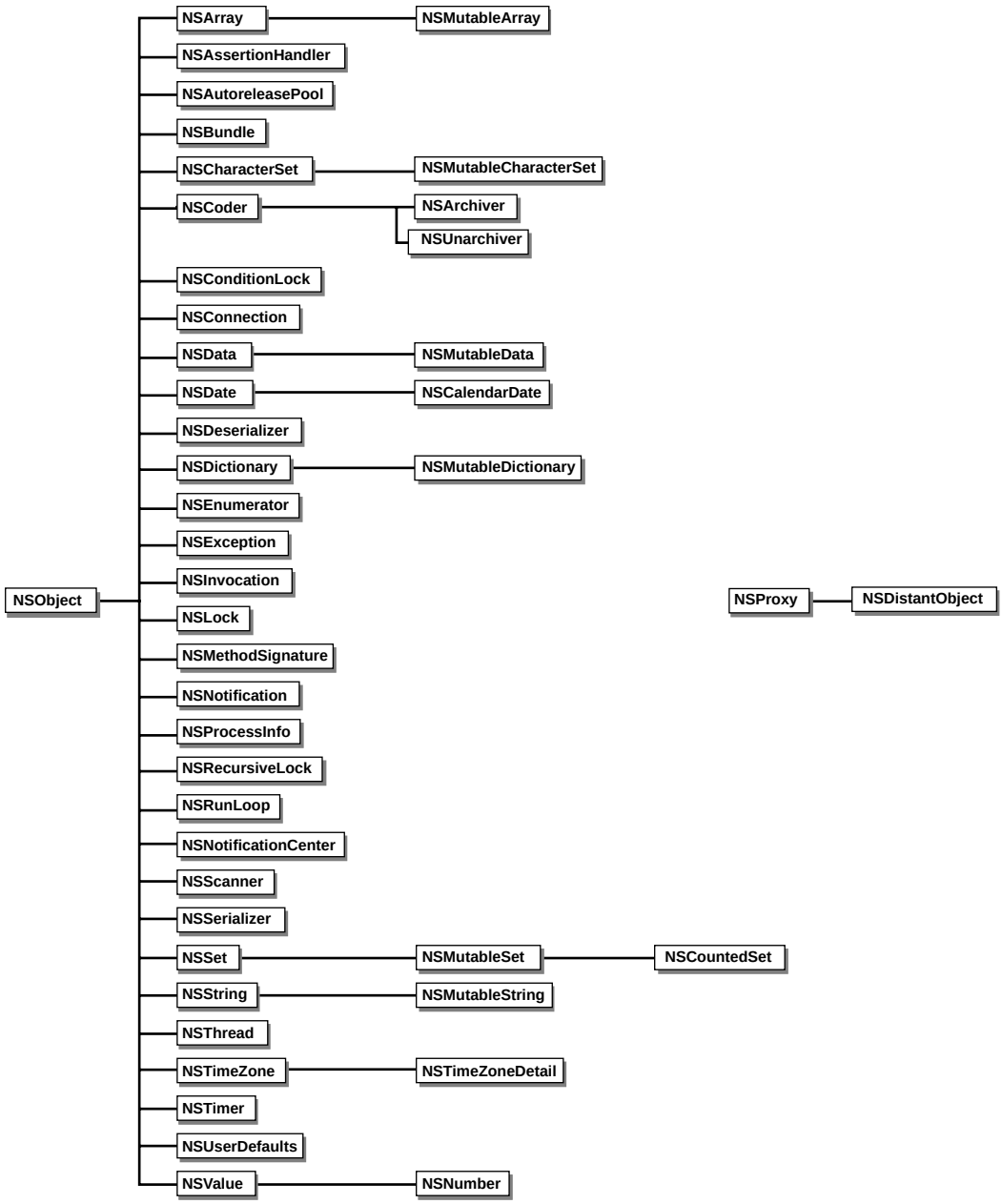


Figure 5-1 Foundation Kit Classes

NSArchiver

Characteristic	Description
Inherits From:	NSCoder : NSObject
Conforms To:	NSObject (NSObject)
Declared In:	Foundation / NSArchiver.h

Class Description

`NSArchiver`, a concrete subclass of `NSCoder`, defines an object that encodes Objective C objects into an architecture-independent file storage format that can be stored in a file. When objects are archived, their class information and the values of their instance variables are written to the archive. `NSArchiver`'s companion class, `NSUnarchiver`, takes an archive file and decodes its contents into a set of objects equivalent to the original one.

Archiving is typically initiated by sending an `encodeRootObject:` or `archiveRootObject:toFile:` message to an archiver object. These messages specify a single object that is the starting point for archiving. The root object receives an `encodeWithCoder:` message (see the `NSCoding` protocol) that allows it to begin archiving itself and the other objects that it's connected to. An object responds to an `encodeWithCoder:` message by writing its instance variables to the archiver.

An object doesn't have to archive the values of each of its instance variables. Some values may not be important to reestablish and others may be derivable from related state upon unarchiving. Other instance variables should be written to the archive only under certain conditions, as explained in the following.

`NSArchiver` overrides the inherited `encodeRootObject:` and `encodeConditionalObject:` methods to support the conditional archiving of members of a graph of objects. When an object receives an `encodeWithCoder:` message, it should respond by unconditionally archiving instance variables that are intrinsic to its nature, with the exceptions noted above, and conditionally archiving those that are not. For example, an `NSView` unconditionally archives its array of subviews using `encodeObject:`, but conditionally archives its superview using `encodeConditionalObject:`. The archiving system notes each reference to a conditional object, but doesn't actually archive the object unless some other object in the graph requests the

object to be archived unconditionally. This ensures that an object is only archived once despite multiple references to it in the object graph. When the objects are extracted from the archive, the multiple references to objects are resolved, and an equivalent graph of objects is reestablished. See also `NSUnarchiver`, `NSSerializer`.

Method Types

Activity	Class Method
Initializing an NSArchiver	– initForWritingWithMutableData:
Archiving data	+ archivedDataWithRootObject: + archivedRootObject:toFile: – encodeArrayOfObjCType:count:at: – encodeConditionalObject: – encodeRootObject:
Getting data from the NSArchiver	– archiverData
Substituting one class for another	– classNameEncodedForTrueClassName: – encodeClassName:intoClassName:

Class Methods

`archivedDataWithRootObject:`

```
+ (NSData *)archivedDataWithRootObject:(id)rootObject
```

Creates and returns a data object after initializing an archiver with that data object, and encoding the archiver with `rootObject`.

`archivedRootObject:toFile:`

```
+ (BOOL)archiveRootObject:(id)rootObjectToFile:(NSString *)path
```

Archives `rootObject` by encoding it as a data object in an archiver and writing that data object to file `path`. Returns `YES` upon success, and returns `NO` otherwise.

Instance Methods

archiverData

```
- (NSMutableData *)archiverData
```

Returns the data object, in mutable form, that is associated with the receiving archiver.

classNameEncodedForTrueClassName:

```
- (NSString *)classNameEncodedForTrueClassName:(NSString *)trueName
```

Returns the class name used to archive instances of the class `trueName`. See also `encodeClassName:intoClassName:`.

encodeArrayOfObjCType:count:at:

```
- (void)encodeArrayOfObjCType:(const char *)itemType
    count:(unsigned int)count at:(const void *)array
```

Encodes an `array` of `count` data elements of the same Objective C data `type`. `itemType` can be some combination of the following type descriptors in the following table.

Table 5-1 Type Descriptors

Descriptor	Type
id	@
Class	#
SEL	:
char	c
unsigned char	C
short	s
unsigned short	S
int	i
unsigned int	I
long	l

Table 5-1 Type Descriptors

Descriptor	Type
unsigned long	L
long long	q
float	f
double	d
bitfield	b
void	v
undefined	?
pointer	^
char *	*
array	[<count><types>]
union	(<types>)
structure	{<types>}

For example, if `itemType` were "{sic*@}", a structure containing a short, an int, a char, a char *, and an object would be encoded for each array element. See also `decodeArrayOfObjCType:count:at:` (`NSUnarchiver`).

encodeClassName:intoClassName:

```
- (void)encodeClassName:(NSString *)trueName
    intoClassName:(NSString *)inArchiveName
```

Encodes in the archive a substitute class name for the real class name (`trueName`).

encodeConditionalObject:

```
- (void)encodeConditionalObject:(id)object
```

Encodes into the linearized data a conditional `object` that points back toward a root object. If `nil` is specified for `object`, it encodes it as `nil` unconditionally. This method raises an `NSInvalidArgumentException` if no root object has been encoded.

encodeRootObject:

```
- (void)encodeRootObject:(id)rootObject
```

Encodes the `rootObject` at the start of the linearized data representing the object graph. This method raises an `NSInvalidArgumentException` if the root object has already been encoded.

initForWritingWithMutableData:

```
- (id)initForWritingWithMutableData:(NSMutableData *)mdata
```

Initializes an archiver, encoding stream and version information into mutable data `mdata`. This method raises `NSInvalidArgumentException` if the `mdata` argument is `nil`.

NSArray

Characteristic	Description
Inherits From:	NSObject
Conforms To:	NSCoding, NSCopying, NSMutableCopying NSObject (NSObject)
Declared In:	Foundation / NSArray.h

Class Description

The `NSArray` class declares the programmatic interface to an object that manages an immutable array of objects. (The complementary class `NSMutableArray` manages modifiable arrays of objects.) `NSArray`'s two primitive methods—`count` and `objectAtIndex:`—provide the basis for all the other methods in its interface. The `count` method returns the number of elements in the array. `objectAtIndex:` gives you access to the array elements by index value, with index values starting at 0.

The methods `objectEnumerator` and `reverseObjectEnumerator` also permit sequential access of the elements of the array, differing only in the direction of travel through the elements. These methods are provided so that array objects can be traversed in a manner similar to that used for objects of other collection classes, such as `NSDictionary`.

Generally, you instantiate an `NSArray` by sending one of the `array...` messages to the `NSArray` class object. These methods return an `NSArray` containing the elements you pass in as arguments. (Note that arrays can't contain `nil` objects.) These objects aren't copied; rather, each object receives a `retain` message before it's added to the array. When an object is removed from an array, it's sent a `release` message.

`NSArray` provides methods for querying the elements of the array. `indexOfObject:` searches the array for the object that matches its argument. To determine whether the search is successful, each element of the array is sent an `isEqual:` message, as declared in the `NSObject` protocol. Another method, `indexOfObjectIdenticalTo:`, is provided for the less common case of determining whether a specific object is present in the array. `indexOfObjectIdenticalTo:` tests each element in the array to see whether its `id` matches that of the argument.

The `makeObjectsPerform:` and `makeObjectsPerform:withObject:` methods let you act on the individual objects in the array by sending them messages. To act on the array as a whole, a variety of methods are defined. You can create a sorted version of the array (`sortedArrayUsingSelector:` and `sortedArrayUsingFunction:context:`), extract a subset of the array (`subarrayWithRange:`), or concatenate the elements of an array of `NSString` objects into a single string (`componentsJoinedByString:`). In addition, you can compare two array objects using the `isEqualToArray:` and `firstObjectCommonWithArray:` methods.

Method Types

Activity	Class Method
Allocating, initializing, and storing an array	+ allocWithZone: + array + arrayWithContentsOfFile: + arrayWithObject: + arrayWithObjects: – arrayByAddingObject: – arrayByAddingObjectsFromArray: – initWithArray: – initWithContentsOfFile: – initWithObjects: – initWithObjects:count: – writeToFile:atomically:
Querying the array	– containsObject: – count – getObjects: – getObjects:range: – indexOfObject: – indexOfObject:inRange: – indexOfObjectIdenticalTo: – indexOfObjectIdenticalTo:inRange: – lastObject – objectAtIndex: – objectEnumerator – reverseObjectEnumerator – sortedArrayHint
Sending messages to elements	– makeObjectsPerform: – makeObjectsPerform:withObject:
Comparing arrays	– firstObjectCommonWithArray: – isEqualToArray:
Deriving new arrays	– sortedArrayUsingFunction:context: – sortedArrayUsingFunction:context:hint: – sortedArrayUsingSelector: – subarrayWithRange:
Joining string elements	– componentsJoinedByString:
Creating a string description of the array	– description – descriptionWithLocale: – descriptionWithLocale:indent:

Class Methods

allocWithZone:

```
+ (id)allocWithZone:(NSZone *)zone
```

Returns an uninitialized array object in `zone`. See also `array`.

array

```
+ (id)array
```

Returns an empty array object. See also `arrayWithObject:`, `arrayWithObjects:`, `initWithObjects:`.

arrayWithContentsOfFile:

```
+ (id)arrayWithContentsOfFile:(NSString *)path
```

Returns an array initialized with the contents of the file specified by path. Returns `nil` if `path` does not contain an array, or if there is a file error. See also `initWithContentsOfFile:`, `writeToFile:atomically:`, `arrayWithObjects:`.

arrayWithObject:

```
+ (id)arrayWithObject:(id)anObject
```

Returns an `NSArray` containing the single element `anObject`. Raises an `NSInvalidArgumentException` if `anObject` is `nil`. See also `arrayWithObjects:`, `arrayByAddingObject:`, `arrayByAddingObjectsFromArray:`, `arrayWithContentsOfFile:`, `initWithArray:`.

arrayWithObjects:

```
+ (id)arrayWithObjects:(id)firstObj,...
```

Returns an `NSArray` containing the objects in the argument list. The object list is comma-separated and ends with `nil`. See also `arrayWithObject:`, `arrayWithContentsOfFile:`.

Instance Methods

arrayByAddingObject:

```
- (NSArray *)arrayByAddingObject:(id)anObject
```

Returns an NSArray containing the receiver's elements plus anObject at the end of the array. See also arrayByAddingObjectsFromArray:.

arrayByAddingObjectsFromArray:

```
- (NSArray *)arrayByAddingObjectsFromArray:(NSArray *)anotherArray
```

Returns an NSArray containing the receiver's elements plus the elements from anotherArray added to the end of the returned array. See also arrayByAddingObject:.

componentsJoinedByString:

```
- (NSString *)componentsJoinedByString:(NSString *)separator
```

Returns a string that's the result of interposing separator between the elements of the receiver's array. If the receiver's length is 0, the null string (@"") is returned.

containsObject:

```
- (BOOL)containsObject:(id)anObject
```

Returns YES if anObject is present in the array, and returns NO otherwise. See also indexOfObject:, count, objectAtIndex:, objectEnumerator, lastObject.

count

```
- (unsigned int)count
```

Returns the number of objects currently in the array. The default implementation returns 0.

description

```
- (NSString *)description
```

Returns a string object that represents the contents of the receiving array in human-readable form. This method sends the message `descriptionWithLocale:nil indent:0` to each object in the array. See also `descriptionWithLocale:`, `descriptionWithLocale:indent:`.

descriptionWithLocale:

```
- (NSString *)descriptionWithLocale:
    (NSDictionary *)localeDictionary
```

Returns a string representation of the `NSArray` object. Included are the key and values that represent the locale data from `localeDictionary`. This method sends the message `descriptionWithLocale:localeDictionary indent:0`. See also `description`, `descriptionWithLocale:indent:`.

descriptionWithLocale:indent:

```
- (NSString *)descriptionWithLocale:
    (NSDictionary *)localeDictionary
    indent:(unsigned int)level
```

Returns a string representation of the `NSArray` object. Included are the key and values that represent the locale data from `localeDictionary`. Elements of the array are indented from the left margin by `level` + 1 multiples of four spaces, to make the output more readable. See also `description`, `descriptionWithLocale:indent:`.

firstObjectCommonWithArray:

```
- (id)firstObjectCommonWithArray:(NSArray *)otherArray
```

Returns the first object from the receiver's array that's equal to an object in `otherArray`. Returns `nil` if no common object is found. See also `isEqualToArray:`.

getObjects:

```
- (void)getObjects:(id *)buf
```

Returns the receiving array's contents in `buf`. See also `getObjects:range:`.

getObjects:range:

```
- (void)getObjects:(id *)buf range:(NSRange)range
```

Returns the receiving array's contents, within `range`, in `buf`. See also `getObjects:`.

indexOfObject:

```
– (unsigned int)indexOfObject:(id)anObject
```

Returns the index of `anObject`, if found; otherwise, returns `NSNotFound`. This method is similar to `indexOfObjectIdenticalTo:`, but instead of just comparing the `id`s, this method sends an `isEqual:` message (see the `NSObject` protocol) to each object in the array using the `anObject` as the argument. If the objects in the array are of a class that overrides `NSObject`'s default `isEqual:`, the index of the last object which "is equal" to the argument will be returned (where "is equal" means whatever the class defines it to mean). If `NSObject`'s default `isEqual:` method is not overriden by the array elements class, then this method is equivalent to `indexOfObject:`, but is less efficient. See also `indexOfObjectIdenticalTo:`.

indexOfObjectIdenticalTo:

```
– (unsigned int)indexOfObjectIdenticalTo:(id)anObject
```

Returns the index of `anObject`, if found; otherwise, returns `NSNotFound`. This method checks the elements in the array from first to last by comparing their `id`s. See also `indexOfObject:`, `indexOfObjectIdenticalTo:inRange:`.

indexOfObjectIdenticalTo:inRange:

```
- (unsigned)indexOfObjectIdenticalTo:(id)anObject
    inRange:(NSRange)range
```

Searches the specifed array `range` for `anObject` by comparing `id`s. Returns `anObject`'s array index if found, otherwise `NSNotFound` is returned. See also `indexOfObjectIdenticalTo:`, `indexOfObject:inRange:`.

indexOfObject:inRange:

```
- (unsigned)indexOfObject:(id)anObject inRange:(NSRange)range
```

Searches the specifed array `range` for `anObject`. Returns `anObject`'s array index if found, otherwise `NSNotFound` is returned. Object equality is determined by `NSObject` protocol's `isEqual:` method. See also `indexOfObject:`, `indexOfObjectIdenticalTo:inRange:`.

initWithArray:

```
– (id)initWithArray:(NSArray *)anotherArray
```

Initializes a newly allocated array object by placing in it the objects contained in `anotherArray`. See also `initWithContentsOfFile:`, `initWithObjects:`.

initWithContentsOfFile:

```
- (id)initWithContentsOfFile:(NSString *)path
```

Initializes a newly allocated array with the contents of `path`. Returns `nil` if `path` does not represent an array, or if there is a file error. See also `writeToFile:atomically:`, `arrayWithContentsOfFile:`, `initWithArray:`.

initWithObjects:

```
– (id)initWithObjects:(id)firstObj,...
```

Initializes a newly allocated array object by placing in it the objects in the argument list. The object list is comma-separated and ends with `nil`. This method raises an `NSInvalidArgumentException` if any object in the list of objects is `nil`. See also `initWithObjects:count:`, `initWithArray:`, `initWithContentsOfFile:`.

initWithObjects:count:

```
– (id)initWithObjects:(id *)objects count:(unsigned int)count
```

Initializes a newly allocated array object by placing in it `count` objects from the `objects` array. This method raises an `NSInvalidArgumentException` if any objects in the `objects` array is `nil`. See also `initWithObjects:`.

isEqualToArray:

```
- (BOOL)isEqualToArray:(NSArray *)otherArray
```

Compares the receiving array object to `otherArray`. This method returns `YES` if the receiver and `otherArray` contain the identical or equal objects at each location (where identical means the same `id`s, and equal is defined by `NSObject` protocol's `isEqual:` method). Returns `NO` otherwise. See also `firstObjectCommonWithArray:`.

lastObject

```
- (id)lastObject
```

Returns the last object in the array, or `nil` if the array is empty. See also `containsObject:`, `count`.

makeObjectsPerform:

```
- (void)makeObjectsPerform:(SEL)aSelector
```

Sends an `aSelector` message to each object in the array (last to first). See also `makeObjectsPerform:withObject:`.

makeObjectsPerform:withObject:

```
- (void)makeObjectsPerform:(SEL)aSelector withObject:(id)anObject
```

Sends an `aSelector` message to each object in the array (last to first), with `anObject` as an argument. See also `makeObjectsPerform:`.

objectAtIndex:

```
- (id)objectAtIndex:(unsigned int)index
```

Returns the object located at `index`. An array's index starts at 0. This method raises an `NSRangeException` if `index` is beyond the end of the array. See also `indexOfObject:`, `objectEnumerator`.

objectEnumerator

```
- (NSEnumerator *)objectEnumerator
```

Returns an enumerator object that lets you access each object in the array, starting with the first element. See also `NSEnumerator`.

reverseObjectEnumerator

```
- (NSEnumerator *)reverseObjectEnumerator
```

Returns an enumerator object that lets you access each object in the array, from the last element to the first. See also `NSEnumerator`.

sortedArrayHint

```
- (NSData *)sortedArrayHint
```

Returns a hint about the state of the array's sort, which is used by `sortedArrayUsingFunction:context:hint:`. Returns nil for uninteresting hints. See also `sortedArrayUsingFunction:context:hint:`.

sortedArrayUsingFunction:context:

```
- (NSArray *)sortedArrayUsingFunction:(int(*)(id element1,
    id element2,void *userData))comparator
    context:(void *)context
```

Returns an array listing the receiver's elements in ascending order as defined by the comparison function `comparator`. `context` is passed to the comparator function as its third argument. See also `sortedArrayUsingSelector:`, `subarrayWithRange:`.

sortedArrayUsingFunction:context:hint:

```
- (NSArray *)sortedArrayUsingFunction:
    (int (*)(id element1, id element2, void *userData))comparator
    context:(void *)context
    hint:(NSData *)hint
```

Returns an array listing the receiver's elements in ascending order as defined by the comparison function `comparator`. `context` is passed to the comparator argument as its third argument. If hint is `nil`, this method behaves identically to `sortedArrayUsingFunction:context:`. See also `sortedArrayHint`.

sortedArrayUsingSelector:

```
- (NSArray *)sortedArrayUsingSelector:(SEL)comparator
```

Returns an array listing the receiver's elements in ascending order, as determined by the comparison method specified by the selector `comparator`. See also `sortedArrayUsingFunction:context:`, `subarrayWithRange:`.

subarrayWithRange:

```
- (NSArray *)subarrayWithRange:(NSRange)range
```

Returns an array containing the receiver's elements that fall within the limits specified by `range`. See also `sortedArrayUsingSelector:`.

writeToFile:atomically:

```
- (BOOL)writeToFile:(NSString *)path
    atomically:(BOOL)useAuxiliaryFile
```

Writes the array to the file specified by `path`. If `useAuxiliaryFile` is `YES`, the data is written to a backup file and then, assuming no errors occur, the backup file is renamed atomically to the intended file name. See also `arrayWithContentsOfFile:`, `initWithContentsOfFile:`.

5

NSAssertionHandler

Characteristic	Description
Inherits From:	NSObject
Conforms To:	NSObject (NSObject)
Declared In:	Foundation / NSExceptions.h

Class Description

An *assertion* is a statement about conditions during the execution of program code, such as the relationship between variables, the state of a Boolean variable, or the value of an expression. If the statement about the conditions proves false, the assertion is said to have failed, and usually some action must be taken to report the failed assertion. Application programmers wishing to provide more detailed control over assertion failures than provided by the macros defined below can use the methods of `NSAssertionHandler` to report assertion failures.

With `NSAssertionHandler` each distinct thread of execution can have a separate handler to deal with failed assertions in code. The `fileName` and `line` arguments to the methods described below can be obtained by using the `__FILE__` and `__LINE__` macros that are predefined in the C pre-processor.

The `Foundation/NSExceptions.h` header file contains a collection of macros that can be used to state assertions within methods, and contains a parallel collection of macros that can be used to state assertions within regular C functions. If the condition tested in any of these macros fails, the current assertion handler is invoked with one of the methods defined below, depending on whether the macro is one of the `NSAssertN` or one of the `NSCAssertN` macros. Separate macros have from one to five arguments. The macros for dealing with assertion failures within methods are:

```
NSAssert1(condition, description, argument1);
NSAssert2(condition, description, argument1, argument2);
NSAssert3(condition, description, argument1, argument2,
argument3);
NSAssert4(condition, description, argument1, argument2,
argument3, argument4);
NSAssert5(condition, description, argument1, argument2, argument3,
argument4, argument5);
```

In each case, `condition` is the statement to be tested, for example, `index < length`; `description` is a description of the reason for the failure (in the form of a printf-style format `NSString`); and each `argument`*N* is an argument to be formatted according to the `description` string.

The parallel set of macros for dealing with failed assertions from within C functions have names of the form `NSCAssert`*N* instead of `NSAssert`*N*. The arguments are otherwise the same as the `NSAssert`*N* macros.

Method Types

Activity	Class Method
Getting the current handler	+ currentHandler
Handling failures	– handleFailureInFunction:file:lineNumber: description: – handleFailureInMethod:object:file:lineNumber: description:

Class Methods

`currentHandler`

```
+ (NSAssertionHandler *)currentHandler
```

Returns the assertion handler for the current thread.

Instance Methods

`handleFailureInFunction:file:lineNumber: description:`

```
- (void)handleFailureInFunction:(NSString *)functionName
    file:(NSString *)fileNamelineNumber:(int)line
    description:(NSString *)format,...
```

Logs an error message that includes `functionName`, the source file `fileName` and the `line` number where the failure occurred, and a short description of the failure described by `format`. It then raises an `NSInternalInconsistencyException`.

`handleFailureInMethod:object:file:lineNumber:description:`

```
- (void)handleFailureInMethod:(SEL)selector object:(id)object
    file:(NSString *)fileName lineNumber:(int)line
    description:(NSString *)format,...
```

Logs an error message that includes the method (`selector`) and `object` associated with the failure, the source file `fileName` and `line` number in that file where the failure occured, and a short description of the failure, described by `format`. It then raises an `NSInternalInconsistencyException`.

NSAutoreleasePool

Characteristic	Description
Inherits From:	NSObject
Conforms To:	NSObject (NSObject)
Declared In:	Foundation / NSAutoreleasePool.h

Class Description

The Foundation Kit uses the `NSAutoreleasePool` class to implement `NSObject`'s `autorelease` method. An autorelease pool simply contains other objects, and when deallocated, sends a `release` message to each of those objects. An object can be put into the same pool several times, and receives a `release` message for each time it was put into the pool.

You use autorelease pools to limit the time an object remains valid after it's been "autoreleased", that is, after it's been sent an `autorelease` message or has otherwise been added to an autorelease pool. Autorelease pools are created using the usual `alloc` and `init` messages, and disposed of with `release`. An autorelease pool should always be released in the same context that it was created (invocation of a method or function, or body of a loop). You should never send `retain` or `autorelease` messages to an autorelease pool.

Autorelease pools are automatically created and destroyed in OpenStep applications, so your code normally doesn't have to worry about them. There are two cases, though, where you should explicitly create and destroy your own autorelease pools. If you're writing a program that's not based on the Application Kit, such as a UNIX tool, there's no built-in support for autorelease pools; you must create and destroy them yourself. Also, if you need to write a loop that creates many temporary objects, you should create an autorelease pool in the loop to prevent too long a delay in the disposal of those objects.

Enabling the autorelease feature in a program that's not based on the Application Kit is easy. Many programs have a top-level loop where they do most of their work. To enable the autorelease feature you create an autorelease pool at the beginning of this loop and release it at the end. An `autorelease` message sent in the body of the loop automatically puts its receiver into this pool. The `main()` function might look like this:

```
int main(int argc, char *argv[])
{
    int i;

    /* Do whatever setup is needed. */
    for (i = 0; i < argc; i++) {
        NSString *fileContents;
        NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
        fileContents = [[[NSString alloc]
                        initWithContentsOfFile:argv[i]]autorelease];
        processFile(fileContents);
        [pool release];
    }

    /* Do whatever cleanup is needed. */
    exit(EXIT_SUCCESS);
}
```

Any object autoreleased inside the `for` loop, such as the `fileContents` string object, is added to `pool`. When `pool` is released at the end of the loop those objects added are also released.

Note that autoreleasing doesn't work outside of the loop. This isn't a problem, since the program terminates shortly after the loop ends, and memory leaks aren't usually serious at that stage of execution. Your cleanup code shouldn't refer to any objects created inside the loop, though, since they may be autoreleased in the loop and therefore released as soon as it ends.

Nesting Autorelease Pools

You may need to manually create and destroy autorelease pools even in an application that uses the Application Kit if you write loops that create many temporary objects. For example, if you write a loop that iterates 1000 times and invokes a method that creates 15 temporary objects, those 15,000 objects will remain until the application's autorelease pool is deallocated, possibly well after they're no longer needed.

You can create your own autorelease pools within the loop to prevent these unwanted objects from remaining around. Autorelease pools nest themselves on a per-thread basis, so that if you create your own pool, it adds itself to the application's default pool, forming a stack of autorelease pools. Likewise, if you create another pool (within a nested loop, perhaps), it adds itself to the first pool you created. `autorelease` automatically adds its receiver to the last pool created, creating a nesting of autorelease contexts. The implications of this are described in the following.

A method that creates autorelease pools looks much like the `main()` function given above:

```
- (void)processString:(NSString *)aString
{
    int i;

    for (i = 0; i < 1000; i++) {
        NSAutoreleasePool *subpool = [[NSAutoreleasePool alloc]
init];
        NSString *thisLine;

        thisLine = [self lineNumbered:i fromString:aString];
        /* Do some work with thisLine. */
        [subpool release];
    }
    return;
}
```

If you assume that `lineNumbered:fromString:` returns a string object that's been autoreleased while `subpool` is in effect, that object is released with `subpool` at the end of the loop. The work involving `thisLine` may create other temporary objects, which are also released at the end of the loop. None of these objects remains outside of this loop or the `processString:` method unless they've been retained.

Note that because an autorelease pool adds itself to the previous pool when created, it doesn't cause a memory leak in the face of an exception or other sudden transfer out of the current context. If an exception occurs in the above loop, or if the work in the loop involves immediately returning or breaking out of the loop, the subpool is released by the application's default pool or whatever pool was in effect before the subpool was created, "unwinding" the autorelease-pool stack up to the one that's supposed to be active.

Guaranteeing the Foundation Ownership Policy

By manually creating an autorelease pool, you reduce the potential lifetime of temporary objects to the lifetime of that pool. After an autorelease pool is deallocated, you should regard as "disposed of" any object that was autoreleased while that pool was in effect, and not send a message to that object or return it to the invoker of your method. This method, for example, is incorrect:

```
- findMatchingObject:anObject
{
    id match = nil;
    while (match == nil) {
        NSAutoreleasePool *subpool = [[NSAutoreleasePool alloc]
init];

      /* Do some searching that creates a lot of temporary objects.*/

        match = [self expensiveSearchForObject:anObject];
        [subpool release];
    }
    /* Danger!! The match object may not exist at this point! */
    [match setIsMatch:YES forObject:anObject];
    return match;
}
```

`expensiveSearchForObject:` is invoked while `subpool` is in effect, which means that `match`, which may have been autoreleased, is released at the bottom of the loop. Sending `setIsMatch:forObject:` after the loop could cause the application to crash. Similarly, returning `match` allows the sender of `findMatchingObject:` to send a message to it, also causing your application to crash.

If you must pull a temporary object out of a nested autorelease context, you can do so by retaining the object within the context and then autoreleasing it after the pool has been released. Here's a correct implementation of `findMatchingObject:`.

```
- findMatchingObject:anObject
{
    id match = nil;
    while (match == nil) {
        NSAutoreleasePool *subpool = [[NSAutoreleasePool alloc]
init];


        /* Do a search that creates a lot of temporary objects. */

        match = [self expensiveSearchForObject:anObject];
        if (match != nil) [match retain]; /* Keep match around. */
        [subpool release];
    }
    [match setIsMatch:YES forObject:anObject];
    return [match autorelease];   /* Let match go and return it. */
}
```

By retaining `match` while `subpool` is in effect and autoreleasing it after the `subpool` has been released, `match` is effectively moved from `subpool` to the pool that was previously in effect. This gives it a longer lifetime and allows it to be sent messages outside the loop and to be returned to the invoker of `findMatchingObject:`.

General Exception Conditions

An `NSInvalidArgumentException` is raised on any attempt to send either `retain` or `autorelease` messages to an autorelease pool object.

Method Types

Activity	Class Method
Adding an object to the current pool	+ addObject:
Adding an object to a pool	- addObject:

Class Methods

`addObject:`

```
+ (void)addObject:(id)anObject
```

Adds `anObject` to the active autorelease pool in the current thread.

Instance Methods

`addObject:`

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

Adds `anObject` to the receiver.

NSBundle

Characteristic	Description
Inherits From:	NSObject
Conforms To:	NSObject (NSObject)
Declared In:	Foundation / NSBundle.h

Class Description

A *bundle* is a mechanism for grouping application *resources* into convenient chunks. A typical use for a bundle is to group executable code together with the resources used by that executable code. A major use of bundles is to handle localization issues, as described below in “Localized Resources”.

An `NSBundle` is an object that corresponds to a directory (or folder in the terminology of some operating systems) where application resources are stored. The directory, in essence, "bundles" a set of resources used by an application, and the `NSBundle` object makes those resources available to the application. `NSBundle` is able to find requested resources in the directory and, in some cases, dynamically load executable code. The term "bundle" is used both for the object and for the directory it represents.

Bundled *resources* might include such things as:

- Images, for example, TIFF or EPS images, used by an application's user interface components
- Sounds
- Localized character strings
- Executable code
- User Interface resources—files describing the layout of user interface objects and their relationships with other objects

Each resource within a bundle usually resides in a separate file.

Localized Resources

If an application is to be used in more than one part of the world, its resources may need to be customized, or "localized", for language, country, or cultural region. An application may need, for example, to have separate Japanese, English, French, Hindi, and Swedish versions of the character strings that label menu commands.

Resource files specific to a particular language are grouped together in a subdirectory of the bundle directory. The subdirectory has the name of the language (in English) followed by a `.lproj` extension (for "language project"). The application mentioned above, for example, would have `Japanese.lproj`, `English.lproj`, `French.lproj`, `Hindi.lproj`, and `Swedish.lproj` subdirectories.

Each `.lproj` subdirectory in a bundle has the same set of files; all versions of a resource file must have the same name.

Main Bundle

Every application is considered to have at least one bundle—its *main bundle*—the directory where its executable file is located. If the application is organized into a file package marked by a `.app` extension, the file package is the main bundle.

Other Bundles

An application can be organized into any number of other bundles in addition to the main bundle. For example, an application for managing PostScript printers may have a bundle full of PostScript code to be downloaded to printers. These other bundles usually reside inside the application file package, but they can be located anywhere in the file system. Each bundle directory is represented in the application by a separate `NSBundle` object. By convention, bundle directories other than the main bundle end in a `.bundle` extension.

Dynamically Loadable Classes

Any bundle directory can contain a file with executable code. For the main bundle, that file is the application executable that's loaded into memory when the application is launched. The executable in the main bundle includes the `main()` function and other code necessary to start up the application.

Executable files in other bundle directories hold class and category definitions that the bundle object can dynamically load while the application runs. When asked, the bundle returns class objects for the classes and categories stored in the file. It waits to load the file until those classes are needed.

By using a number of separate bundles, you can split an application into smaller, more manageable pieces. Each piece is loaded into memory only when the code being executed requires it, so the application can start up faster than it otherwise would. Assuming users will rarely use every part of an application, the application will also consume less memory as it runs.

The file that contains dynamically loadable code must have the same name as the bundle directory, but without the `.bundle` extension.

Since each bundle can have only one executable file, that file should be kept free of localizable content. Anything that needs to be localized should be segregated into separate resource files and stored in `.lproj` subdirectories.

Bundle Notification

After a bundle dynamically loads its code, the bundle sends out the `NSBundleDidLoadNotification` notification. This notification's user information dictionary contains an array of strings which are the names of the classes loaded. The key for this dictionary entry is `@"NSLoadedClasses"`. See also `NSNotification`.

Working with Bundles

Generally, you instantiate a bundle object by sending one of the `bundleForClass:`, `bundleWithPath:`, or `mainBundle` methods to the `NSBundle` class object. `mainBundle` gives you the `NSBundle` object corresponding to the directory containing the application's executable.

Method Types

Activity	Class Method
Initializing an NSBundle	– initWithPath:
Getting an NSBundle	+ bundleForClass: + bundleWithPath: + mainBundle
Getting a bundled class	– classNamed: – principalClass
Finding a resource	– pathForResource:ofType: + pathForResource:ofType:inDirectory: – pathForResource:ofType:inDirectory: – pathsForResourcesOfType:inDirectory: – resourcePath
Getting the bundle directory	– bundlePath
Managing localized resources	– localizedStringForKey:value:table:

Class Methods

`bundleForClass:`

```
+ (NSBundle *)bundleForClass:(Class)aClass
```

Returns the NSBundle object that dynamically loaded aClass, or the main bundle object if aClass wasn't dynamically loaded. See also bundleWithPath:, mainBundle, initWithPath:.

bundleWithPath:

```
+ (NSBundle *)bundleWithPath:(NSString *)path
```

Returns an NSBundle object that's initialized for the path directory. See also bundleForClass:, mainBundle, initWithPath:.

mainBundle

```
+ (NSBundle *)mainBundle
```

Returns the NSBundle object that corresponds to the directory where the application executable is located. See also bundleWithPath:.

pathForResource:ofType:inDirectory:

```
+ (NSString *)pathForResource:(NSString *)name
    ofType:(NSString *)ext
    inDirectory:(NSString *)bundlePath
```

Returns the path for the resource identified by name, having the specified filename extension ext, and residing in bundlePath. See also pathForResource:ofType:inDirectory: (instance method) for more information.

Instance Methods

bundlePath

```
- (NSString *)bundlePath
```

Returns a string containing the full path name of the receiver's bundle directory.

classNamed:

```
- (Class)classNamed:(NSString *)className
```

Ensures that the receiver is loaded. Returns the class object for the `className` class, or `nil` if `className` isn't one of the classes associated with the receiver. This method ensures that any code in the bundle directory has been loaded into memory, so the `className` class will be part of the executable image if available. See also `principalClass`.

initWithPath:

```
- (id)initWithPath:(NSString *)path
```

Initializes a newly allocated `NSBundle` object to make it the `NSBundle` for the `path` directory. `path` must be a full pathname or directory. If `path` does not exist or the user doesn't have access to it, the bundle is freed and `nil` is returned. If the application already has a bundle object for `path`, this method then frees the receiver and returns the existing object. It's not necessary to allocate and initialize an object for the main bundle. The `mainBundle` method provides this capability. See also `bundleForClass:`, `mainBundle`.

localizedStringForKey:value:table:

```
- (NSString *)localizedStringForKey:(NSString *)key
    value:(NSString *)value
    table:(NSString *)tableName
```

Returns a localized version of the string designated by `key`. `tableName` specifies the string table to search; if `tableName` is `NULL`, the file `Localizable.strings` is used. `value` specifies the value to return if the key or table can't be found (or if `key` is `NULL`).

pathForResource:ofType:

```
- (NSString *)pathForResource:(NSString *)name
    ofType:(NSString *)ext
```

Returns the path for the resource identified by `name` having the specified file name extension `ext`, or returns `nil` if the resource is not found. To find the resource this method first looks in the bundle directory for `.lproj` subdirectories that match the user's language preferences (as specified in the Preferences application). Subdirectories are searched in order of user preference. If `ext` (which can be `NULL`) does not repeat an extension already added to `name`, it is added to `name` before searching begins.

When this method finds a `.lproj` directory for a preferred language, the bundle first makes sure that the subdirectory contains the requested resource file. If the resource is not located, the bundle looks in the `.lproj` directory of the next most preferred language. If the file can't be found, the bundle looks for a nonlocalized version in the bundle directory. See also `pathForResource:ofType:inDirectory:`.

pathForResource:ofType:inDirectory:

```
- (NSString *)pathForResource:(NSString *name)
    ofType:(NSString *)ext inDirectory:(NSString *)subpath
```

Discovers and returns the full path name of resource `name`, with extension `ext`, within the given `subpath`. The preferred language's resource path is returned; if the resource is not found then `nil` is returned.

`ext` may be `nil` in which case no extension is appended. `subpath` specifies the subpath appended to `Resources/`, and may be `nil`, meaning no subpath is appended. The bundle directories (in the fully specified case) are searched in the following order:

```
<bundle_path>/Resources/'subpath'/<language0>.lproj/'name'.'ext'
<bundle_path>/Resources/'subpath'/<language1>.lproj/name.ext
...
<bundle_path>/Resources/'subpath'/<languageN>.lproj/'name'.'ext'
<bundle_path>/'subpath'/<language0>.lproj/'name'.'ext'
<bundle_path>/'subpath'/<language1>.lproj/'name'.'ext'
...
<bundle_path>/'subpath'/<languageN>.lproj/'name'.'ext'
```

<bundle_path> is the bundle resource path returned by `resourcePath`. See also `pathsForResourcesOfType:inDirectory:`, `pathForResource:ofType:`.

pathsForResourcesOfType:inDirectory:

```
- (NSArray *)pathsForResourcesOfType:(NSString*)ext
inDirectory:(NSString *)subpath
```

Returns an array containing the paths to resources of type `ext`, located within the given `subpath`. `ext` may be `nil` in which case the search returns all resources. The returned array is in no particular order, and contains the full paths of the resources matching the search criteria.

subpath specifies the subpath appended to Resources/, and may be nil. All .lproj's from the user's preferred language list are searched. The bundle directories (in the fully specified case) are searched in the following order:

```
<bundle_path>/Resources/‘subpath'/<language0>.lproj/*.‘ext'
<bundle_path>/Resources/‘subpath'/<language1>.lproj/*.‘ext'
...
<bundle_path>/Resources/‘subpath'/<languageN>.lproj/*.‘ext'
<bundle_path>/‘subpath'/<language0>.lproj/*.‘ext'
<bundle_path>/‘subpath'/<language1>.lproj/*.‘ext'
...
<bundle_path>/‘subpath'/<languageN>.lproj/*.‘ext'
```

<bundle_path> is the bundle resource path returned by resourcePath. The resulting array of strings can each be searched with rangeOfString: if a particular substrings is needed. See also pathForResource:ofType:inDirectory:, pathForResource:ofType:.

principalClass

```
- (Class)principalClass
```

Returns the class object for the first class that’s dynamically loaded by the NSBundle, or nil if the NSBundle can’t dynamically load any classes. See also classNamed:.

resourcePath

```
- (NSString *)resourcePath
```

Returns the directory where pathForResource:ofType:, and other resource path searching methods, looks for a bundle's resources. See also pathForResource:ofType:, pathsForResourcesOfType:inDirectory:.

NSCalendarDate

Characteristic	Description
Inherits From:	NSDate : NSObject
Conforms To:	NSCoding, NSCopying (NSDate) NSObject (NSObject)
Declared In:	Foundation / NSDate.h

Class Description

`NSCalendarDate` is a public subclass of `NSDate` that defines concrete date objects. These objects have time zones and format strings bound to them and are especially suited for representing and manipulating dates according to western calendrical systems.

By drawing on the behavior of the `NSTimeZone` class, `NSCalendarDate` objects adjust their visible representations to reflect their associated time zones. Because of this, you can track an `NSCalendarDate` object across different time zones. You can also present date information from time-zone viewpoints other than the one for the current locale.

Each `NSCalendarDate` object also has a calendar format string bound to it. This format string contains date-conversion specifiers that are very similar to those used in the standard C library function `strftime()`. By reference to this format string, `NSCalendarDate` can interpret dates that are represented as strings conforming to the format. Several methods allow you to specify formats other than the one bound to the object, and `setCalendarFormat:` lets you change the default format string for an `NSCalendarDate` object.

`NSCalendarDate` provides both class and instance methods for obtaining initialized objects. Some of these methods allow you to initialize date objects from strings while others initialize objects from sets of integers corresponding the standard time values (months, hours, seconds, and so on). As always, you are responsible for deallocating any objects obtained through an `alloc...` or `copy...` method.

To retrieve conventional elements of a date, use the methods of the form `dayOfWeek`, `monthOfYear`, and so on. For example, `dayOfWeek` returns a number that indicates the day of the week (0 is Sunday). The `monthOfYear` method returns a number from 1 to 12 that indicates the month.

`NSCalendarDate` provides several methods for representing dates as strings. These methods—`description`, `descriptionWithLocale:`, `descriptionWithCalendarFormat:`, and `descriptionWithCalendarFormat:timeZone:`—take an implicit or explicit format string.

`NSCalendarDate` performs date computations based on western calendar systems, primarily the Gregorian. (The algorithms are derived from public domain software described in "Calendrical Calculations," a two-part series by Nachum Dershowitz and Edward M. Reingold in the book *Software—Practice and Experience*).

General Exceptions

`NSCalendarDate` will raise `NSInvalidArgumentException` in the general case where numeric character strings to specify years, months, days, and so on, are not valid numbers.

Method Types

Activity	Class Method
Getting and initializing an NSCalendar date	+ calendarDate + dateWithString:calendarFormat: + dateWithString:calendarFormat:locale: + dateWithYear:month:day:hour:minute:second: timeZone: – initWithString: – initWithString:calendarFormat: – initWithString:calendarFormat:locale: – initWithYear:month:day:hour:minute:second: timeZone:
Retrieving date elements	– dayOfCommonEra – dayOfMonth – dayOfWeek – dayOfYear – hourOfDay – minuteOfHour – monthOfYear – secondOfMinute – years:months:days:hours:minutes: seconds:sinceDate: – yearOfCommonEra
Providing adjusted dates	– dateByAddingYears:month:day:hour: minute:second:
Getting string descriptions of dates	– description – descriptionWithCalendarFormat: – descriptionWithCalendarFormat:locale: – descriptionWithLocale:
Getting and setting calendar formats	– calendarFormat – setCalendarFormat:
Getting and setting time zones	– setTimeZone: – timeZoneDetail

Class Methods

calendarDate

```
+ (NSCalendarDate *)calendarDate
```

Returns an `NSCalendarDate` initialized to the current date and time. See also `dateWithString:calendarFormat:`, `initWithString:`.

dateWithString:calendarFormat:

```
+ (NSCalendarDate *)dateWithString:(NSString *)description
    calendarFormat:(NSString *)format
```

Returns an `NSCalendarDate` object initialized with the date specified in `description` and interpreted according the the conversion specifiers in `format`. Raises `NSInvalidArgumentException` if the `description` and `format` do not correspond exactly (see the "Class description"). See also `initWithString:calendarFormat:`.

dateWithString:calendarFormat:locale:

```
+ (NSCalendarDate *) dateWithString:(NSString *)description
    calendarFormat:(NSString *)format
    locale:(NSDictionary *)dictionary
```

Returns an `NSCalendarDate` object initialized with the date specified in `description` and interpreted according the the conversion specifiers in `format`. String components of the date are fetched from the locale `dictionary`. This method raises `NSInvalidArgumentException` if the `description` and `format` do not correspond exactly. See also `initWithString:calendarFormat:locale:`.

dateWithYear:month:day:hour:minute:second:timeZone:

```
+ (NSCalendarDate *)dateWithYear:(int)year
    month:(unsigned int)month
    day:(unsigned int)day
    hour:(unsigned int)hour
```

```
minute:(unsigned int)minute
second:(unsigned int)second
timeZone:(NSTimeZone *)aTimeZone
```

Returns an `NSCalendarDate` object initialized with integers that specify a `year` (which must include the century), `month`, `day`, `hour`, `minute`, and `second`. Also include a time-zone object or time-zone detail object (`aTimeZone`) to have the date adjusted to a particular locale. If you specify `nil` for a time zone, `NSInvalidArgumentException` is raised. (See the methods grouped under "Retrieving Date Elements" for the proper ranges of the date and time integers.) See also `initWithYear:month:day:hour:minute:second: timeZone:`, `NSTimeZone`.

Instance Methods

dateByAddingYears:month:day:hour: minute:second:

```
- (NSCalendarDate *)dateByAddingYears:(int)years
    months:(int)months days:(int)days hours:(int)hours
    minutes:(int)minutes seconds:(int)seconds
```

Returns an `NSCalendarDate` object with the `years`, `months`, `days`, `hours`, `minutes`, and `seconds` offsets specified as arguments and the correct time-zone detail object for the computed date. These offsets are relative to the object and can be positive or negative. This method preserves “clock time” during transitions to and from Daylight Savings Time and in leap years.

calendarFormat

```
- (NSString *)calendarFormat
```

Returns the calendar format (a string of date-conversion specifiers) for the receiving object. The default calendar format is “`%Y-%m-%d %H:%M:%S %z`”. See also `setCalendarFormat:`.

dayOfCommonEra

```
- (int)dayOfCommonEra
```

Returns the number of days since the beginning of the Common Era. See also `yearOfCommonEra`.

dayOfMonth

```
- (int)dayOfMonth
```

Returns the day of the month (1 through 31) of the `NSCalendarDate` object. See also `dayOfWeek`, `dayOfYear`.

dayOfWeek

```
- (int)dayOfWeek
```

Returns a number indicating the day of the week (0 [Sun] through 6 [Sat]) of the `NSCalendarDate` object. See also `dayOfMonth`, `dayOfYear`.

dayOfYear

```
- (int)dayOfYear
```

Returns a number indicating the day of the year (1 through 366) of the `NSCalendarDate` object. See also `dayOfWeek`, `dayOfMonth`.

description

```
- (NSString *)description
```

Returns a string description of the receiver's date using the default format string (`%Y-%m-%d %H:%M:%S %z`) and the locale and time-zone information associated with the receiver. See also `initWithString:`, `descriptionWithCalendarFormat:`, `calendarFormat`.

descriptionWithCalendarFormat:

```
- (NSString *)descriptionWithCalendarFormat:(NSString *)format
```

Returns a string description of the receiver's date that is formatted according to the conversion specifiers in `format` and using the locale and time-zone detail information associated with the receiver. See also `initWithString:calendarFormat:`, `description`, `descriptionWithCalendarFormat:locale:`, `calendarFormat`.

descriptionWithCalendarFormat:locale:

```
- (NSString *)descriptionWithCalendarFormat:(NSString *)format
    locale:(NSDictionary *)locale
```

Returns a string description of the receiver's date that is formatted according to the conversion specifiers in `format`, represented according to the locale information in `locale`, and adjusted according to the time-zone detail information associated with the receiver. See also `initWithString:calendarFormat:locale:`, `description`, `calendarFormat`.

descriptionWithLocale:

```
- (NSString *)descriptionWithLocale:(NSDictionary *)locale
```

Returns a string description of the receiver's date using the default format string (`%Y-%m-%d %H:%M:%S %z`), with information localized according to the locale information in `locale`, and using the time zone information associated with the receiver. See also `initWithString:calendarFormat:locale:`.

hourOfDay

```
- (int)hourOfDay
```

Returns a number indicating the hour of the day (0 through 23) of the `NSCalendarDate` object. See also `minuteOfHour`, `secondOfMinute`.

initWithString:

```
- (id)initWithString:(NSString *)description
```

Initializes and returns an `NSCalendarDate` object specified by `description` in the international format for date representation (YYYY-MM-DD HH:MM:SS HHMM, where HHMM is an of fset from GMT). See also `dateWithString:calendarFormat:`, `initWithString:calendarFormat:`.

initWithString:calendarFormat:

```
- (id)initWithString:(NSString *)description
    calendarFormat:(NSString *)format
```

Initializes and returns an NSCalendarDate object specified as a string object in description and interpreted according to the extended strftime() date-conversion specifiers in format. Raises NSInvalidArgumentException if the description and format do not correspond exactly. See also dateWithString:calendarFormat:, initWithString:calendarFormat:locale:.

initWithString:calendarFormat:locale:

```
- (id)initWithString:(NSString *)description
    calendarFormat:(NSString *)format
    locale:(NSDictionary *)dictionary
```

Initializes and returns an NSCalendarDate object specified as a string object in description and interpreted according to the extended strftime() date-conversion specifiers in format. String components of the date are fetched from the locale dictionary. Raises NSInvalidArgumentException if the description and format do not correspond exactly. See also dateWithString:calendarFormat:locale:, initWithString:.

initWithYear:month:day:hour:minute:second:timeZone:

```
- (id)initWithYear:(int)year month:(unsigned int)month
    day:(unsigned int)day
    hour:(unsigned int)hour minute:(unsigned int)minute
    second:(unsigned int)second timeZone:(NSTimeZone *)aTimeZone
```

Returns an NSCalendarDate object initialized with integers that specify a year (which must include the century), month, day, hour, minute, and second. Also include a time-zone object (aTimeZone) to have the date adjusted for a particular locale. Raises an NSInvalidArgumentException if you specify nil for a time zone. See the methods grouped under "RetrievingDate Elements," for the proper ranges of the date and time integers. See also dateWithYear:month:day:hour:minute:second: timeZone:.

minuteOfHour

```
- (int)minuteOfHour
```

Returns a number indicating the minute of the hour (0 through 59) of the NSCalendarDate object. See also secondOfMinute, hourOfDay, monthOfYear.

monthOfYear

```
- (int)monthOfYear
```

Returns a number indicating the month of the year (1 through 12) of the NSCalendarDate object. See also hourOfDay, minuteOfHour, secondOfMinute.

secondOfMinute

```
- (int)secondOfMinute
```

Returns a number indicating the second of the minute (0 through 59) of the NSCalendarDate object. See also hourOfDay.

setCalendarFormat:

```
- (void)setCalendarFormat:(NSString *)format
```

Sets the calendar format for the receiving object to format. See also calendarFormat.

setTimeZone:

```
- (void)setTimeZone:(NSTimeZone *)aTimeZone
```

Sets the time-zone object associated with the NSCalendarDate object to aTimeZone..

timeZoneDetail

```
- (NSTimeZoneDetail *)timeZoneDetail
```

Returns the NSTimeZoneDetail object associated with the receiver. See also setTimeZone:, NSTimeZoneDetail.

yearOfCommonEra

```
- (int)yearOfCommonEra
```

Returns a number indicating the year, including the century, of the `NSCalendarDate` object. See also `dayOfCommonEra`.

years:months:days:hours:minutes:seconds:sinceDate:

```
- (void)years:(int *)yp months:(int *)mop
    days:(int *)dp hours:(int *)hp
    minutes:(int *)mp seconds:(int *)sp
    sinceDate:(NSDate *)date
```

Returns the amount of time since the given `date`.

NSCharacterSet

Characteristic	Description
Inherits From:	NSObject
Conforms To:	NSCoding, NSCopying, NSMutableCopying NSObject (NSObject)
Declared In:	Foundation / NSCharacterSet.h

Class Description

The `NSCharacterSet` class declares the programmatic interface to objects that construct immutable *descriptions* of character sets in the Unicode character encoding. Using `NSCharacterSet` objects, you can determine if a given Unicode character belongs to a specified set. See `NSMutableCharacterSet` for a class that constructs descriptions of character sets that can be modified dynamically. `NSCharacterSet`'s primitive methods are `characterIsMember:` and `bitmapRepresentation`. Subclasses of `NSCharacterSet` must implement these two methods.

`NSCharacterSet` objects can be thought of as loosely analogous to the `is`… macros (such as `isupper()`)available in the `ctype` collection of most standard C libraries. `NSCharacterSet` objects, however, offer much greater flexibility in that you can dynamically construct your own custom character sets against which you can test characters.

Note – The term “bitmap” in the descriptions below does not refer to “bitmap characters” in the sense of screen fonts for display. The “bitmaps” referred to here are compact ordered *bit set* representations of Unicode character positions or ranges of Unicode characters.

You create “standard” character sets—such as a set of alphanumerics, or a set of decimal digits—by invoking the `NSCharacterSet` class object with one of the methods grouped under “Creating a Standard Character Set” in the table below. These methods provide convenient means to create a standard set without needing to specify the character positons explicitly.

You can also create your own “custom” character sets by using one of the methods grouped under “Creating a Custom Character Set” below. To create a character set with multiple disjoint ranges, see the `add`… methods described in `NSMutableCharacterSet`.

Method Types

Activity	Class Method
Creating a standard character set	+ alphanumericCharacterSet + controlCharacterSet + decimalDigitCharacterSet + decomposableCharacterSet + illegalCharacterSet + letterCharacterSet + lowercaseLetterCharacterSet + nonBaseCharacterSet + punctuationCharacterSet + uppercaseLetterCharacterSet + whitespaceAndNewlineCharacterSet + whitespaceCharacterSet
Creating a custom character set	+ characterSetWithBitmapRepresentation: + characterSetWithCharactersInString: + characterSetWithRange:
Getting a binary representation	– bitmapRepresentation – characterIsMember: – invertedSet

Class Methods

alphanumericCharacterSet

```
+ (NSCharacterSet *)alphanumericCharacterSet
```

Returns a character set containing the uppercase and lowercase alphabetic characters (a–z, A–Z, other alphabetic characters such as é, É, ç, Ç, and so on) and the decimal digit characters (0–9). See also `controlCharacterSet`.

characterSetWithBitmapRepresentation:

```
+ (NSCharacterSet *)characterSetWithBitmapRepresentation:
    (NSData *)data
```

Returns a character set containing characters determined by the bitmap representation `data`.

characterSetWithCharactersInString:

```
+ (NSCharacterSet *)characterSetWithCharactersInString:
    (NSString *)aString
```

Returns a character set containing the characters in `aString`. If `aString` is empty, an empty character set is returned. `aString` must not be `nil`.

characterSetWithRange:

```
+ (NSCharacterSet *)characterSetWithRange:(NSRange)aRange
```

Returns a character set containing characters whose Unicode values are given by `aRange`.

controlCharacterSet

```
+ (NSCharacterSet *)controlCharacterSet
```

Returns a character set containing the control characters (characters with decimal Unicode values 0 to 31 and 127 to 159).

decimalDigitCharacterSet

```
+ (NSCharacterSet *)decimalDigitCharacterSet
```

Returns a character set containing only decimal digit characters (0–9).

decomposableCharacterSet

```
+ (NSCharacterSet *)decomposableCharacterSet
```

Returns a character set containing all individual Unicode characters that can also be represented as composed character sequences.

illegalCharacterSet

```
+ (NSCharacterSet *)illegalCharacterSet
```

Returns a character set containing the illegal Unicode values.

letterCharacterSet

```
+ (NSCharacterSet *)letterCharacterSet
```

Returns a character set containing the uppercase and lowercase alphabetic characters (a–z, A–Z, other alphabetic characters such as é, É, ç, Ç, and so on).

lowercaseLetterCharacterSet

```
+ (NSCharacterSet *)lowercaseLetterCharacterSet
```

Returns a character set containing only lowercase alphabetic characters (a–z, other alphabetic characters such as é, ç, and so on).

nonBaseCharacterSet

```
+ (NSCharacterSet *)nonBaseCharacterSet
```

Returns a set containing all characters which are not defined to be base characters for purposes of dynamic character composition.

punctuationCharacterSet

```
+ (NSCharacterSet *)punctuationCharacterSet
```

Returns a character set containing all punctuation characters.

uppercaseLetterCharacterSet

```
+ (NSCharacterSet *)uppercaseLetterCharacterSet
```

Returns a character set containing only uppercase alphabetic characters (A–Z, other alphabetic characters such as É, Ç, and so on).

whitespaceAndNewlineCharacterSet

```
+ (NSCharacterSet *)whitespaceAndNewlineCharacterSet
```

Returns a character set containing only whitespace characters (space and tab) and the newline character. See also `whitespaceCharacterSet`.

whitespaceCharacterSet

```
+ (NSCharacterSet *)whitespaceCharacterSet
```

Returns a character set containing only in-line whitespace characters (space and tab). This set doesn't contain the newline or carriage return characters. See also `whitespaceAndNewlineCharacterSet`.

Instance Methods

bitmapRepresentation

```
- (NSData *)bitmapRepresentation
```

Returns an `NSData` object encoding the receiving character set in binary format. This format is suitable for saving to a file or otherwise transmitting or archiving.

characterIsMember:

```
- (BOOL)characterIsMember:(unichar)aCharacter
```

Returns `YES` if `aCharacter` is in the receiving character set, and returns `NO` if it isn't.

invertedSet

```
- (NSCharacterSet *)invertedSet
```

Returns a character set containing only characters that *don't* exist in the receiver.

5

NSCoder

Characteristic	Description
Inherits From:	NSObject
Conforms To:	NSObject (NSObject)
Declared In:	Foundation / NSCoder.h Foundation / NSGeometry.h

Class Description

`NSCoder` is an abstract class that declares the interface used by subclasses to take objects from dynamic memory and code them into and out of some other format. This capability provides the basis for archiving where objects and other structures are stored on disk, and distribution where objects are copied to different address spaces. See the `NSArchiver` and `NSUnarchiver` class specifications for more information on archiving.

`NSCoder` operates on the basic C and Objective C types—`int`, `float`, `id`, and so on (but excluding `void *` and `union`)—as well as on user-defined structures and pointers to these types (see the type descriptors in the next section).

`NSCoder` declares methods that a subclass can override if it wants:

- To encode or decode an object only under certain conditions, such when it is an intrinsic part of a larger structure (`encodeRootObject:` and `encodeConditionalObject:`)
- To allow decoded objects to be allocated from a specific memory zone (`setObjectZone:`)
- To allow system versioning (`systemVersion`)

`NSCoder` differs from the `NSSerializer` and `NSDeserializer` classes in that `NSCoder`s aren't restricted to operating on property list objects (objects of the `NSData`, `NSString`, `NSArray`, and `NSDictionary` classes). Also, unlike `NSSerializer`s, `NSCoder`s store type information along with the data. Thus, an object decoded from a stream of bytes will be of the same class as the object that was originally encoded into the stream.

Encoding and Decoding Objects

In OpenStep, coding is facilitated by methods declared in several places, most notably the `NSCoder` class, the `NSObject` class, and the `NSCoding` protocol.

Objects are encoded and decoded by using the type descriptors in the following table.d

Descriptor	Type
id	@
Class	#
SEL	:
char	c
unsigned char	C
short	s
unsigned short	S
int	i
unsigned int	I
long	l
unsigned long	L
long long	q
float	f
double	d
bitfield	b
void	v
undefined	?
pointer	^
char *	*
array	[<count><types>]
union	(<types>)
structure	{<types>}

For example, "{sic*@}" represents a structure containing a short, an int, a char, a char *, and an object; the descriptor "[99b]" represents an array containing 99 bitfields.

The `NSCoding` protocol declares the two methods (`encodeWithCoder:` and `initWithCoder:`) that a class must implement so that objects of that class can be encoded and decoded. When an object receives an `encodeWithCoder:` message, it should send a message to `super` to encode inherited instance variables before it encodes the instance variables that its class declares. For example, a fictitious `MapView` class that displays a legend and a map at various magnifications, might implement `encodeWithCoder:` like this:

```
- (void)encodeWithCoder:(NSCoder *)coder
{
    [super encodeWithCoder:coder];//Unless superclass is NSObject
    [coder encodeValuesOfObjCTypes:"si@", &mapName,
            &magnification, &legendView];
}
```

Note – Do not send `super` the `encodeWithCoder:` or `initWithCoder:` messages if the immediate superclass is `NSObject`. Doing so will result in an error.

Objects are decoded in two steps. First, an object of the appropriate class is allocated and then it's sent an `initWithCoder:` message to allow it to initialize its instance variables. Again, the object should first send a message to `super` to initialized inherited instance variables, and then it should initialize its own. `MapView`'s implementation of this method looks like this:

```
- (id)initWithCoder:(NSCoder *)coder
{
   self = [super initWithCoder:coder];//Unless superclass
                                      // is NSObject
   [coder decodeValuesOfObjCTypes:"si@", &mapName, &magnification,
    &legendView];
   return self;
}
```

Note the assignment of the return value of `initWithCoder:` to `self` in the example above. This is done in the subclass because the superclass, in its implementation of `initWithCoder:` may decide to return a object other than itself.

There are other methods that allow an object to customize its response to encoding or decoding. `NSObject` declares these methods:

Table 5-2 Customizing the Encoding and Decoding of Objects

Method	Typical Use
classForCoder:	Allows an object, when being encoded, to substitute a class other than its own. For example, the private subclasses of a class cluster substitute the name of their public superclass when being archived.
replacementObjectForCoder:	Allows an object, when being encoded, to substitute another object for itself. For example, an object might encode itself into an archive, but encode a proxy for itself if it's being encoded for distribution.
awakeAfterUsingCoder:	Allows an object, when being decoded, to substitute another object for itself. For example, an object that represents a font might, upon being decoded, release itself and return an existing object having the same font description as itself. In this way, redundant objects can be eliminated.

See the `NSObject` class specification for more information. See also `NSArchiver`, `NSUnarchiver`, `NSSerializer`, `NSDeserializer`.

Note – Because it's an abstract class, most `NSCoder` methods raise an `NSInvalidArgumentException` and return `nil` where appropriate.

Method Types

Activity	Class Method
Encoding data	– encodeArrayOfObjCType:count:at: – encodeBycopyObject: – encodeBytes:length: – encodeConditionalObject: – encodeDataObject: – encodeObject: – encodePropertyList: – encodePoint: – encodeRect: – encodeRootObject: – encodeSize: – encodeValueOfObjCType:at: – encodeValuesOfObjCTypes:
Decoding data	– decodeArrayOfObjCType:count:at: – decodeBytesWithReturnedLength: – decodeDataObject – decodeObject – decodePropertyList – decodePoint – decodeRect – decodeSize – decodeValueOfObjCType:at: – decodeValuesOfObjCTypes:
Managing zones	– objectZone – setObjectZone:
Getting a version	– systemVersion – versionForClassName:

Instance Methods

decodeArrayOfObjCType:count:at:

```
- (void)decodeArrayOfObjCType:(const char *)types
    count:(unsigned)count
    at:(void *)address
```

Decodes data of Objective C types listed in `types` having `count` elements residing at `address`. See the class description for a list of value `types` can be.

decodeBytesWithReturnedLength:

```
(void *) decodeBytesWithReturnedLength: (unsigned *) length
```

Decodes `length` number of bytes. See also `encodeBytes:length:`.

decodeDataObject

```
- (NSData *)decodeDataObject
```

Decodes and returns an `NSData` object. See also `encodeDataObject:`.

decodeObject

```
- (id)decodeObject
```

Decodes an Objective C object. See also `encodeObject:`.

decodePoint

```
- (NSPoint)decodePoint
```

Decodes a point structure. See also `encodePoint:`.

decodePropertyList

```
- (id)decodePropertyList
```

Decodes a property list (`NSData`, `NSArray`, `NSDictionary`, or `NSString` objects). See also `encodePropertyList:`.

decodeRect

```
- (NSRect)decodeRect
```

Decodes a rectangle structure. See also `encodeRect:`.

decodeSize

```
- (NSSize)decodeSize
```

Decodes a size structure. See also `encodeSize:`.

decodeValueOfObjCType:at:

```
- (void)decodeValueOfObjCType:(const char *)type at:(void *)address
```

Decodes data of the specified Objective C `type` into `address`. You are responsible for releasing the resulting objects. `type` can be a type descriptor described in the class description. See also `encodeValueOfObjCType:at:`.

decodeValuesOfObjCTypes:

```
- (void)decodeValuesOfObjCTypes:(const char *)types,...
```

Decodes values corresponding to the Objective C types listed in `types` into the following argument list. You are responsible for releasing the resulting objects. See the class description for an example, and a list of type descriptors that `types` can be. See also `encodeValuesOfObjCTypes:`.

encodeArrayOfObjCType:count:at:

```
- (void)encodeArrayOfObjCType:(const char *)types
    count:(unsigned int)count at:(const void *)array
```

Encodes data of Objective C types listed in `types` having `count` elements residing at address `array`. See the class description for a list of type descriptors that `types` can be. See also `encodeArrayOfObjCType:count:at:` (`NSArchiver`).

encodeBycopyObject:

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

Overridden by subclasses to encode the supplied Objective C object so that a copy rather than a proxy of `anObject` is created upon decoding. `NSCoder`'s implementation simply invokes `encodeObject:`.

encodeBytes:length:

```
(void) encodeBytes:(void*)byteAddress (unsigned *)length
```

Encodes `length` number of bytes, located at `byteAddress`. See also `decodeBytesWithReturnedLength:`.

encodeConditionalObject:

```
– (void)encodeConditionalObject:(id)anObject
```

Overridden by subclasses to conditionally encode the supplied Objective C object. The object should be encoded only if it is an intrinsic member of the larger data structure. `NSCoder`'s implementation simply invokes `encodeObject:`.

encodeDataObject:

```
– (void)encodeDataObject:(NSData *)data
```

Encodes the `NSData` object `data`. See also `decodeDataObject`.

encodeObject:

```
– (void)encodeObject:(id)anObject
```

Encodes the supplied Objective C object. See also `decodeObject`.

encodePoint:

```
– (void)encodePoint:(NSPoint)point
```

Encodes the supplied point structure. See also `decodePoint`.

encodePropertyList:

```
– (void)encodePropertyList:(id)aPropertyList
```

Encodes the supplied property list (`NSData`, `NSArray`, `NSDictionary`, or `NSString` objects). See also `decodePropertyList`.

encodeRect:

```
- (void)encodeRect:(NSRect)rect
```

Encodes the supplied rectangle structure. See also `decodeRect`.

encodeRootObject:

```
- (void)encodeRootObject:(id)rootObject
```

Overridden by subclasses to start encoding an interconnected group of Objective C objects, starting with `rootObject`. `NSCoder`'s implementation simply invokes `encodeObject:`.

encodeSize:

```
- (void)encodeSize:(NSSize)size
```

Encodes the supplied size structure. See also `decodeSize`.

encodeValueOfObjCType:at:

```
- (void)encodeValueOfObjCType:(const char *)type
    at:(const void *)address
```

Encodes data of the specified Objective C `type` residing at `address`. See the class description for a list of type descriptors that `type` can be. See also `decodeValueOfObjCType:at:`.

encodeValuesOfObjCTypes:

```
- (void)encodeValuesOfObjCTypes:(const char *)types,...
```

Encodes values corresponding to the Objective C types listed in types argument list. See the class description for a list of type descriptors that `type` can be. See also `decodeValuesOfObjCTypes:`, `encodeValueOfObjCType:at:`.

objectZone

```
- (NSZone *)objectZone
```

Returns the memory zone used by decoded objects. For instances of `NSCoder`, this is the default memory zone, the one returned by `NSDefaultMallocZone()`. See also `setObjectZone:`.

setObjectZone:

```
- (void)setObjectZone:(NSZone *)zone
```

Sets the memory zone used by decoded objects. Instances of `NSCoder` always use the default memory zone, the one returned by `NSDefaultMallocZone()` (see the FoundationKit's "Functions" chapter), so ignore this method. See also `objectZone`.

systemVersion

```
- (unsigned int)systemVersion
```

Returns the system version number as of the time the archive was created. The default implementation returns 1000. See also `versionForClassName:`.

versionForClassName:

```
- (unsigned int)versionForClassName:(NSString *)className
```

Returns the version number of the class `className` as of the time it was archived. The default implementation returns 0. See also `systemVersion`.

5

NSConditionLock

Characteristic	Description
Inherits From:	NSObject
Conforms To:	NSLocking NSObject (NSObject)
Declared In:	Foundation / NSLock.h

Class Description

`NSConditionLock` objects are used to lock and unlock threads when specified conditions occur. The user of an `NSConditionLock` object can lock when a process enters a particular state and can set the state to something else when releasing the lock. The states are defined by the lock's user. `NSConditionLock` is well suited to synchronizing different modules such as a producer and a consumer where the two modules must share data, but the consumer must sleep until a condition is met such as more data being available.

The `NSConditionLock` class provides four ways of locking its objects (`lock`, `lockWhenCondition:`, `tryLock`, and `tryLockWhenCondition`) and two ways of unlocking (`unlock` and `unlockWithCondition:`). Any combination of locking method and unlocking method is legal.

The following example shows how the producer-consumer problem might be handled using condition locks. The producer need not wait for a condition, but must wait for the lock to be made available so it can safely create shared data. For example, a producer could use a lock this way:

```
/* create the lock only once */
id condLock = [NSConditionLock new];

[condLock lock];
/* Manipulate global data... */
[condLock unlockWithCondition:HAS_DATA];
```

Multiple consumer threads can then lock until there's data available and everyone is out of locked critical sections. In the following code sample, the consumer sleeps until the producer invokes `unlockWithCondition:` with the parameter `HAS_DATA`:

```
[condLock lockWhenCondition:HAS_DATA];
/* Manipulate global data if necessary... */
[condLock unlockWithCondition:(moreData ? HAS_DATA : NO_DATA)];
```

An `NSConditionLock` object doesn't busy-wait, so it can be used to lock time-consuming operations without degrading system performance.

The `NSConditionLock`, `NSLock`, and `NSRecursiveLock` classes all implement the `NSLocking` protocol with various features and performance characteristics; see the other class descriptions for more information.

Method Types

Activity	Class Method
Initializing an NSConditionLock	– initWithCondition:
Returning the condition	– condition
Acquiring and releasing a lock	– lockWhenCondition: – unlockWithCondition: – tryLock – tryLockWhenCondition:

Instance Methods

`condition`

`- (int)condition`

Returns the receiver's condition, the state that must be achieved before a conditional lock can be acquired or released. This condition can be set with `intWithCondition:` or `unlockWithCondition:`.

`initWithCondition:`

`- (id)initWithCondition:(int)condition`

Initializes a newly created `NSConditionLock` and sets its condition to `condition`. This message should not be sent to an already initialized instance. See also `condition`.

lockWhenCondition:

```
- (void)lockWhenCondition:(int)condition
```

Waits until the lock isn't in use, and the lock's condition matches `condition`, then grabs the lock. The lock's condition can be set with `initWithCondition:` or `unlockWithCondition:`. The lock can subsequently be released with `unlockWithCondition:`.

tryLock

```
- (BOOL)tryLock
```

Attempts to acquire a lock. Returns `YES` if successful and `NO` otherwise.

tryLockWhenCondition:

```
- (BOOL)tryLockWhenCondition:(int)condition
```

Attempts to acquire a lock when `condition` is met. Returns `YES` if successful and `NO` otherwise.

unlockWithCondition:

```
- (void)unlockWithCondition:(int)condition
```

Releases the lock and sets lock state to `condition`.

NSConnection

Characteristic	Description
Inherits From:	NSObject
Conforms To:	NSObject (NSObject)
Declared In:	Foundation / NSConnection.h

Class Description

When objects in different threads can send messages to each other, these objects have a *connection*. The `NSConnection` class describes objects that manage a connection (typically, in another process), and defines instances that manage each side of such a connection.

Each distinct thread of execution has one default connection defined. Any given thread can have as many connections as desired, but a given connection can be served by only one thread.

To set up a connection, some object in your application must be established as what is known as a “root” object and registered with a name in the Network Name Server. Such root objects can then be connected to by other threads, and can receive messages sent to them from other threads. An easy way to establish an object as a root object is to send the `defaultConnection` method to the `NSConnection` class object to obtain a connection object. Then use `setRootObject:` to establish the desired object as the object that will be registered, followed by `registerName:` to make that object available to the Network Name Server under the specified name.

To obtain a connection to an object registered elsewhere, you will generally send the `rootProxyForConnectionWithRegisteredName:host:` method to the `NSConnection` class object. This method returns a proxy to the remote object. You should then use `setProtocolForProxy:` to inform the proxy about the protocols the remote object responds to. To obtain the actual connection object instead of the proxy, use the `connectionWithRegisteredName:host:` method.

If the string `@"*"` is used where a hostname is required, it implies a lookup for any server registered with the specified name on the local subnet. If `nil` is supplied where a hostname is required, the name lookup occurs only on the local host.

When an `NSConnection` object is deallocated, the notification `NSConnectionDeath` is posted to the default notification center with that `NSConnection` object.

Exceptions

`NSConnection` can raise `NSInternalInconsistencyException` for a variety of reasons when it detects "impossible" situations. In addition, `NSConnection` can raise `NSInvalidArgumentException` when a remote method invocation sends an unknown selector.

Method Types

Activity	Class Method
Initializing a connection	– init
Establishing a connection	+ connectionWithRegisteredName:host: + defaultConnection +rootProxyForConnectionWithRegisteredName:host:
Determining connections	+ allConnections – invalidate – isValid
Registering a connection	– registerName:
Assigning a delegate	– delegate – setDelegate:
Getting and setting the root object	– rootObject – rootProxy – setRootObject:
Request mode	– addRequestMode: – removeRequestMode: – requestModes
Conversation queueing	– independentConversationQueueing – setIndependentConversationQueueing:
Timeouts	– replyTimeout – requestTimeout – setReplyTimeout: – setRequestTimeout:
Get statistics	– statistics
Methods Implemented by the Delegate	– makeNewConnection:sender:

Class Methods

allConnections

```
+ (NSArray *)allConnections
```

Returns an array containing all existing valid connections. See also `isValid`.

`connectionWithRegisteredName:host:`

```
+ (NSConnection *)connectionWithRegisteredName:(NSString *)name
    host:(NSString *)hostName
```

Registers and returns a connection with `name` on `hostName`, or returns `nil` if no connection can be established. If `hostName` is specified, this method queries the Network Name Server on `hostName` for the object registered under `name`. If `hostName` is `nil`, this method queries the Network Name Server on the local host. If `hostName` is "`*`", this method will query the Network Name Server on each machine on the subnet until it finds an object under `name`.

`defaultConnection`

```
+ (NSConnection *)defaultConnection
```

Establishes and returns a default per-thread connection.

`rootProxyForConnectionWithRegisteredName:host:`

```
+ (NSDistantObject *)rootProxyForConnectionWithRegisteredName:
    (NSString *)name host:(NSString *)hostName
```

Registers a connection with `name` on `hostName` and returns its root proxy. See also `NSDistantObject`.

Instance Methods

`addRequestMode:`

```
- (void)addRequestMode:(NSString *)rmode
```

Adds request mode rmode to the list of modes the connection honors. See also `removeRequestMode:`, `requestModes`.

`delegate`

```
– (id)delegate
```

Returns the connection's delegate.

independentConversationQueueing

```
-(BOOL)independentConversationQueueing
```

Returns conversationQueuing mode. The default value is NO. See also setIndependentConversationQueueing:.

init

```
- (id)init
```

Initializes a newly allocated NSConnection suitable for a new registry and new name.

invalidate

```
- (void)invalidate
```

Invalidates the receiving connection object. See also isValid.

isValid

```
- (BOOL)isValid
```

Identifies YES if the receiver is a valid connection, and returns NO otherwise. See also invalidate.

registerName:

```
- (BOOL)registerName:(NSString *)name
```

Registers the connection with name on the local system and returns YES if the registration was successful, NO otherwise.

removeRequestMode:

```
- (void)removeRequestMode:(NSString *)rmode
```

Removes rmode from the list of modes that the connection honors. See also addRequestMode:, requestModes.

replyTimeout

```
- (NSTimeInterval)replyTimeout
```

Returns the reply timeout time interval.

requestModes

```
- (NSArray *)requestModes
```

Returns the mode in which requests are honored. See also `addRequestMode:`.

requestTimeout

```
- (NSTimeInterval)requestTimeout
```

Returns the request timeout time interval. See also `setRequestTimeout:`, `replyTimeout`.

rootObject

```
- (id)rootObject
```

Returns the root object served (see the “Class Description”). See also `rootProxy`.

rootProxy

```
- (NSDistantObject *)rootProxy
```

Returns an `NSDistantObject` proxy to the root object served by this connection. See also `rootObject`.

setDelegate:

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

Sets the connection’s delegate. See also `delegate`.

setIndependentConversationQueueing:

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

If `flag` is `YES`, unrelated requests are queued for later processing. This allows a server to use distributed objects freely in its implementation without concern for the consistency of its internal state. Note that this can cause deadlocks among peers. See also `independentConversationQueueing`.

`setReplyTimeout:`

```
- (void)setReplyTimeout:(NSTimeInterval)interval
```

Sets the reply timeout to the time interval `interval`.

`setRequestTimeout:`

```
- (void)setRequestTimeout:(NSTimeInterval)interval
```

Sets the request timeout to the time interval `interval`.

`setRootObject:`

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

Sets the root object being served to `anObject`; if the root object already exists, replaces it with `anObject`. Be aware that if the root object is replaced while a connection is active, existing root proxies on the client side of the connection will continue to communicate with the previous root object, while new proxies will communicate with the newly established root object.

`statistics`

```
- (NSDictionary *)statistics
```

Returns statistics for this connection.

Methods Implemented by the Delegate

`makeNewConnection:sender:`

```
- (BOOL)makeNewConnection:(NSConnection *)connection
    sender:(NSConnection *)ancestor
```

Asks permission to create a new connection `connection` where `ancestor` is the ancestral connection; returns `YES` if `connection` allowed.

NSCountedSet

Characteristic	Description
Inherits From:	NSMutableSet : NSSet : NSObject
Conforms To:	NSCoding, NSCopying, NSMutableCopying (NSSet) NSObject (NSObject)
Declared In:	Foundation / NSSet.h

Class Description

The `NSCountedSet` class declares the programmatic interface to an object that manages a mutable set of objects. `NSCountedSet` provides support for the mathematical concept of a *counted set*. A counted set, both in its mathematical sense and in the OpenStep implementation of `NSCountedSet`, is an unordered collection of elements, just as in a regular set, but the elements of the set aren't necessarily distinct. In the literature, a counted set is also knownas a *bag*.

Each new—that is, distinct—object inserted into an `NSCountedSet` object has a counter associated with it. `NSCountedSet` keeps track of the number of times objects are inserted and requires that objects are removed the same number of times. OpenStep also provides the `NSSet` class for sets whose elements *are* distinct—that is, there is only one instance of an object in an `NSSet` even if the object has been added to the set multiple times.

Use set objects as an alternative to array objects when the order of elements is not important, but performance in testing whether an object is contained in the set *is* a consideration—while arrays are ordered, testing for membership is slower than with sets.

Objects in a set must respond to `hash` and `isEqual:` methods. See the `NSObject` protocol for details on `hash` and `isEqual:`. Each new distinct object must provide a unique hash value.

Generally, you instantiate an `NSCountedSet` object by sending one of the `set…` methods to the `NSCountedSet` class object, as described in `NSSet`. These methods return an `NSCountedSet` object containing the elements (if any) you pass in as arguments. Newly created instances of `NSCountedSet` created by

invoking the `set` method can be populated with objects using any of the `init…` methods. The designated initializer for this class is `initWithObjects:` (`NSSet`).

You add or remove objects from a counted set using the `addObject:` and `removeObject:` methods.

An `NSCountedSet` may be queried using the `objectEnumerator` method, which provides for traversing elements of the set one by one. The `countForObject:` method returns the number of times the specified object has been added to this set.

Method Types

Activity	Class Method
Initializing an NSCountedSet	– initWithArray: – initWithCapacity: – initWithSet:
Adding objects	– addObject:
Removing objects	– removeObject:
Querying the NSCountedSet	– countForObject: – objectEnumerator

Instance Methods

`addObject:`

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

Adds `anObject` to the set, unless `anObject` is equal to some object already in the set. In either case, the counter that's returned by `countForObject:` is incremented.

`countForObject:`

```
- (unsigned int)countForObject:(id)anObject
```

Returns the number of times that an object equal to `anObject` has ostensibly been added to the set. (This number is incremented by `addObject:` and decremented by `removeObject:`.)

initWithArray:

```
- (id)initWithArray:(NSArray *)anArray
```

Initializes a newly allocated set object by placing in it the objects contained in `anArray`.

initWithCapacity:

```
- (id)initWithCapacity:(unsigned int)numItems
```

Initializes a newly allocated set object, giving it enough memory to hold `numItems` objects.

initWithSet:

```
- (id)initWithSet:(NSSet *)anotherSet
```

Initializes a newly allocated set object by placing in it the objects contained in `anotherSet`.

objectEnumerator

```
- (NSEnumerator *)objectEnumerator
```

Returns an enumerator object that will access each object in the set only once, regardless of its count.

removeObject:

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

Decrements the counter for the object if the set contains an object that's equal to `anObject`. If this causes the counter to reach zero, the object that's equal to `anObject` is removed from the set.

NSData

Characteristic	Description
Inherits From:	NSObject
Conforms To:	NSCoding, NSCopying, NSMutableCopying NSObject (NSObject)
Declared In:	Foundation / NSData.h

Class Description

The `NSData` class declares the interface to objects that contain bytes. `NSData` objects hold a static collection of bytes; `NSData`'s subclass, `NSMutableData`, defines objects that hold modifiable data. These two classes provide an object-oriented approach to memory allocation, a facility that in procedural programming is accessed through functions like `malloc()`. Furthermore, these classes take advantage of operating system primitives when allocating large blocks of memory.

`NSData`'s two primitive methods—`bytes` and `length`—provide the basis for all the other methods in its interface. The `bytes` method returns a pointer to the bytes contained in the data object. `length` returns the number of bytes contained in the data object.

`NSData` and `NSMutableData` objects are commonly used to hold the contents of a file. The methods `dataWithContentsOfFile:` and `dataWithContentsOfMappedFile:` return objects that represent a file's contents. The `writeToFile:atomically:` method enables you to write the contents of a data object to a file.

`NSData` provides access methods for copying bytes from a data object into a buffer. Use `getBytes:` to copy the entire contents of the object or `getBytes:length:` to copy a subset, starting with the first byte. `getBytes:range:` copies a range of bytes from a starting point within the bytes themselves. You can also return a data object that contains a subset of the bytes in another data object by using the `subdataWithRange:` method. Or, you can use the `description` method to return an `NSString` representation of the bytes in a data object.

For determining if two data objects are equal, `NSData` provides the `isEqualToData:` method, which does a byte-for-byte comparison.

Method Types

Activity	Class Method
Allocating and initializing an NSData object	+ allocWithZone: + data + dataWithBytes:length: + dataWithBytesNoCopy:length: + dataWithContentsOfFile: + dataWithContentsOfMappedFile: – initWithBytes:length: – initWithBytesNoCopy:length: – initWithContentsOfFile: – initWithContentsOfMappedFile: – initWithData:
Accessing data	– bytes – description – getBytes: – getBytes:length: – getBytes:range: – subdataWithRange:
Querying a data object	– isEqualToData: – length
Storing data	– writeToFile:atomically:
Deserializing data	– deserializeAlignedBytesLengthAtCursor: – deserializeBytes:length:atCursor: – deserializeDataAt:ofObjCType:atCursor:context: – deserializeIntAtCursor: – deserializeIntAtIndex: – deserializeInts:count:atCursor: – deserializeInts:count:atIndex:

Class Methods

`allocWithZone:`

```
+ (id)allocWithZone:(NSZone *)zone
```

Creates and returns an uninitialized object from `zone`. If `zone` is `NULL`, the default alloction zone is used. See also `data`, `NSAllocateObject()`.

data

```
+ (id)data
```

Creates and returns an empty object. This method is declared primarily for mutable subclasses of NSData. See also dataWithBytes:length:.

dataWithBytes:length:

```
+ (id)dataWithBytes:(const void *)bytes length:(unsigned int)length
```

Creates and returns an object containing length bytes of data copied from the buffer bytes. See also dataWithBytesNoCopy:length:, initWithBytes:length:.

dataWithBytesNoCopy:length:

```
+ (id)dataWithBytesNoCopy:(void *)bytes length:(unsigned int)length
```

Creates and returns an object containing length bytes from the buffer bytes. The NSData object will take over ownership of the bytes, and free()them when the NSData object is deallocated. See also dataWithBytes:length:, initWithBytesNoCopy:length:.

dataWithContentsOfFile:

```
+ (id)dataWithContentsOfFile:(NSString *)path
```

Creates and returns an object by reading data from the file specified by path. Returns nil if the file cannot be found. See also dataWithContentsOfMappedFile:.

dataWithContentsOfMappedFile:

```
+ (id)dataWithContentsOfMappedFile:(NSString *)path
```

Creates and returns an object whose contents come from the mapped file path, assuming mapped files are available on the underlying operating system. Returns nil if the file cannot be found. If mapped files are not available, this method is identical to dataWithContentsOfFile:.

Instance Methods

bytes

```
- (const void *)bytes
```

Returns a pointer to the object's contents. This method returns read-only access to the data. See also `getBytes:`.

description

```
- (NSString *)description
```

Returns an `NSString` object that contains a hexadecimal representation of the receiver's contents.

deserializeAlignedBytesLengthAtCursor:

```
- (unsigned int)deserializeAlignedBytesLengthAtCursor:
    (unsigned int*)cursor
```

Returns the length of the serialized bytes at the location referenced by `cursor`. If the bytes have been page-aligned, it also obtains the relevant "hole" information and adjusts the cursor. An invocation of this method must have a corresponding `serializeAlignedBytesLength:` invocation. See also `deserializeBytes:length:atCursor:`, `deserializeDataAt:ofObjCType:atCursor:context:`.

deserializeBytes:length:atCursor:

```
- (void)deserializeBytes:(void *)buffer length:(unsigned int)bytes
    atCursor:(unsigned int*)cursor
```

Deserializes `bytes` number of bytes in the buffer pointed at by `buffer`, places them internally starting at `cursor`, and advances the `cursor`.

deserializeDataAt:ofObjCType:atCursor:context:

```
- (void)deserializeDataAt:(void *)data ofObjCType:(const char
*)type
    atCursor:(unsigned int*)cursor
    context:(id <NSObjCTypeSerializationCallBack>)callback
```

Deserializes the data pointed at by `cursor`, interpreting it by the Objective C type specifier `type` and writing it to the memory location referenced by `data`. If the data element is an object other than an instance of `NSDictionary`, `NSArray`, `NSString`, or `NSData`, a callback from object `callback` can provide further definition of the object. All Objective C types are currently supported except `union` and `void *`. Pointers refer to a single item.

deserializeIntAtCursor:

```
- (int)deserializeIntAtCursor:(unsigned int*)cursor
```

Deserializes and returns the integer encoded at `cursor`. Also advances the cursor.

deserializeIntAtIndex:

```
- (int)deserializeIntAtIndex:(unsigned int)index
```

Deserializes and returns the integer encoded at offset `index`. Does not advance the cursor.

deserializeInts:count:atCursor:

```
- (void)deserializeInts:(int *)intBuffer
    count:(unsigned int)numInts
    atCursor:(unsigned int*)cursor
```

Deserializes `numInts` integers encoded at the location referenced by `cursor` and puts them in the buffer `intBuffer`. Also advances the cursor.

deserializeInts:count:atIndex:

```
- (void)deserializeInts:(int *)intBuffer
    count:(unsigned int)numInts
    atIndex:(unsigned int)index
```

Deserializes `numInts` integers encoded at offset `index` and puts them in the buffer `intBuffer`. Does not advance the cursor.

getBytes:

```
- (void)getBytes:(void *)buffer
```

Copies the receiver's contents into buffer.

getBytes:length:

```
- (void)getBytes:(void *)buffer length:(unsigned int)length
```

Copies length bytes of the receiver's contents into buffer.

getBytes:range:

```
- (void)getBytes:(void *)buffer range:(NSRange)aRange
```

Copies into buffer the portion of the receiver's contents within aRange. Raises NSRangeException if aRange is not within the range of the receiver's data.

initWithBytes:length:

```
- (id)initWithBytes:(const void *)bytes length:(unsigned int)length
```

Initializes a newly allocated NSData object by putting in it length bytes of data copied from the buffer bytes. See also initWithBytesNoCopy:length:, dataWithBytes:length:.

initWithBytesNoCopy:length:

```
- (id)initWithBytesNoCopy:(void *)bytes length:(unsigned int)length
```

Initializes a newly allocated NSData object by putting in it length bytes of data from the buffer bytes. The NSData object will take over ownership of the bytes, and free()them when the NSData object is deallocated. See also initWithBytes:length:, dataWithBytesNoCopy:length:.

initWithContentsOfFile:

```
- (id)initWithContentsOfFile:(NSString *)path
```

Initializes a newly allocated NSData object by reading into it the data from the file specified by path. Returns nil if path cannot be found.

initWithContentsOfMappedFile:

```
- (id)initWithContentsOfMappedFile:(NSString *)path
```

Initializes a newly allocated NSData object to contain the data residing in the mapped file path, assuming mapped files are available on the underlying operating system. If mapped files are not available, this method is identical to initWithContentsOfFile:.

initWithData:

```
- (id)initWithData:(NSData *)data
```

Initializes a newly allocated NSData object by placing in it the contents of another NSData object, data.

isEqualToData:

```
- (BOOL)isEqualToData:(NSData *)other
```

Compares the receiving object to other. If the contents of other are equal to the contents of the receiver, this method returns YES, otherwise it returns NO.

length

```
- (unsigned int)length
```

Returns the number of bytes contained in the receiver.

subdataWithRange:

```
- (NSData *)subdataWithRange:(NSRange)aRange
```

Returns an object containing a copy of the receiver's bytes that fall within the limits specified by aRange. Raises an NSRangeException if aRange is not within the range of the receiver's data.

writeToFile:atomically:

```
- (BOOL)writeToFile:(NSString *)path
    atomically:(BOOL)useAuxiliaryFile
```

Writes the bytes in the receiving object to the file specified by `path`. If `useAuxiliaryFile` is `YES`, the data is written to a backup file and then, assuming no errors occur, the backup file is renamed atomically to the intended file name.

NSDate

Characteristic	Description
Inherits From:	NSObject
Conforms To:	NSCoding, NSCopying NSObject (NSObject)
Declared In:	Foundation / NSDate.h

Class Description

`NSDate` is an abstract class that provides behavior for creating dates, comparing dates, representing dates, computing time intervals, and similar functionality. It presents a programmatic interface through which suitable date objects are requested and returned. `NSDate` objects are lightweight and immutable since they represent an invariant point in time. This class is designed to provide the foundation for arbitrary calendrical representations. Its subclass `NSCalendarDate` offers date objects that are suitable for representing dates according to western calendrical systems.

"Date" as used here implies clock time as well. The standard unit of time for date objects is a value typed as `NSTimeInterval` (a `double`) and expressed as seconds. The `NSTimeInterval` type makes possible a wide and fine-grained range of date and time values, giving accuracy within milliseconds for dates 10,000 years apart.

`NSDate` and its subclasses compute time as seconds *relative* to an absolute reference date. This reference date is the first instant of January 1, 2001. `NSDate` converts all date and time representations to and from `NSTimeInterval` values that are relative to this absolute reference date. A positive interval relative to a date represents a point in the future, a negative interval represents a time in the past.

Conventional UNIX systems implement time according to the Network Time Protocol (NTP) standard, which is based on Coordinated Universal Time. The private implementation of NSDate follows the NTP standard. However, this standard doesn't account for leap seconds and therefore isn't synchronized with International Atomic Time (the most accurate).

Like various other Foundation Kit classes, `NSDate` lets you obtain operating system functionality (dates and times) without depending on operating system internals. It also provides a basis for the `NSRunLoop` and `NSTimer` classes, which use concrete date objects to implement local event loops and timers.

`NSDate`'s sole primitive method, `timeIntervalSinceReferenceDate`, provides the basis for all the other methods in the `NSDate` interface. It returns a time value relative to an absolute reference date.

Using NSDate

The date objects dispensed by `NSDate` give you a diverse range of date and time functionality. To obtain dates, send one of the `date...` messages to the `NSDate` class object. One of the most useful is `date` itself, which returns a date object representing the current date and time. You can get new date objects with date and time values adjusted from existing date objects by sending `addTimeInterval:`.

You can obtain relative date information by sending the `timeInterval...` messages to a date object. For instance, `timeIntervalSinceNow` gives you the time, in seconds, between the current time and the receiving date object. Compare dates with the `isEqualToDate:`, `compare:`, `laterDate:`, and `earlierDate:` methods and use the `description` method to obtain a string object that represents the date in a standard international format.

Method Types

Activity	Class Method
Creating an NSDate Object	+ allocWithZone: + date + dateWithTimeIntervalSinceNow: + dateWithTimeIntervalSince1970: + dateWithTimeIntervalSinceReferenceDate: + distantFuture + distantPast – init – initWithString: – initWithTimeInterval:sinceDate: – initWithTimeIntervalSinceNow: – initWithTimeIntervalSinceReferenceDate:
Converting to an NSCalendar object	– dateWithCalendarFormat:timeZone:
Representing dates	– description – descriptionWithCalendarFormat:timeZone:locale: – descriptionWithLocale:
Adding and getting intervals	+ timeIntervalSinceReferenceDate – addTimeInterval: – timeIntervalSince1970 – timeIntervalSinceDate: – timeIntervalSinceNow – timeIntervalSinceReferenceDate
Comparing dates	– compare: – earlierDate: – isEqualToDate: – laterDate:

Class Methods

`allocWithZone:`

```
+ (id)allocWithZone:(NSZone *)zone
```

Allocates an unitialized `NSDate` in `zone`. Returns `nil` if allocation fails. See also `date`, `init`.

date

```
+ (NSDate *)date
```

Creates and returns an NSDate set to the current date and time. See also dateWithTimeIntervalSinceNow:, dateWithTimeIntervalSince1970:, dateWithTimeIntervalSinceReferenceDate:, timeIntervalSinceReferenceDate, distantFuture, distantPast, init.

dateWithTimeIntervalSinceNow:

```
+ (NSDate *)dateWithTimeIntervalSinceNow:(NSTimeInterval)seconds
```

Creates and returns an NSDate set to seconds seconds from the current date and time. See also timeIntervalSinceNow, dateWithTimeIntervalSince1970:, dateWithTimeIntervalSinceReferenceDate:.

dateWithTimeIntervalSince1970:

```
+ (NSDate *)dateWithTimeIntervalSince1970:(NSTimeInterval)seconds
```

Creates and returns an NSDate set to to seconds seconds from the reference date used by UNIX® systems. Use a negative argument value to specify a date and time before the reference date. See also timeIntervalSince1970, dateWithTimeIntervalSinceNow:.

dateWithTimeIntervalSinceReferenceDate:

```
+ (NSDate *)dateWithTimeIntervalSinceReferenceDate:
    (NSTimeInterval)seconds
```

Creates and returns an NSDate set to seconds seconds from the absolute reference date (the first instant of 1 January, 2001). Use a negative argument value to specify a date and time before the reference date. See also timeIntervalSinceReferenceDate (class method), timeIntervalSinceReferenceDate (instance method), dateWithTimeIntervalSinceNow:.

distantFuture

```
+ (NSDate *)distantFuture
```

Creates and returns an `NSDate` that represents a date in the distant future in terms of centuries. You can use this object in your code as a control date, a guaranteed outer temporal limit. See also `distantPast`.

distantPast

```
+ (NSDate *)distantPast
```

Creates and returns an `NSDate` that represents a date in the distant past in terms of centuries. You can use this object in your code as a control date, a guaranteed temporal boundary. See also `distantFuture`.

timeIntervalSinceReferenceDate

```
+ (NSTimeInterval)timeIntervalSinceReferenceDate
```

Returns the interval between the system's absolute reference date and the current date and time. This value is less than zero until the first instant of 1 January 2001. See also `timeIntervalSinceReferenceDate` (instance method), `timeIntervalSince1970`, `timeIntervalSinceDate:`, `timeIntervalSinceNow`, `date`.

Instance Methods

addTimeInterval:

```
- (NSDate *)addTimeInterval:(NSTimeInterval)seconds
```

Returns an `NSDate` that's set to a specified number of seconds relative to the receiver. See also `timeIntervalSinceReferenceDate` (class method).

compare:

```
- (NSComparisonResult)compare:(NSDate *)anotherDate
```

Compares the receiver's date to that of `anotherDate` and returns `NSOrderedDescending` if the receiver is temporally later, `NSOrderedAscending` if the receiver is temporally earlier, and `NSOrderedSame` if they are equal.

dateWithCalendarFormat:timeZone:

```
- (NSCalendarDate *)dateWithCalendarFormat:(NSString *)formatString
    timeZone:(NSTimeZone *)timeZone
```

Returns an `NSCalendarDate` object bound to the format string `formatString` and the time zone `timeZone`. If you specify `nil` after either or both of these arguments, the default format string (`@"%Y-%m-%d %H:-%M:-%S %z`) and time zone are used. See also `description`.

description

```
- (NSString *)description
```

Returns a string representation of the receiver. The representation conforms to the international format YYYY-MM-DD HH:MM:SS ±HHMM, where ±HHMM represents the time-zone offset in hours and minutes from Greenwich Mean Time (GMT). See also `descriptionWithCalendarFormat:timeZone:locale:`, `descriptionWithLocale:`.

descriptionWithCalendarFormat:timeZone:locale:

```
- (NSString *)descriptionWithCalendarFormat:
    (NSString *)formatString
    timeZone:(NSTimeZone *)aTimeZone
    locale:(NSDictionary *)localeDictionary
```

Returns a string representation of the receiver. The representation conforms to `formatString` (a `strftime`-style date-conversion string) and is adjusted to `aTimeZone`. Included are the keys and values that represent the locale data from `localeDictionary`. See also `description`.

descriptionWithLocale:

```
- (NSString *)descriptionWithLocale:
    (NSDictionary *)localeDictionary
```

Returns a string representation of receiver (see `description`). Included are the key and values that represent the locale data from `localeDictionary`. See also `descriptionWithCalendarFormat:timeZone:locale:`.

earlierDate:

```
- (NSDate *)earlierDate:(NSDate *)anotherDate
```

Compares the receiver's date to `anotherDate` and returns the one that's temporally earlier. See also `laterDate:`, `compare:`.

init

```
- (id)init
```

Initializes a newly allocated `NSDate` to the current date and time. See also initWithString:, `initWithTimeInterval:sinceDate:`, `initWithTimeIntervalSinceNow:`, `initWithTimeIntervalSinceReferenceDate:`.

initWithString:

```
- (id)initWithString:(NSString *)description
```

Returns an `NSDate` with a date and time value specified by the international string-representation format: YYYY-MM-DD HH:MM:SS ±HHMM, where ±HHMM is a time zone offset in hours and minutes from Greenwich Mean Time. See also `init`.

initWithTimeInterval:sinceDate:

```
- (NSDate *)initWithTimeInterval:(NSTimeInterval)seconds
    sinceDate:(NSDate *)anotherDate
```

Returns an `NSDate` initialized relative to another date object by `seconds` (plus or minus). See also `init`.

initWithTimeIntervalSinceNow:

```
- (NSDate *)initWithTimeIntervalSinceNow:(NSTimeInterval)seconds
```

Returns an NSDate initialized relative to the current date and time by seconds (plus or minus). See also init.

initWithTimeIntervalSinceReferenceDate:

```
- (id)initWithTimeIntervalSinceReferenceDate:
    (NSTimeInterval)seconds
```

Returns an NSDate initialized relative to the reference date and time by seconds (plus or minus). See also init.

isEqualToDate:

```
- (BOOL)isEqualToDate:(NSDate *)otherDate
```

Compares the receiver with otherDate. Returns YES if the dates are equal, and returns NO otherwise.

laterDate:

```
- (NSDate *)laterDate:(NSDate *)anotherDate
```

Compares the receiver's date to anotherDate and returns the one that's temporally later. See also earlierDate:, compare:.

timeIntervalSince1970

```
- (NSTimeInterval)timeIntervalSince1970
```

Returns the time interval between the receiver and the reference date used by UNIX systems. See also timeIntervalSinceReferenceDate (class method).

timeIntervalSinceDate:

```
- (NSTimeInterval)timeIntervalSinceDate:(NSDate *)anotherDate
```

Returns the interval between the receiver and anotherDate. See also timeIntervalSinceReferenceDate (class method).

timeIntervalSinceNow

```
- (NSTimeInterval)timeIntervalSinceNow
```

Returns the interval between the receiver and the current date and time. See also `timeIntervalSinceReferenceDate` (class method).

`timeIntervalSinceReferenceDate`

```
- (NSTimeInterval)timeIntervalSinceReferenceDate
```

Returns the interval between the receiver and the system's absolute reference date. This value is less than zero until the first instant of 1 January 2001. See also `timeIntervalSinceReferenceDate` (class method).

NSDateFormatter

Inherits From:	NSFormatter
Conforms To:	NSCoding, NSCopying
Declared In:	Foundation/NSDateFormatter.h

Class Description

Instances of `NSDateFormatter` format the textual representation of cells that contain `NSDate`s (including `NSCalendarDate`s), and convert textual representations of dates and times into `NSDate`s. You can express the representation of dates and times very flexibly: "Thu 22 Dec 1994" is just as acceptable as "12/22/94". With natural-language processing for dates enabled, users can also express dates colloquially, such as "today", "the day after tomorrow", and "a month from today".

To use an `NSDateFormatter`, allocate an instance of it and initialize it with `initWithDateFormat:allowNaturalLanguage:`. In the first argument use strftime-style conversion specifiers to compose the format string for textual representation. (For more information on these specifiers, see `NSCalendarDate`) Then use `NSCell`'s `setFormatter:` method to associate the `NSDateFormatter` object with a cell. The value of a cell (`NSCell`) is represented by an object, typically an `NSDate` object in this case. When this value needs to be displayed or edited, the cell passes its object to the `NSDateFormatter` instance, which returns the formatted string. When the user enters a string, or when one is programmatically written in a cell (using `setStringValue:`), the cell obtains the equivalent `NSDate` object from the `NSDateFormatter`.

NSControl provides delegation methods that permit you to validate cell contents and to handle errors in formatting. See the specification of the NSFormatter class for details.

When a cell with a NSDateFormatter is copied, the new cell retains the NSDateFormatter object instead of copying it. You remove an NSDateFormatter from a cell by specifying nil as the argument of setFormatter:.

NSCell Methods for Date Formatting

Alternatively, you can associate an NSDateFormatter object with a cell using an NSCell method. Send setEntryType: with an argument of NSDateType to a cell to associate that cell with a NSDateFormatter. The date format string is taken from the user default NSDateFormatString and natural-language processing of dates is enabled. To determine if a cell can accept a date conforming to the NSDateFormatString, send isEntryAcceptable: to the cell.

The NSDateFormatter approach is recommended over NSCell's setEntryType: because it allows you greater freedom in specifying the representation of dates. However, NSCell's setEntryType: and isEntryAcceptable: are OpenStep-compliant whereas the NSDateFormatter API is an extension to OpenStep.

Instances of NSDateFormatter are immutable.

Method Types

Activity	Class Method
Initializing an NSDateFormatter	– initWithDateFormat:allowNaturalLanguage:
Determining attributes	– allowsNaturalLanguage – dateFormat

Instance Methods

allowsNaturalLanguage

```
- (BOOL)allowsNaturalLanguage;
```

Returns YES if the NSDateFormatter attempts to process dates entered as a vernacular string (for example “today”, and “day before yesterday”). Returns NO if the NSDateFormatter does not do any natural-language processing of these date expressions.

dateFormat

```
- (NSString *)dateFormat
```

Returns the date format string used by an NSDateFormatter object. See NSCalendarDate for a list of the conversion specifiers permitted in date format strings.

initWithDateFormat:allowNaturalLanguage:

```
- (id)initWithDateFormat:(NSString *)format
    allowNaturalLanguage:(BOOL)flag;
```

Initializes and returns an NSDateFormatter instance that uses the date format in its conversions. See NSCalendarDate for a list of the conversion specifiers permitted in date format strings. Set flag to YES if you want the NSDateFormatter to process dates entered as expressions in the vernacular (for example, “tomorrow”); NSDateFormatter attempts natural-language processing only after it fails to interpret an entered string according to format. The following example creates a date formatter with the format string (as example) “Mar 15 1994” and then associates the formatter with the cells of a form (contactsForm).

```
NSDateFormatter *dateFormat = [[NSDateFormatter alloc]
    initWithDateFormat:@"%b %d %Y" allowNaturalLanguage:NO];
[[contactsForm cells] makeObjectsPerform:@selector(setFormatter:)
    withObject:dateFormat];
```

NSDeserializer

Characteristic	Description
Inherits From:	NSObject
Conforms To:	NSObject (NSObject)
Declared In:	Foundation / NSSerialization.h

Class Description

The `NSDeserializer` class declares methods that convert an abstract representation of a property list (as contained in an `NSData` object) into a graph of property list objects in memory. The `NSDeserializer` class object itself provides these methods; you don't create instances of `NSDeserializer`.

Options to these methods specify that container objects (arrays or dictionaries) in the resulting graph be mutable or immutable; that deserialization begin at the start of the data or from some position within it; or that deserialization occur lazily, so that a property list is deserialized only if it is actually going to be accessed. See the `NSSerializer` specification for more information on serialization.

Method Types

Activity	Class Method
Deserialization into property lists	+ deserializePropertyListFromData:atCursor: mutableContainers: + deserializePropertyListFromData: mutableContainers: + deserializePropertyListLazilyFromData: atCursor:length:mutableContainers:

Class Methods

deserializePropertyListFromData:atCursor: mutableContainers:

```
+ (id)deserializePropertyListFromData:(NSData *)data
    atCursor:(unsigned int*)cursor mutableContainers:(BOOL)mutable
```

Returns a property list object corresponding to the abstract representation in `data` at the location `cursor`. If `mutable` is `YES` and the object is a dictionary or an array, the re-composed object is made mutable. Returns `nil` if the object is not a valid one for property lists.

deserializePropertyListFromData: mutableContainers:

```
+ (id)deserializePropertyListFromData:(NSData *)data
    mutableContainers:(BOOL)mutable
```

Returns a property list object corresponding to the abstract representation in `data`. If `mutable` is `YES` and the object is a dictionary or an array, the re-composed object is made mutable. Returns `nil` if the data doesn't represent a property list.

deserializePropertyListLazilyFromData: atCursor:length:mutableContainers:

```
+ (id)deserializePropertyListLazilyFromData:(NSData *)data
    atCursor:(unsigned int*)cursor length:(unsigned int)length
    mutableContainers:(BOOL)mutable
```

Returns a property list from `data` at location `cursor`. The deserialization proceeds lazily. That is, if the data at the specified location has a length greater than `length`, a proxy is substituted for the actual property list as long as the constituent objects of that property list are not being accessed. If `mutable` is `YES` and the object is a dictionary or an array, the recomposed object is made mutable. Returns `nil` if the data doesn't represent a property list.

NSDictionary

Characteristic	Description
Inherits From:	NSObject
Conforms To:	NSCoding, NSCopying, NSMutableCopying NSObject (NSObject)
Declared In:	Foundation / NSDictionary.h

Class Description

The `NSDictionary` class declares the programmatic interface to objects that manage immutable associations of keys and values. Use this class when you need a convenient and efficient way to retrieve data associated with an arbitrary key.

A key-value pair within an `NSDictionary` is called an *entry*. Each entry consists of a string object that represents the key and another object (of any class) that is that key's value. You establish the entries when the `NSDictionary` is created, and thereafter the entries can't be modified. The complementary class `NSMutableDictionary` defines objects that manage modifiable collections of entries. See the `NSMutableDictionary` class specification for more information.

Internally, an NSDictionary uses a hash table to organize its storage and to provide rapid access to a value given the corresponding key. However, the methods defined for this class insulate you from the complexities of working with hash tables, hashing functions, or the hashed value of keys. These methods take key values directly, not their hashed form.

Generally, you instantiate an NSDictionary by sending one of the dictionary... messages to the class object. These methods return an NSDictionary containing the associations specified as arguments to the method. Each key argument is copied and the copy is added to the NSDictionary. Each corresponding value object receives a retain message to ensure that it won't be deallocated prematurely.

NSDictionary's three primitive methods—count and objectForKey: and keyEnumerator—provide the basis for all the other methods in its interface. The count method returns the number of entries in the object; objectForKey: returns the value associated with the given key; and keyEnumerator returns an object that lets you step through entries in the dictionary.

The other methods declared here operate by invoking one or more of these primitives. The nonprimitive methods provide convenient ways of accessing multiple entries at once. The description... methods and the writeToFile:atomically: method cause an NSDictionary to generate a description of itself and store it in a string object or a file.

Method Types

Activity	Class Method
Creating and initializing an NSDictionary	+ allocWithZone: + dictionary + dictionaryWithContentsOfFile: + dictionaryWithObjects:forKeys: + dictionaryWithObjects:forKeys:count: + dictionaryWithObjectsAndKeys: – initWithContentsOfFile: – initWithDictionary: – initWithObjectsAndKeys: – initWithObjects:forKeys: – initWithObjects:forKeys:count:
Accessing keys and values	– allKeys – allKeysForObject: – allValues – keyEnumerator – objectEnumerator – objectForKey: – objectsForKeys:notFoundMarker:
Counting entries	– count
Comparing dictionaries	– isEqualToDictionary:
Storing dictionaries	– description – descriptionInStringsFileFormat – descriptionWithLocale: – descriptionWithLocale:indent: – writeToFile:atomically:

Class Methods

`allocWithZone:`

```
+ (id)allocWithZone:(NSZone *)zone
```

Creates and returns an uninitialized `NSDictionary` in `zone`.

dictionary

```
+ (id)dictionary
```

Creates and returns an empty NSDictionary.

dictionaryWithContentsOfFile:

```
+ (id)dictionaryWithContentsOfFile:(NSString *)path
```

Creates and returns an NSDictionary from the keys and values found in the file specified by path. Returns nil if path cannot be found, or if path's contents don't represent a dictionary property list.

dictionaryWithObjects:forKeys:

```
+ (id)dictionaryWithObjects:(NSArray *)objects
    forKeys:(NSArray *)keys
```

Creates and returns an NSDictionary that associates objects from the objects array with keys from the keys array. Keys must be strings. Raises NSInvalidArgumentException if the number of objects is not equal to the number of keys.

dictionaryWithObjects:forKeys:count:

```
+ (id)dictionaryWithObjects:(id *)objects forKeys:(id *)keys
    count:(unsigned int)count
```

Creates and returns an NSDictionary containing count objects from the objects array. The objects are associated with count keys taken from the keys array.

dictionaryWithObjectsAndKeys:

```
+ (id)dictionaryWithObjectsAndKeys:(id)firstObject, ...
```

Creates and returns an NSDictionary that associates objects and keys from the argument list. The list must be in form: object1, key1, object2, key2, ..., nil. Raises NSInvalidArgumentException if any of the keys are nil, or if any of the keys are not of the NSString class.

Instance Methods

allKeys

```
– (NSArray *)allKeys
```

Returns an `NSArray` containing the receiver's keys or an empty array if the receiver has no entries.

allKeysForObject:

```
– (NSArray *)allKeysForObject:(id)object
```

Finds all occurrences of the value `anObject` in the receiver and returns an array with the corresponding keys.

allValues

```
– (NSArray *)allValues
```

Returns an `NSArray` containing the dictionary's values, or an empty array if the dictionary has no entries.

count

```
– (unsigned)count
```

Returns the number of entries in the receiver.

description

```
– (NSString *)description
```

Returns a string that represents the contents of the receiver. The form of the returned string is

```
{
key1 = key1description;
key2 = key2description;
...
}
```

If the description is less than 80 characters long, the returned string is put on one line. See also `description` (`NSArray`).

descriptionInStringsFileFormat

```
- (NSString *)descriptionInStringsFileFormat
```

Returns a string that represents the contents of the receiver. Key-value pairs are appropriate for use in `.strings` files.

descriptionWithLocale:

```
- (NSString *)descriptionWithLocale:
    (NSDictionary *)localeDictionary
```

Returns a string representation of the `NSDictionary` object. Included are the key and values that represent the locale data from `localeDictionary`. See also `description`.

descriptionWithLocale:indent:

```
- (NSString *)descriptionWithLocale:
    (NSDictionary *)localeDictionary
    indent:(unsigned int)level
```

Returns a string representation of the `NSDictionary` object. Included are the key and values that represent the locale data from `localeDictionary`. Elements are indented from the left margin by `level` + 1 multiples of four spaces, to make the output more readable. See also `description`.

initWithContentsOfFile:

```
- (id)initWithContentsOfFile:(NSString *)path
```

Initializes a newly allocated `NSDictionary` using the keys and values found in the file `path`. If `path` cannot be found, or path does not represent a dictionary, this method returns `nil`. Returns `self`. See also `propertyList` (`NSString`).

initWithDictionary:

```
- (id)initWithDictionary:(NSDictionary *)dictionary
```

Initializes a newly allocated NSDictionary by placing in it the keys and values contained in otherDictionary. Returns self.

initWithObjects:forKeys:

```
- (id)initWithObjects:(NSArray *)objects forKeys:(NSArray *)keys
```

Initializes a newly allocated NSDictionary by associating objects from the objects array with keys from the keys array. Keys must be strings. This method raises NSInvalidArgumentException if the number of objects is not equal to the number of keys.

initWithObjects:forKeys:count:

```
- (id)initWithObjects:(id *)objects
    forKeys:(id *)keys count:(unsigned)count
```

Initializes a newly allocated NSDictionary by associating count objects from the objects array with an equal number of keys from the keys array. Raises NSInvalidArgumentException if any of the objects or keys are nil.

initWithObjectsAndKeys:

```
- (id)initWithObjectsAndKeys:(id)firstObject,...
```

Initializes a newly allocated NSDictionary by placing in it the objects and keys from the argument list. The list must be in form: object1, key1, object2, key2, ..., nil. Raises NSInvalidArgumentException if any of the keys are nil, or if any of the keys are not of the NSString class.

isEqualToDictionary:

```
- (BOOL)isEqualToDictionary:(NSDictionary *)other
```

Compares the receiver to otherDictionary. If the contents of otherDictionary are equal to the contents of the receiver, this method returns YES. If not, it returns NO.

keyEnumerator

```
- (NSEnumerator *)keyEnumerator
```

Returns an NSEnumerator for accessing each of the receiver's keys. The following code shows how this method is used

```
id keyEnumerator = [self keyEnumerator];
id key;
while ( key = [keyEnumerator nextObject] ){
...
};
```

See also objectEnumerator, NSEnumerator.

objectEnumerator

```
- (NSEnumerator *)objectEnumerator
```

Returns an NSEnumerator that lets you access each the receiver's values. See also keyEnumerator, NSEnumerator.

objectForKey:

```
- (id)objectForKey:(id)aKey
```

Returns an entry's value given its key, or nil if no value is associated with aKey.

objectsForKeys:notFoundMarker:

```
- (NSArray *)objectsForKeys:(NSArray *)keys
    notFoundMarker:(id) marker
```

Given an array of keys, this method fills the corresponding array of corresponding objects. If the object is not present, marker is used instead. marker cannot be set to nil, because arrays can't contain nil. A common marker is @"".

writeToFile:atomically:

```
- (BOOL)writeToFile:(NSString *)path
    atomically:(BOOL)useAuxiliaryFile
```

Writes a textual description of the contents of the receiver to `filename`. If `useAuxiliaryFile` is `YES`, the data is written to a backup file and then, assuming no errors occur, the backup file is renamed to the intended file name. See also `description`.

NSDistantObject

Characteristic	Description
Inherits From:	NSProxy
Conforms To:	NSCoding NSObject (NSProxy)
Declared In:	Foundation / NSDistantObject.h

Class Description

The `NSDistantObject` class declares the programmatic interface to objects that serve as proxies to remote real objects.

Your application does not in general need to explicitly create `NSDistantObject` objects—they are created automatically when you create `NSConnection` objects for a remote object.

Exceptions

`NSDistantObject` raises an `NSInternalInconsistencyException` for a variety of exceptions resulting from internal consistency failures.

Method Types

Activity	Class Method
Building a proxy	+ proxyWithLocal:connection: + proxyWithTarget:connection:
Initializing a proxy	– initWithLocal:connection: – initWithTarget:connection:
Specifying a protocol	– setProtocolForProxy:
Returning the proxy's connection	– connectionForProxy

Class Methods

proxyWithLocal:connection:

```
+ (NSDistantObject *)proxyWithLocal:(id)target
    connection:(NSConnection *)connection
```

Builds and returns a local proxy for a local object `target`, forming a remote proxy on the other side of `connection`.

proxyWithTarget:connection:

```
+ (NSDistantObject *)proxyWithTarget:(id)target
    connection:(NSConnection *)connection
```

Builds and returns a remote proxy where `target` is an object on the other side of `connection`.

Instance Methods

connectionForProxy

```
- (NSConnection *)connectionForProxy
```

Returns the `NSConnection` instance used by the proxy.

initWithLocal:connection:

```
- (id)initWithLocal:(id)target
    connection:(NSConnection *)connection
```

Builds a local proxy for a local object `target`, forming a remote proxy on the other side of `connection`. You may not retain or otherwise use this proxy.

initWithTarget:connection:

```
- (id)initWithTarget:(id)target
    connection:(NSConnection *)connection
```

Builds a remote proxy where `target` is an object on the other side of `connection`. It may deallocate and return `nil` if this target is already known on the connection. This is the designated initializer for subclasses.

setProtocolForProxy:

```
- (void)setProtocolForProxy:(Protocol *)proto
```

Sets the proxy's protocol to `proto` for efficiency.

NSEnumerator

Characteristic	Description
Inherits From:	NSObject
Conforms To:	NSObject (NSObject)
Declared In:	Foundation / NSUtilities.h

Class Description

`NSEnumerator` is a simple abstract class whose instances enumerate collections of other objects. Collection objects—such as `NSSet`s, `NSArray`s, and `NSDictionary`s—provide `NSEnumerator` objects that can traverse their contents. You send `nextObject` repeatedly to an `NSEnumerator` to have it return the next object in the collection. When there are no more objects to return, `nextObject` returns `nil`.

Collection classes include methods that return an enumerator appropriate to the type of collection. `NSArray` has two methods that return an `NSEnumerator` object, `objectEnumerator`, and `reverseObjectEnumerator` (the former traverses the array starting at its first object, while the latter starts with the last object and continues backward through the array to the first object). `NSSet`'s `objectEnumerator` provides an enumerator for sets. `NSDictionary` has two enumerator-providing methods: `keyEnumerator` and `objectEnumerator`.

Note – Collections shouldn't be modified during enumeration. `NSEnumerator` imposes this restriction to improve enumeration speed.

Method Types

Activity	Class Method
Traversing a collection	– allObjects – nextObject

Instance Methods

`allObjects`

```
- (NSArray *)allObjects
```

Calls `nextObject` on the collection until the end of the collection is reached. Returns the enumerated objects in an array.

`nextObject`

```
– (id)nextObject
```

Returns the next object in the collection being enumerated, for example, an `NSArray` or `NSDictionary`). Returns `nil` when the collection has been traversed.

NSException

Characteristic	Description
Inherits From:	NSObject
Conforms To:	NSCoding, NSCopying NSObject (NSObject)
Declared In:	Foundation / NSException.h

Class Description

The `NSException` class provides an object-oriented way for applications to announce and react to exceptional conditions.

An exceptional condition is one that interrupts the normal flow of program execution. Each application can interpret different types of conditions as exceptional. For example, one application might view as exceptional the attempt to save a file in a directory that's write-protected. In this sense, an exceptional condition can be equivalent to an error. Another application might interpret the user's keypress as an exceptional condition—an indication that a long-running process should be aborted.

Raising an Exception

Once an exceptional condition is detected, it must be propagated to the routine or routines that will handle it, a process referred to as "raising an exception." In the OpenStep exception handling system, exceptions are raised by instantiating an exception object and sending it a `raise` message.

Exception objects encapsulate:

- A name—A short `NSString` that is used to uniquely identify the exception
- A reason—A longer `NSString` that contains a "human-readable" reason for the exception—This reason object is printed when the exception object is printed using the "%@" format.
- `userInfo`—An `NSDictionary` object that you can use to supply application-specific data to the exception handler. For example, if a function's return value caused the exception to be raised, you could pass the

return value to the exception handler through the `userInfo` dictionary. Or, if the exception handler displays a panel in response to the exception, `userInfo` could contain the text string to be displayed in the panel.

Handling an Exception

Sending a raise message to an exception object initiates the propagation of the exception and passes data about it. Where and how the exception is handled depends on where you send the message from. First, look at a simple case.

In general, a `raise` message is sent to an exception object within the domain of an exception handler. An exception handler is a control structure created by the macros `NS_DURING`, `NS_HANDLER`, and `NS_ENDHANDLER`.

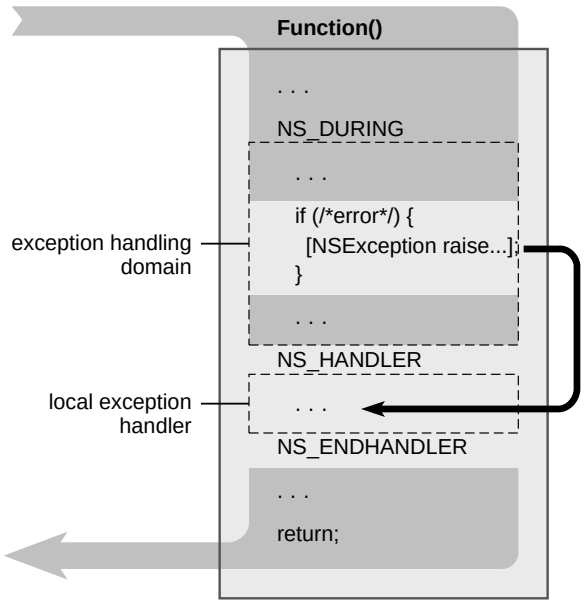


Figure 5-2 Exception Handling Domain and Handler

The section of code between `NS_DURING` and `NS_HANDLER` is the exception handling domain; the section between `NS_HANDLER` and `NS_ENDHANDLER` is the local exception handler. The normal flow of program execution is marked by the gray arrow; the code within the local exception handler is executed only

if an exception is raised. Sending a `raise` message to an exception object causes program control to jump to the first executable line following `NS_HANDLER`, as indicated by the black arrow.

An exception can be raised directly within the exception handling domain, or indirectly from one of the methods or functions invoked from the domain. No matter how deeply in a call sequence an exception is raised, execution jumps to the local exception handler (assuming there are no intervening exception handlers, as discussed in the next section). In this way, exceptions raised at a low level can be caught at a high level.

If an exception is raised and execution begins within the local exception handler, it either continues until all appropriate statements are executed or the exception is raised again to invoke the services of an encompassing exception handler, as described in the next section.

If the exception isn't raised again, execution within the local exception handler continues until it leaves the local handler by:

- "Falling off the end"
- Calling `NS_VALRETURN()`
- Calling `NS_VOIDRETURN`

Note – A simple return from the exception-handling domain is not permitted.

"Falling off the end" is simply the normal execution pathway introduced above. After all appropriate statements within the domain are executed (and no exception is raised), execution continues on the line following `NS_ENDHANDLER`. Alternatively, you can return control to the caller from within the domain by calling `NS_VALRETURN()` or `NS_VOIDRETURN`, depending on whether you need to return a value.

You can't use `goto` or `return()` to exit an exception-handling domain—errors will result. Nor can you use `setjmp()` and `longjmp()` if the jump entails crossing an `NS_DURING` statement. Since in many cases you won't know if the code that your program calls has exception-handling domains within, it is generally not recommended that you use `setjmp()` and `longjmp()` in your application.

Nested Exception Handlers

Exception handlers can be nested so that an exception raised in an inner domain can be treated by the local exception handler and any number of encompassing exception handlers. The following diagram illustrates the use of nested exception handlers, and is discussed in the text that follows.

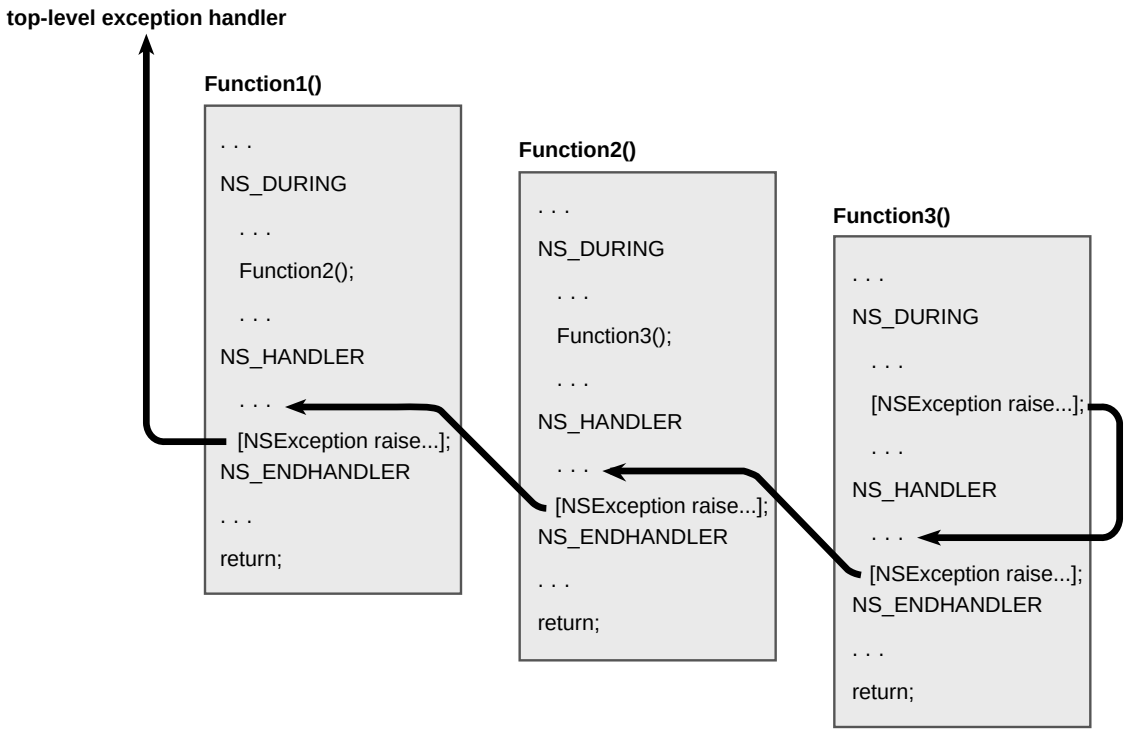


Figure 5-3 Nested Exception Handlers

An exception raised within `Function3`'s domain causes execution to jump to its local exception handler. In a typical application, this exception handler checks the values contained in the `NSException` object to determine the nature of the exception. For exception types that it recognizes, the local handler responds and then sends a `raise` message to the exception object to pass notification of the exception to the handler above it (in this case, the handler in `Function2`). `Function2`'s exception handler does the same and then raises the exception to Function1's handler. Finally, `Function1`'s handler re-raises

the exception. Since there's no exception-handling domain above `Function1`, the exception is transferred to a default top-level error handler. For applications based on the Application Kit, this top-level handler invokes `NSApplication`'s `reportException:` method, which writes an error message to the console.

An exception that's re-raised appears to the next higher handler just as if the initial exception had been raised within its own exception-handling domain.

Raising an Exception Outside of an Exception Handler

If an exception is raised outside of any exception handler, it's intercepted by the uncaught exception handler, a function set by `NSSetUncaughtExceptionHandler()` and returned by `NSUncaughtExceptionHandler()`(see the Foundation Kit's "Functions" chapter). You can change the way uncaught exceptions are handled by using `NSSetUncaughtExceptionHandler()` to establish a different procedure as the handler. However, because of the design of the Application Kit, it's rare for an exception to be raised outside of an exception handling domain. The `NSApplication` object's event loop itself is within an exception handling domain. On each cycle of the loop, the `NSApplication` object retrieves an event and sends an event message to the appropriate object in the application. The code you write for custom objects and Application Kit objects is executed within the context of the event loop's exception handler.

Predefined Exceptions

OpenStep predefines a number of exception names. These exceptions are listed in `NSException.h`; for example:

```
extern NSString *NSGenericException;
extern NSString *NSRangeException;
extern NSString *NSInvalidArgumentException;
```

For a complete list of global exception names, see the "Types and Constants" chapter. You can catch any of these exceptions from within your exception handler by comparing the exception's name with these predefined exception names.

Method Types

Activity	Class Method
Creating and raising exceptions	+ exceptionWithName:reason:userInfo: + raise:format: + raise:format:arguments: – initWithName:reason:userInfo: – raise
Querying exceptions	– name – reason – userInfo

Class Methods

exceptionWithName:reason:userInfo:

```
+ (NSException *)exceptionWithName:(NSString *)name
    reason:(NSString *)reason userInfo:(NSDictionary *)userInfo
```

Creates an exception object, assigning it `name` as its name, `reason` as its human-readable explanation, and `userInfo` as arbitrary data that will accompany the exception.

raise:format:

```
+ (volatile void)raise:(NSString *)name
    format:(NSString *)format,...
```

Creates and raises an exception with name `name` and a reason constructed from `format` and the following arguments in the manner of `printf()`. The user-defined information is `nil`. Invokes `raise` as part of its implementation.

raise:format:arguments:

```
+ (volatile void)raise:(NSString *)name format:(NSString *)format
    arguments:(va_list)argList
```

Creates and raises an exception with name `name` and a reason constructed from `format` and the arguments in `argList`, in the manner of `vprintf()`. The user-defined information is `nil`. Invokes `raise` as part of its implementation.

Instance Methods

`initWithName:reason:userInfo:`

```
- (id)initWithName:(NSString *)name reason:(NSString *)reason
    userInfo:(NSDictionary *)userInfo
```

Initializes a newly allocated exception object, assigning it `name` as its name, `reason` as its human-readable explanation, and `userInfo` as arbitrary data that will accompany the exception.

`name`

```
- (NSString *)name
```

Returns the exception's name. See also `exceptionWithName:reason:userInfo:`.

`raise`

```
- (volatile void)raise
```

Raises the exception, causing program flow to jump to the enclosing error handler.

`reason`

```
- (NSString *)reason
```

Returns the exception's reason. See also `exceptionWithName:reason:userInfo:`.

`userInfo`

```
- (NSDictionary *)userInfo
```

Returns the exception's user-defined data. See also `exceptionWithName:reason:userInfo:`.

NSFormatter

Inherits From:	NSObject
Conforms To:	NSCoding, NSCopying
Declared In:	Foundation / NSFormatter.h

Class Description

`NSFormatter` is an abstract class that declares an interface for objects that format the textual representation of cell contents. The Foundatition Kit provides two concrete subclasses of `NSFormatter`: `NSNumberFormatter` and `NSDateFormatter`.

Cells, which are instances of `NSCell` and its subclasses, can have any arbitrary object as their content. However, when cells are to be displayed or edited, they must convert this object to an `NSString`. If no formatting object is associated with a cell, the cell displays its content by invoking the localized description method of the object it contains. But if the cell has a formatting object, the cell invokes this object's `stringForObjectValue:` method to obtain the correctly formatted string. Conversely, when the user enters text into a cell, the cell needs to convert the text to the underlying object; formatting objects handle this conversion as well.

To use a formatting object, you must create an instance of `NSNumberFormatter`, `NSDateFormatter`, or a custom `NSFormatter` subclass and associate the object with a cell. The cell invokes the formatting behavior of this instance every time it needs to display its object or have it edited, and every time it needs to convert a textual representation to its object.

Instances of `NSFormatter` subclasses are immutable. In addition, when a cell with a formatter object is copied, the new cell retains the formatter object instead of copying it.

Note that `NSCell` provides two methods that operate almost the same as instances of `NSNumberFormatter` and `NSDateFormatter`. One method, `setEntryType:`, takes a constant that specifies date formatting (as specified in the user defaults) or a typical numeric format (integer, float, positive float,

double, and so on). With `isEntryAcceptable:`, you can ask a cell for the type of value it expects. Another method, `setFloatingPointFormat:left:right:`, allows you to specify the digits that appear to the left and right of the decimal point. See the `NSNumberFormatter` and `NSDateFormatter` for further details.

Delegation Methods for Validation and Error Handling

`NSControl` provides the delegation method `control:isValidObject:` for validating the contents of cells embedded in controls (instances of `NSTextField` and `NSMatrix` in particular). Validation checks for values that are permissible as objects but that are undesirable in a given context, such as a date field in which dates should never be in the future, or zip codes that are valid for a certain state.

The method `control:isValidObject:` is invoked when the cursor leaves a cell (that is, the associated control relinquishes first-responder status) but before the string value of the cell's object is displayed. Return `YES` to allow display of the string and `NO` to reject display and return the cursor to the cell. The following example evaluates an object (an `NSDate`) and rejects it if the date is in the future:

```
- (BOOL)control:(NSControl *)control isValidObject:(id)obj
{
    if (control == contactsForm) {
        if (![obj isKindOfClass:[NSDate class]]) return NO;
        if ([[obj laterDate:[NSDate date]] isEqual:obj]) {
            NSRunAlertPanel(@"Date not valid",
            @"Reason: date in future", NULL, NULL, NULL);
            return NO;
        }
    }
    return YES;
}
```

`NSControl` also has delegation methods for handling errors returned in implementations of `NSFormatter`'s `getObjectValue:forString:errorDescription:` and `isPartialStringValid:newEditingString:errorDescription:`. These delegation methods are, respectively, `control:didFailToFormatString:errorDescription:` and `control:didFailToValidatePartialString:errorDescription:`.

Making an NSFormatter Subclass

There are many possibilities for custom subclasses of `NSFormatter`. You might find use for a custom formatter of telephone numbers, or a custom formatter of part numbers. To subclass `NSFormatter`, you must, at the least, override the two primitive methods `stringForObjectValue:` and `getObjectValue:forString:errorDescription:`. In the first method you convert the cell's object to a string representation; in the second method you convert the string to the object associated with the cell.

If the string for editing is different than the string for display (for example, the display version of a currency field should show a dollar sign but the editing version shouldn't) implement `editingStringForObjectValue:` in addition to `stringForObjectValue:`.

The method `isPartialStringValid:newEditingString:errorDescription:` allows you to edit the textual contents of a cell at each key press or to prevent entry of invalid characters. You might apply this on-the-fly editing to things like telephone numbers or social security numbers; the person entering data only needs to enter the number since the formatter automatically inserts the separator characters.

Method Types

Activity	Class Method
Textual representation of cell content	– editingStringForObjectValue: – stringForObjectValue:
Object equivalent to textual representation	– getObjectValue:forString:errorDescription:
Dynamic cell editing	– isPartialStringValid:newEditingString: errorDescription:

Instance Methods

`editingStringForObjectValue:`

```
-(NSString *)editingStringForObjectValue:(id)anObject
```

The default implementation of this method invokes `stringForObjectValue:`. When implementing a subclass, override this method only when the string that users see and the string that they edit are different. In your implementation, return an `NSString` that is used for editing, following the logic recommended for implementing `stringForObjectValue:`. As an example, you would implement this method if you want the dollar signs in displayed strings removed for editing. See also `stringForObjectValue:`.

getObjectValue:forString:errorDescription:

```
-(BOOL)getObjectValue:(id *)anObject
    forString:(NSString *)string
    errorDescription:(NSString **)error
```

The default implementation of this method raises an exception. In your subclass implementation, return by reference the object `anObject` after creating it from the string passed in. Return `YES` if the conversion from string to cell-content object was successful and `NO` if any error prevented the conversion. If you return `NO`, also return by indirection an `NSString` (in `error`) that explains the reason why the conversion failed; the delegate (if any) of the `NSControl` managing the cell can then respond to the failure in `control:didFailToFormatString:errorDescription:`.

The following implementation example (which is paired with the `stringForObjectValue:` example below) converts an `NSString` representation of a dollar amount that includes the dollar sign; it uses an `NSScanner` to convert this amount to a float after stripping out the initial dollar sign.

```
- (BOOL)getObjectValue:(id *)obj forString:(NSString *)string
    errorDescription:(NSString **)error
{
    float floatResult;
    NSScanner *scanner;
    BOOL retval = NO;
    NSString *err = nil;

    scanner = [NSScanner scannerWithString:string];
    if ([string hasPrefix:@"$"]) [scanner setScanLocation:1];
    if ([scanner scanFloat:&floatResult]
                && ([scanner scanLocation] == [string length] )) {
        if (obj) {
```

```
            *obj = [NSNumber numberWithFloat:floatResult];
            retval = YES;
        }
        else {
                err = @"Couldn't convert to float";
        }
    }
    if (error) {
        *error = err;
    }
    return retval;
}
```

See also `stringForObjectValue:`.

`isPartialStringValid:newEditingString: errorDescription:`

```
-(BOOL)isPartialStringValid:(NSString *)partialString
    newEditingString:(NSString **)newString
    errorDescription:(NSString **)error
```

Since this method is invoked at each key press in the cell, it permits editing or evaluation of cell text as it is typed. The text as currently typed (`partialString`) is passed in. Evaluate this text according to the context, edit the text if necessary, and return by reference any edited `NSString` in `newString`. Return `YES` if `partialString` is acceptable and `NO` if `partialString` is unacceptable. If you return `NO` and `newString` is `nil`, `partialString` minus the last character typed is displayed. If you return `NO`, you can also return by indirection an `NSString` (in `error`) that explains the reason why the validation failed. The delegate (if any) of the `NSControl` managing the cell can then respond to the failure in `control:didFailToValidatePartialString:errorDescription:`.

`stringForObjectValue:`

```
-(NSString *)stringForObjectValue:(id)anObject
```

The default implementation of this method raises an exception. When subclassing, return the `NSString` that textually represents the cell's object for display and, if `editingStringForObjectValue:` is unimplemented, for

editing. First test the passed-in object to see if it is of the correct class. If it is not, return `nil`; if it is of the correct class, return a properly formatted and, if necessary, localized string.

The following implementation, which is paired with the `getObjectValue:forString:errorDescription:` example above, prefixes a two-digit float representation with a dollar sign:

```
- (NSString *)stringForObjectValue:(id)anObject
{
    if (![anObject isKindOfClass:[NSNumber class]]) {
        return nil;
    }
    return [NSString stringWithFormat:@"$%.2f", [anObject
           floatValue]];
}
```

NSInvocation

Characteristic	Description
Inherits From:	NSObject
Conforms To:	NSCoding NSObject (NSObject)
Declared In:	Foundation / NSInvocation.h

Class Description

Objects of the `NSInvocation` class provide a system-independent means to construct message calls to other objects. An `NSInvocation` object constructs a *target* object to which a message can be sent, a *selector* for that method, an *argument list* for the selector, and a return value. `NSInvocation` objects provide great flexibility in that the methods, method arguments, and targets of the methods may be constructed dynamically.

The final sending of the message to the target object can be performed at any time, independent of constructing the invocation. For example, methods could be dispatched based on timer events. In addition, return values from the methods are stored in the `NSInvocation` object and can be retrieved at any later stage in processing.

Note – See `NSMethodSignature` for a description of how to construct method signatures.

Method Types

Activity	Class Method
Creating invocations	+ invocationWithMethodSignature:
Managing invocation arguments	– argumentsRetained – getArgument:atIndex: – getReturnValue: – methodSignature – retainArguments – selector – setArgument:atIndex: – setReturnValue: – setSelector: – setTarget: – target
Dispatching an invocation	– invoke – invokeWithTarget:

Class Methods

`invocationWithMethodSignature:`

```
+ (NSInvocation *)invocationWithMethodSignature:
    (NSMethodSignature *)sig
```

Returns an invocation object able to construct calls to objects using method selectors with type signatures described by `sig`. Raises `NSInvalidArgumentException` if `sig` is `nil`.

Instance Methods

`argumentsRetained`

```
- (BOOL)argumentsRetained
```

Returns YES if arguments are retained, and returns NO otherwise. See also retainArguments.

getArgument:atIndex:

```
- (void)getArgument:(void *)argumentLocation atIndex:(int)index
```

Copies the argument stored at index into the storage pointed to by argumentLocation where 2 is the index of the first argument, 3 is the index of the second argument, and so on. Raises NSInvalidArgumentException if index is greater than the number of arguments (or less than -1), or if arguments aren't available. See also setArgument:atIndex:.

getReturnValue:

```
- (void)getReturnValue:(void *)retLoc
```

Copies the invocation's return value into the storage pointed to by retLoc. Raises NSInvalidArgumentException if index is greater than the number of arguments, or if arguments aren't available.

invoke

```
- (void)invoke
```

Causes the message encoded in the invocation to be dispatched to its target.

invokeWithTarget:

```
- (void)invokeWithTarget:(id)target
```

Causes the message encoded in the invocation to be dispatched to target.

methodSignature

```
- (NSMethodSignature *)methodSignature
```

Returns the invocation's method signature object. See also NSMethodSignature.

retainArguments

```
- (void)retainArguments
```

By default, target and arguments are not retained, and C strings are not copied. This method instructs the invocation to retain its arguments and target, and to make copies of C strings. This method is invoked automatically by timers. This method should be invoked whenever the dynamic scope of the invocation can exceed its arguments. See also `argumentsRetained`.

selector

```
- (SEL)selector
```

Returns the invocation's selector.

setArgument:atIndex:

```
- (void)setArgument:(void *)argumentLocation atIndex:(int)index
```

Sets the argument stored at `index` to the storage pointed to by `argumentLocation` where 2 is the index of the first argument, 3 is the index of the second, and so on. See also `getArgument:atIndex:`.

setReturnValue:

```
- (void)setReturnValue:(void *)retLoc
```

Sets the invocation's return value to that indicated by `retLoc`.

setSelector:

```
- (void)setSelector:(SEL)selector
```

Sets the invocation's selector to `selector`.

setTarget:

```
- (void)setTarget:(id)target
```

Sets the invocation's target to `target`.

target

```
- (id)target
```

Returns the invocation's target; returns `nil` if there is no target.

NSLock

Characteristic	Description
Inherits From:	NSObject
Conforms To:	NSLocking NSObject (NSObject)
Declared In:	Foundation / NSLock.h

Class Description

An `NSLock` is used to protect critical regions of code. A lock is created once and is subsequently used to protect one or more regions of code. If a region of code is in use, an `NSLock` waits using the `condition_wait()` function, so the thread doesn't busy-wait. The following example shows the use of an `NSLock` with the methods `lock` and `unlock` defined in the `NSLocking` protocol:

```
NSLock  *theLock = [NSLock new];       // done once!
/* ... other code */
[theLock lock];
/* ... possibly a long time of fussing with global data... */
[theLock unlock];
```

The `NSConditionLock`, `NSLock`, and `NSRecursiveLock` classes all implement the `NSLocking` protocol with various features and performance characteristics; see the other class descriptions for more information.

Note – See also the `NSLocking` protocol.

Method Types

Activity	Class Method
Acquiring a lock	– tryLock

Instance Methods

tryLock

```
- (BOOL)tryLock
```

Attempts to acquire a lock. Returns `YES` if successful and `NO` otherwise. Returns immediately.

NSMethodSignature

Characteristic	Description
Inherits From:	NSObject
Conforms To:	NSObject (NSObject)
Declared In:	Foundation / NSMethodSignature.h

Class Description

`NSMethodSignature` provides the programmatic interface to objects that provide access to the "type signatures" of an object's methods—that is, the types of the arguments and return value. A *method signature* is used by the distributed objects machinery to determine how to correctly encode method names and arguments for the underlying interprocess communications. The typical use of method signatures is when a message is sent to a remote object via a proxy. If the proxy doesn't know the types of arguments a remote object will use, the proxy first has to query the remote object for its method signature object, which specifies the types the method requires as arguments. The proxy then knows how to encode the data it has been passed, and forward it correctly to the real object.

Given a method signature, all other available instance methods query the object for information about the signature, such as its return type, number of arguments, stack frame size (obviously architecture-dependent), and so on.

See the `NSInvocation` for the class which can use method signature objects to send messages to other objects.

Method Types

Activity	Class Method
Querying a method signature	– argumentInfoAtIndex: – frameLength – getArgumentTypeAtIndex: – isOneway – methodReturnLength – methodReturnType – numberOfArguments

Instance Methods

`argumentInfoAtIndex:`

```
– (NSArgumentInfo)argumentInfoAtIndex:(unsigned)index
```

Returns information about the argument at `index`. Indices begin with 0. The "hidden" arguments `self` and `_cmd` are indexed at 0 and 1; method-specific arguments begin at index 2. If `index` is too large for the actual number of arguments, `NSInvalidArgumentException` is raised. See also `NSArgumentInfo`.

`frameLength`

```
– (unsigned)frameLength
```

Returns the number of bytes that the arguments, taken together, would occupy on the stack.

`getArgumentTypeAtIndex:`

```
- (const char *)getArgumentTypeAtIndex:(unsigned)index
```

Returns the type of method argument `index`. Raises `NSInvalidArgumentException` if index is out of bounds.

isOneway

```
- (BOOL)isOneway
```

Returns `YES` if the method is asynchronous (that is, it returns without waiting for the receiver to finish processing it), and returns `NO` otherwise.

methodReturnLength

```
- (unsigned)methodReturnLength
```

Returns the number of bytes required by the return value.

methodReturnType

```
- (const char *)methodReturnType
```

Returns a string encoding the return type of the method. What the characters in the string represent is usually defined by some implementation-dependent runtime types.

numberOfArguments

```
- (unsigned)numberOfArguments
```

Returns the number of arguments recorded in the receiver. This will be at least two, since it includes the "hidden" arguments `self` and `_cmd`, which are the first two arguments passed to every method implementation.

NSMutableArray

Characteristic	Description
Inherits From:	NSArray : NSObject
Conforms To:	NSCoding, NSCopying, NSMutableCopying (NSArray) NSObject (NSObject)
Declared In:	Foundation / NSArray.h

Class Description

The `NSMutableArray` class declares the programmatic interface to objects that manage a modifiable array of objects. This class adds insertion and deletion operations to the basic array-handling behavior it inherits from `NSArray`.

The array operations that `NSMutableArray` declares are conceptually based on these three methods:

```
addObject:
replaceObjectAtIndex:withObject:
removeLastObject
```

The other methods in its interface provide convenient ways of inserting an object into a specific slot in the array and of removing an object based on its identity or position in the array.

When an object is removed from a mutable array it receives a `release` message, which can cause it to be deallocated. Note that if your program keeps a reference to such an object, the reference may become invalid unless you remember to send the object a `retain` message before it is removed from the array. For example, the third statement in the following example could result in a run-time error, except for the `retain` message in the first statement:

```
id anObject = [[anArray objectAtIndex:0] retain];
[anArray removeObjectAtIndex:0];
[anObject someMessage];
```

Implementing Subclasses of NSMutableArray

Although conceptually the interface to the `NSMutableArray` class is based on the three methods listed previously, for performance reasons two others—`insertObject:atIndex:` and `removeObjectAtIndex:`—also

directly access the object's data. These two methods could be implemented using the methods listed above but in doing so would incur unnecessary overhead from the `retain` and `release` messages that objects would receive as they are shifted to accommodate the insertion or deletion of an element. Thus, if you create a subclass of `NSMutableArray`, you should override all five primitive methods so that the other methods in `NSMutableArray`'s interface work properly.

Method Types

Activity	Class Method
Creating and initializing an NSMutableArray	+ allocWithZone: + arrayWithCapacity: – initWithCapacity:
Adding objects	– addObject: – addObjectsFromArray: – insertObject:atIndex:
Removing objects	– removeAllObjects – removeLastObject – removeObject: – removeObjectAtIndex: – removeObjectIdenticalTo: – removeObjectIdenticalTo:inRange: – removeObject:inRange: – removeObjectsFromIndices:numIndices: – removeObjectsInArray: – removeObjectsInRange:
Replacing objects	– replaceObjectAtIndex:withObject: – replaceObjectsInRange:withObjectsFromArray: – replaceObjectsInRange:withObjectsFromArray:range: – setArray:
Sorting elements	– sortUsingFunction:context: – sortUsingSelector:

Class Methods

`allocWithZone:`

```
+ (id)allocWithZone:(NSZone *)zone
```

Creates and returns an uninitialized `NSMutableArray` in `zone`. See also `arrayWithCapacity:`, `initWithCapacity:`.

arrayWithCapacity:

```
+ (id)arrayWithCapacity:(unsigned int)aNumItems
```

Creates and returns an NSMutableArray, giving it enough allocated memory to hold numItems objects. See also initWithCapacity:, allocWithZone:.

Instance Methods

addObject:

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

Inserts anObject at the end of the array. Raises NSInvalidArgumentException if anObject is nil. See also addObjectsFromArray:.

addObjectsFromArray:

```
- (void)addObjectsFromArray:(NSArray *)anotherArray
```

Adds the objects contained in anotherArray to the end of the receiver's array. Raises NSInvalidArgumentException if any member of anotherArray is nil. See also addObject:.

initWithCapacity:

```
- (id)initWithCapacity:(unsigned int)aNumItems
```

Initializes a newly allocated NSMutableArray, giving it enough memory to hold numItems objects. See also arrayWithCapacity:.

insertObject:atIndex:

```
- (void)insertObject:(id)anObject atIndex:(unsigned int)index
```

Inserts anObject into the array at index. Raises NSInvalidArgumentException if anObject is nil. Raises NSRangeException if index is outside of the bounds of the array. See also addObject:.

removeAllObjects

```
- (void)removeAllObjects
```

Sends the `removeLastObject` message to empty the array of all its elements, from last to first. See also `removeObject:`, `removeLastObject`.

removeLastObject

```
- (void)removeLastObject
```

Removes the last object in the array and sends it a `release` message. Raises `NSRangeException` if there are no objects in the array. See also `removeAllObjects`, `removeObject:`.

removeObject:

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

Removes all occurrences of `anObject`. `isEqual:` is used to test for `anObject`. See also `removeAllObjects`, `removeLastObject`, `removeObjectAtIndex:`, `removeObjectIdenticalTo:`, `removeObjectsFromIndices:numIndices:`, `removeObjectsInArray:`.

removeObjectAtIndex:

```
- (void)removeObjectAtIndex:(unsigned int)index
```

Removes the object at `index` and moves all elements beyond `index` up one slot to fill the gap. Raises `NSRangeException` if `index` is outside of the bounds of the array. See also `removeObjectsFromIndices:numIndices:`, `removeObject:`.

removeObjectIdenticalTo:

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

Removes all elements having the same `id` as `anObject`. See also `removeObject:`.

removeObjectIdenticalTo:inRange:

```
- (void)removeObjectIdenticalTo:(id)anObject
    inRange:(NSRange)range
```

Searches the specified range removing all occurrences of `anObject`. See also `removeObject:inRange:`, `removeObjectIdenticalTo:`.

removeObject:inRange:

```
- (void)removeObject:(id)anObject inRange:(NSRange)range
```

Searches the given array `range` for `anObject`, removing it if found. Raises an exception if `range` yields an out of bounds index.See also `removeObjectIdenticalTo:inRange:`, `removeObjectsInRange:`, `replaceObjectsInRange:withObjectsFromArray:`.

removeObjectsFromIndices:numIndices:

```
– (void)removeObjectsFromIndices:(unsigned int*)indices
    numIndices:(unsigned int)count
```

Removes objects at the positions specified in the `indices` array, which has `count` elements. Raises `NSRangeException` if any of the `indices` is outside of the bounds of the array. This method is provided for efficiency reasons; it will not work if the receiver is a proxy to an array in another process. See also `removeObjectAtIndex:`, `removeObject:`.

removeObjectsInArray:

```
– (void)removeObjectsInArray:(NSArray *)otherArray
```

Removes from the receiver the objects found in `otherArray`. See also `removeObjectsFromIndices:numIndices:`, `removeObject:`.

removeObjectsInRange:

```
- (void)removeObjectsInRange:(NSRange)range
```

Removes the given range of objects from the array. Raises an exception if `range` yields an out of bounds index.See also `removeObject:inRange:`, `replaceObjectsInRange:withObjectsFromArray:`.

replaceObjectAtIndex:withObject:

```
- (void)replaceObjectAtIndex:(unsigned int)index
    withObject:(id)anObject
```

Replaces the object at `index` with `anObject`. Raises `NSInvalidArgumentException` if `anObject` is `nil`. Raises `NSRangeException` if `index` is not within the bounds of the array. See also `setArray:`.

replaceObjectsInRange:withObjectsFromArray:

```
- (void)replaceObjectsInRange:(NSRange)range
    withObjectsFromArray:(NSArray *)otherArray
```

Replaces the given `range` of the receiving array's objects with the contents of `otherArray`. If `range.length` is 0, then this method inserts the contents of `otherArray` at `range.location`. For example, if `range` is {5,0} and `otherArray` contains 3 objects, then the current object at index 5 of the receiver will be at index 8 after the message is sent. Raises an exception if `range` yields an out of bounds index. See also `replaceObjectsInRange:withObjectsFromArray: range:`, `removeObjectsInRange:`.

replaceObjectsInRange:withObjectsFromArray: range:

```
- (void)replaceObjectsInRange:(NSRange)range
    withObjectsFromArray:(NSArray *)otherArray
    range:(NSRange)otherRange
```

Replaces the given `range` of the receiving array's objects with the given `otherRange` of `otherArray`. If `range.length` is 0, then this method inserts the given `otherRange` of `otherArray` at `range.location`. For example, if `range` is {5,0} and `otherRange` specifies 3 objects, then the current object at index 5 of the receiver will be at index 8 after the message is sent.If `otherRange.length` is 0, then this method behaves like `removeObjectsInRange:`. Raises an exception if `range` yields an out of bounds index. See also `replaceObjectsInRange:withObjectsFromArray:`, `removeObjectsInRange:`.

setArray:

```
- (void)setArray:(NSArray *)otherArray
```

Sets the receiver's contents to the elements in `otherArray`. Raises `NSInvalidArgumentException` if any member of `otherArray` is `nil` See `replaceObjectAtIndex:withObject:`.

sortUsingFunction:context:

```
- (void)sortUsingFunction:(int (*)(id element1, id element2,
    void *userData))comparator context:(void *)context
```

Sorts the receiver's elements in ascending order as defined by the comparison function `comparator`. `context` is passed as the function's third argument. See also `sortUsingSelector:`.

sortUsingSelector:

```
- (void)sortUsingSelector:(SEL)comparator
```

Sorts the receiver's elements in ascending order as defined by the comparison method `comparator`. See also `sortUsingFunction:context:`.

NSMutableCharacterSet

Characteristic	Description
Inherits From:	NSCharacterSet : NSObject
Conforms To:	NSCopying, NSMutableCopying NSCoding, NSCopying, NSMutableCopying (NSCharacterSet) NSObject (NSObject)
Declared In:	Foundation / NSCharacterSet.h

Class Description

The `NSMutableCharacterSet` class declares the programmatic interface to objects that construct mutable *descriptions* of character sets in the Unicode character encoding. Having constructed such character set descriptions using methods described in the `NSCharacterSet` class, you can use the methods described here to modify the character sets dynamically.

Method Types

Activity	Class Method
Adding and removing characters	– addCharactersInRange: – addCharactersInString: – removeCharactersInRange: – removeCharactersInString:
Combining character sets	– formIntersectionWithCharacterSet: – formUnionWithCharacterSet:
Inverting character set	– invert

Instance Methods

addCharactersInRange:

`– (void)addCharactersInRange:(NSRange)aRange`

Adds the Unicode characters in `aRange` to the receiver.

addCharactersInString:

`– (void)addCharactersInString:(NSString *)aString`

Adds the characters in `aString` to those in the receiver.

formIntersectionWithCharacterSet:

`– (void)formIntersectionWithCharacterSet:(NSCharacterSet *)otherSet`

Modifies the receiver so that it contains only those characters that exist in both the receiver and in `otherSet`.

formUnionWithCharacterSet:

`– (void)formUnionWithCharacterSet:(NSCharacterSet *)otherSet`

Modifies the receiver so that it contains all characters that exist in either the receiver or `otherSet`, barring duplicates.

invert

```
- (void)invert
```

Replaces all of the characters in the receiver with all the characters it didn't previously contain.

removeCharactersInRange:

```
- (void)removeCharactersInRange:(NSRange)aRange
```

Removes from the receiver the Unicode characters whose values are given by `aRange`.

removeCharactersInString:

```
- (void)removeCharactersInString:(NSString *)aString
```

Removes from the receiver the characters in `aString`.

NSMutableData

Characteristic	Description
Inherits From:	NSData : NSObject
Conforms To:	NSCoding, NSCopying, NSMutableCopying (NSData) NSObject (NSObject)
Declared In:	Foundation / NSData.h Foundation / NSSerialization.h

Class Description

The `NSMutableData` class declares the programmatic interface to objects that contain modifiable data in the form of bytes. This class inherits all read-only access methods from its superclass, `NSData`, and declares only those methods that permit the modification of the data.

`NSMutableData`'s two primitive methods—`mutableBytes` and `setLength:`—provide the basis for all the other methods in its interface. The `mutableBytes` method returns a pointer for writing into the bytes contained in the mutable data object. `setLength:` allows you to truncate or extend the length of a mutable data object.

The `appendBytes:length:` and `appendData:` methods let you append bytes or the contents of another data object to a mutable data object. You can replace a range of bytes in a mutable data object with either zeroes by using the `resetBytesInRange:` method, or with different bytes by using the `replaceBytesInRange:withBytes:` method.

This class declares various serialization methods that enable architecture-independent serialization of arbitrary Objective C types.

Method Types

Activity	Class Method
Creating an NSMutableData object	+ allocWithZone: + dataWithCapacity: + dataWithLength: – initWithCapacity: – initWithLength:
Adjusting capacity	– increaseLengthBy: – mutableBytes – setLength:
Appending data	– appendBytes:length: – appendData:
Modifying data	– replaceBytesInRange:withBytes: – resetBytesInRange:
Serializing data	– serializeAlignedBytesLength: – serializeDataAt:ofObjCType:context: – serializeInt: – serializeInt:atIndex: – serializeInts:count: – serializeInts:count:atIndex:

Class Methods

allocWithZone:

```
+ (id)allocWithZone:(NSZone *)zone
```

Creates and returns an uninitialized mutable data object from `zone`.

dataWithCapacity:

```
+ (id)dataWithCapacity:(unsigned int)numBytes
```

Creates and returns a mutable data object, initially allocating enough memory to hold `numBytes` bytes.

dataWithLength:

```
+ (id)dataWithLength:(unsigned int)length
```

Creates and returns a mutable data object, giving it enough memory to hold `length` bytes. Fills the object with zeroes up to `length`.

Instance Methods

appendBytes:length:

```
– (void)appendBytes:(const void *)bytes length:(unsigned int)length
```

Appends `length` bytes to a mutable data object from the buffer `bytes`. See also `serializeAlignedBytesLength:`.

appendData:

```
– (void)appendData:(NSData *)other
```

Appends the contents of the data object `other` to the receiver.

increaseLengthBy:

```
– (void)increaseLengthBy:(unsigned int)extraLength
```

Increases the length of a mutable data object by `extraLength` zero-filled bytes.

initWithCapacity:

```
– (id)initWithCapacity:(unsigned int)capacity
```

Initializes a newly allocated mutable data object, giving it enough memory to hold `capacity` bytes. Sets the length of the data object to 0.

initWithLength:

```
– (id)initWithLength:(unsigned int)length
```

Initializes a newly allocated mutable data object, giving it enough memory to hold `length` bytes. Fills the object with zeroes up to `length`.

mutableBytes

```
- (void *)mutableBytes
```

Returns a pointer to the bytes in a mutable data object, enabling you to modify the bytes.

replaceBytesInRange:withBytes:

```
- (void)replaceBytesInRange:(NSRange)aRange
    withBytes:(const void *)bytes
```

Replaces the receiver's bytes located in `aRange` with `bytes`. Raises an `NSRangeException` if `aRange` is not within the range of the receiver's data.

resetBytesInRange:

```
- (void)resetBytesInRange:(NSRange)aRange
```

Replaces the receiver's bytes located in `aRange` with zeros. Raises an `NSRangeException` if `aRange` is not within the range of the receiver's data.

serializeAlignedBytesLength:

```
- (void)serializeAlignedBytesLength:(unsigned int)length
```

Prepares bytes for an `appendBytes:length:` invocation by serializing them. If the `length` of the bytes will cause extension past the page size, this method encodes header information, creating a hole so that all bytes in the data object are aligned on page boundaries.

serializeDataAt:ofObjCType:context:

```
- (void)serializeDataAt:(const void *)data
    ofObjCType:(const char *)type
    context:(id <NSObjCTypeSerializationCallBack>)callback
```

Serializes whatever data element is referenced by `data`, interpreting it by the Objective C type specifier `type`.If the data element is an object other than an instance of `NSDictionary`, `NSArray`, `NSString`, or `NSData`, further definition of the object can occur through a callback from object `callback`. All Objective C types are currently supported except `union`s and `void *`. Pointers refer to a single item.

serializeInt:

```
- (void)serializeInt:(int)value
```

Serializes the integer `value` by encoding it as a character representation.

serializeInt:atIndex:

```
- (void)serializeInt:(int)value atIndex:(unsigned int)index
```

Serializes the integer `value` by encoding it as a character representation and replaces the encoded value at the specified `index` in the data.

serializeInts:count:

```
- (void)serializeInts:(int *)intBuffer count:(unsigned int)numInts
```

Serializes `numInts` count of integers in `intBuffer` by encoding each integer as a character representation.

serializeInts:count:atIndex:

```
- (void)serializeInts:(int *)intBuffer count:(unsigned int)numInts
    atIndex:(unsigned int)index
```

Serializes `numInts` count of integers in `intBuffer` by encoding each integer, starting at the specified `index`, and replacing each corresponding integer encoding serially.

setLength:

```
- (void)setLength:(unsigned int)length
```

Extends or truncates the length of a mutable data object by `length` bytes. If the mutable data object is extended, the additional bytes are zero-filled.

5

NSMutableDictionary

Characteristic	Description
Inherits From:	NSDictionary : NSObject
Conforms To:	NSCoding, NSCopying, NSMutableCopying (NSDictionary) NSObject (NSObject)
Declared In:	Foundation / NSDictionary.h

Class Description

The `NSMutableDictionary` class declares the programmatic interface to objects that manage mutable associations of keys and values. With its two efficient primitive methods—`setObject:forKey:` and `removeObject:forKey:`—this class adds modification operations to the basic operations it inherits from `NSDictionary`.

The other methods declared here operate by invoking one or both of these primitives. The derived methods provide convenient ways of adding or removing multiple entries at a time.

When an entry is removed from a mutable dictionary, the key and value objects that make up the entry receive a `release` message, which can cause them to be deallocated. Note that if your program keeps a reference to such objects, the reference will become invalid unless you remember to send the object a `retain` message before it is removed from the dictionary. For example, the third statement following could result in a run-time error, except for the `retain` message in the first statement:

```
id anObject = [[aDictionary objectForKey:theKey] retain];
[aDictionary removeObjectForKey:theKey];
[anObject someMessage];
```

Method Types

Activity	Class Method
Allocating and initializing	+ allocWithZone: + dictionaryWithCapacity: – initWithCapacity:
Adding and removing entries	– addEntriesFromDictionary: – removeAllObjects – removeObjectsForKeys: – setObject:forKey: – setDictionary:

Class Methods

allocWithZone:

```
+ (id)allocWithZone:(NSZone *)zone
```

Creates and returns an uninitialized `NSMutableDictionary` in `zone`.

dictionaryWithCapacity:

```
+ (id)dictionaryWithCapacity:(unsigned int)aNumItems
```

Creates and returns an `NSMutableDictionary`, giving it enough allocated memory to hold `numEntries` entries.

Instance Methods

addEntriesFromDictionary:

```
– (void)addEntriesFromDictionary:(NSDictionary *)otherDictionary
```

Adds the entries from `otherDictionary` to the receiver.

initWithCapacity:

```
– (id)initWithCapacity:(unsigned int)aNumItems
```

Initializes a newly allocated NSMutableDictionary, giving it enough allocated memory to hold numEntries entries.

removeAllObjects

```
- (void)removeAllObjects
```

Empties the receiver of its entries.

removeObjectForKey:

```
- (void)removeObjectForKey:(id)theKey
```

Removes theKey and its associated value object from the dictionary. Raises NSInvalidArgumentException if aKey is nil.

removeObjectsForKeys:

```
- (void)removeObjectsForKeys:(NSArray *)keyArray
```

Removes from the receiver one or more entries as identified by the keys in keyArray. Raises NSInvalidArgumentException if aKey is nil.

setDictionary:

```
- (void)setDictionary:(NSDictionary *)otherDictionary
```

Sets the contents of the receiver to the keys and values in otherDictionary.

setObject:forKey:

```
- (void)setObject:(id)anObject forKey:(id)aKey
```

Adds an entry to the receiver, consisting of anObject and its corresponding key aKey. Raises NSInvalidArgumentException if either anObject or aKey is nil.

NSMutableSet

Characteristic	Description
Inherits From:	NSSet : NSObject
Conforms To:	NSCoding, NSCopying, NSMutableCopying (NSSet) NSObject (NSObject)
Declared In:	Foundation / NSSet.h

Class Description

The `NSMutableSet` class declares the programmatic interface to an object that manages a mutable set of objects. `NSMutableSet` provides support for the mathematical concept of a *set*. A set, both in its mathematical sense, and in the OpenStep implementation of `NSMutableSet`, is an *unordered* collection of distinct elements. OpenStep also provides the `NSCountedSet` class for a mutable set that can contain multiple instances of the same element, and provides the `NSSet` class for creating and managing immutable sets. In general, you should use `NSSet` unless you really need a mutable set.

Use set objects as an alternative to array objects when the order of elements is not important, but performance in testing whether an object is contained in the set *is* a consideration—while arrays are ordered, testing for membership is slower than with sets.

Note – Objects in a set must respond to `hash` and `isEqual:` methods. See the `NSObject` protocol for details on `hash` and `isEqual:`.

Generally, you instantiate an `NSMutableSet` object by sending one of the `set...` methods to the `NSMutableSet` class object, as described in the method descriptions for `NSSet`. These methods return an `NSMutableSet` object containing the elements (if any) you pass in as arguments. Newly created instances of `NSMutableSet` created by invoking the `set` method can be populated with objects using any of the `init...` methods. `initWithObjects:` is the designated initializer for this class.

Objects are added to an `NSMutableSet` using `addObject:`, which adds a single specified object to the set, `addObjectsFromArray:`, which adds all objects from a specified array to the set, or by `unionSet:`, which adds all the objects from another set to this set.

Objects are removed from an `NSMutableSet` using any of the methods `intersectSet:`, `minusSet:`, `removeAllObjects` or `removeObject:`.

Method Types

Activity	Class Method
Allocating and initializating an NSMutableSet	+ allocWithZone: + setWithCapacity: – initWithCapacity:
Adding objects	– addObject: – addObjectsFromArray: – unionSet:
Removing objects	– intersectSet: – minusSet: – removeAllObjects – removeObject:

Class Methods

`allocWithZone:`

```
+ (id)allocWithZone:(NSZone *)zone
```

Creates and returns an uninitialized set object in `zone`.

`setWithCapacity:`

```
+ (id)setWithCapacity:(unsigned)numItems
```

Creates and returns a set object, giving it enough allocated memory to hold `numItems` objects.

Instance Methods

`addObject:`

```
– (void)addObject:(id)object
```

Adds `object` to the set, unless `object` is equal to some object already in the set.

addObjectsFromArray:

```
- (void)addObjectsFromArray:(NSArray *)array
```

Adds to the set all the objects in `array`, by calling `addObject:` for each one.

initWithCapacity:

```
- (id)initWithCapacity:(unsigned)numItems
```

Initializes a newly allocated set object, giving it enough allocated memory to hold `numItems` objects.

intersectSet:

```
- (void)intersectSet:(NSSet *)otherSet
```

Removes from the receiving set every object that's *not equal* to any object in `otherSet`, by calling `removeObject:` for each one.

minusSet:

```
- (void)minusSet:(NSSet *)otherSet
```

Removes from the receiving set every object that's *equal* to some object in `otherSet`, by calling `removeObject:` for each one.

removeAllObjects

```
- (void)removeAllObjects
```

Removes all set elements. This method doesn't call `removeObject:`.

removeObject:

```
- (void)removeObject:(id)object
```

If any member of the receiving set is equal to `object`, this method removes that object from the set.

unionSet:

```
- (void)unionSet:(NSSet *)otherSet
```

Adds to the receiving set all the objects in `otherSet`, by calling `addObject:` for each one.

NSMutableString

Characteristic	Description
Inherits From:	NSString : NSObject
Conforms To:	NSCoding, NSCopying, NSMutableCopying (NSString) NSObject (NSObject)
Declared In:	Foundation / NSString.h

Class Description

`NSMutableString` (and `NSString`) declare the programmatic interface for objects that create and manage mutable, *representation-independent* character strings. For a more general overview of string classes, see the description of `NSString`.

`NSMutableString` (and `NSString`) are abstract classes for string manipulation. `NSMutableString` declares the interface to objects that inherit all the capabilities of `NSString` objects, but in addition allow for modification of the string data. `NSString` and `NSMutableString` provide factory methods that return autoreleased instances of unspecified subclasses of strings.

You can instantiate an `NSMutableString` object by sending the `stringWithCapacity:` or `localizedStringWithFormat:` method to the `NSMutableString` class object. A newly allocated `NSMutableString` object can also be initialized using the `initWithCapacity:` method, to set the string to a specified capacity.

Method Types

Activity	Class Method
Creating temporary strings	+ localizedStringWithFormat: + stringWithCapacity:
Initializing a mutable string	– initWithCapacity:
Modifying a string	– appendFormat: – appendString: – deleteCharactersInRange: – insertString:atIndex: – replaceCharactersInRange:withString: – setString:

Class Methods

localizedStringWithFormat:

```
+ (id)localizedStringWithFormat:
    (NSString *)format,...
```

Returns a string created by using `format` as a `printf()` style format string, and the following arguments as values to be substituted into the format string. The user's default locale is used for format information.

stringWithCapacity:

```
+ (id)stringWithCapacity:(unsigned int)capacity
```

Returns an empty mutable string, using `capacity` as a hint for how much initial storage to reserve.

Instance Methods

appendFormat:

```
– (void)appendFormat:(NSString *)format,...
```

Adds a constructed string to the receiver. The new characters are created by using `format` as a `printf()` style format string, and the following arguments as values to be substituted into the format string. Invokes `replaceCharactersInRange:withString:` as part of its implementation. See also `appendString:`.

appendString:

```
- (void)appendString:(NSString *)aString
```

Adds the characters of `aString` to the end of the receiver. Invokes `replaceCharactersInRange:withString:` as part of its implementation. See also `insertString:atIndex:`, `appendFormat:`.

deleteCharactersInRange:

```
- (void)deleteCharactersInRange:(NSRange)range
```

Removes from the receiver the characters in `range`. This method raises an `NSStringBoundsError` exception if any part of `range` lies beyond the end of the string. Invokes `replaceCharactersInRange:withString:` as part of its implementation.

initWithCapacity:

```
- (id)initWithCapacity:(unsigned int)capacity
```

Initializes a newly allocated mutable string object, giving it enough allocated memory to hold `capacity` characters. See also `stringWithCapacity:`.

insertString:atIndex:

```
- (void)insertString:(NSString *)aString atIndex:(unsigned)index
```

Inserts the characters of `aString` into the receiver, such that the new characters begin at `index` and the existing characters from `index` to the end are shifted by the length of `aString`. This method raises an `NSStringBoundsError` exception if `index` lies beyond the end of the string. Invokes `replaceCharactersInRange:withString:` as part of its implementation. See also `appendString:`.

replaceCharactersInRange:withString:

```
- (void)replaceCharactersInRange:(NSRange)aRange
    withString:(NSString *)aString
```

Inserts the characters of `aString` into the receiver, such that they replace the characters in `aRange`. This method raises an `NSStringBoundsError` exception if any part of `aRange` lies beyond the end of the string. See also `appendString:`.

setString:

```
- (void)setString:(NSString *)aString
```

Replaces the characters of the receiver with those in `aString`.

NSNotification

Characteristic	Description
Inherits From:	NSObject
Conforms To:	NSCopying NSObject (NSObject)
Declared In:	Foundation / NSNotification.h

Class Description

`NSNotification` objects provide a flexible way to transmit event information between objects.

Message passing—invoking a method—is the standard way to convey information between objects. However, this requires the object sending the message know what the receiving object is. At times this explicit binding of two objects is undesirable—most notably because it would tie two otherwise independent subsystems. For these instances, a looser broadcast model is introduced: An object posts a notification, which is dispatched to the appropriate receivers through a notification center.

An object may post an `NSNotification` object (referred to as a *notification object* or simply, a *notification*), which contains information about an object: the notification's name, its sender, and an optional dictionary containing other

information. Other objects can register themselves as observers to receive notification objects when they are posted. When the event happens, the registered objects receive notifications. The object posting the `NSNotification` object, the object the notification is about, and the observer of the notification may all be different objects.

An `NSNotificationCenter` object registers observers for events and notifies the observers if these events occur. An object may ask an `NSNotificationCenter` object (also known as a *notification center*) to observe an event regarding another object. If the event occurs, the posting object tells the notification center to notify its observers that this condition has occurred. The notification center then sends a notification to all observing objects. (See the class specification of `NSNotificationCenter` for more on posting notification objects.)

This notification model frees an object from concern about what objects may want to observe it. An object involved with an event—or another object—may simply post a notification about that event without knowing what objects—if any—are observing the event. The notification center takes care of distributing notifications to registered observers. Another benefit of this model is to allow multiple objects to listen for notifications, an effect that might otherwise require explicitly setting up an array.

You instantiate a notification object directly by sending the `notificationWithName:object:` or `notificationWithName:object:userInfo:` messages to the `NSNotification` class object. You can also create notifications indirectly through the `NSNotificationCenter` class using the `postNotificationName:object:` and `postNotificationName:object:userInfo:` convenience methods.

You can subclass `NSNotification` to contain information in addition to the notification name, sender, and dictionary. `NSNotification` objects are immutable objects.

The `NSNotification` class adopts the `NSCopying` protocol, making it possible to treat notifications as context-independent values that can be copied and reused. You can put notifications in an array and send the `copy` message to that array, which recursively copies every item. This essentially allows clients to deal with notifications as first class values that can be copied by collections.

Method Types

Activity	Class Method
Creating notification objects	+ notificationWithName:object: + notificationWithName:object:userInfo:
Querying a notification object	– name – object – userInfo

Class Methods

notificationWithName:object:

```
+ (NSNotification *)notificationWithName:(NSString *)aName
    object:(id)anObject
```

Returns a notification object that associates the name `aName` with the object `anObject`.

notificationWithName:object:userInfo:

```
+ (NSNotification *)notificationWithName:(NSString *)aName
    object:(id)anObject userInfo:(NSDictionary *)userInfo
```

Returns a notification object that associates the name `aName` with the object `anObject` and the dictionary of arbitrary data `userInfo`. `userInfo` may be `nil`.

Instance Methods

name

```
– (NSString *)name
```

Returns the name of the notification.

object

```
– (id)object
```

Returns the object (such as the sender) that's associated with this notification.

userInfo

```
- (NSDictionary *)userInfo
```

Returns a dictionary object associated with this notification. Returns `nil` if there is no such object.

NSNotificationCenter

Characteristic	Description
Inherits From:	NSObject
Conforms To:	NSObject (NSObject)
Declared In:	Foundation / NSNotification.h

Class Description

An `NSNotificationCenter` object (or simply, *notification center*) is essentially a notification dispatch table. It notifies all observers of events meeting specific criteria of notification and sender. This event information is encapsulated in `NSNotification` objects, also known as *notification objects*, or simply, *notifications*. Client objects register themselves as observers of a specific notification originating in another object. When the condition occurs to signal a notification, some object (which may or may not be the object observed) posts an appropriate notification object to the notification center. See the class specification of `NSNotification` for more on notification objects. The notification center dispatches a message to each observer using the selector provided by the observer, with the notification as the sole argument.

An object registers itself to observe notifications by the `addObserver:selector:name:object:` method, specifying the object and associated notification it wants to see. However, the observer need not specify both of these parameters. If it specifies only the object, it will see *all* notifications associated with that object. If the object specifies only a notification name to observe, it will see that notification for *any* object whenever it's posted.

The methods `postNotificationName:object:` and `postNotificationName:object:userInfo:` are provided as convenience methods, which both create and post notifications.

Each task has a default notification center. As an example of using the notification center, suppose your program can perform a number of conversions on text (for instance, MIF to RTF or RTF to ASCII). You have defined a class of objects that perform those conversions, `Convertor`. `Convertor` objects might be added or removed during program execution. Your program has a client object that wants to be notified when convertors are added or removed, allowing the application to reflect the available options in a pop-up list. The client object would register itself as an observer by sending the following messages to the notification center:

```
[[NSNotificationCenter defaultCenter] addObserver:self
    selector:@selector(objectAddedToConvertorList:)
    name:@"NSConvertorAdded" object:nil];
[[NSNotificationCenter defaultCenter] addObserver:self
    selector:@selector(objectRemovedFromConvertorList:
    name:@"NSConvertorRemoved" object:nil];
```

When a user installs or removes a convertor, the `Convertor` object sends one of the following messages to the notification center:

```
[[NSNotificationCenter defaultCenter]
    postNotificationName:@"NSConverterAdded" object:self];
```

or

```
[[NSNotificationCenter defaultCenter]
    postNotificationName:@"NSConverterRemoved" object:self];
```

The notification center identifies all observers who are interested in the `NSConverterAdded` or `NSConverterRemoved` notifications by invoking the method they specified in the selector argument of `addObserver:selector:name:object:`. In the case of our example observer, the selectors are `objectAddedToConvertorList:` and `objectRemovedFromConvertorList:`. Assume the Convertor class has an instance method `convertorName` that returns the name of the Convertor object. Then the `objectAddedToConvertorList:` method might have the following implementation:

```
- (void)objectAddedToConvertorList:(NSNotification *)notification
{
    Convertor *addedConvertor = [notification object];
```

```
    // Add this to our popup (it will only be added if not there)...
    [myPopUpButton addItem:[addedConvertor convertorName]];
}
```

The convertors don't need to know anything about the pop-up list or any other aspect of the user interface to your program.

Method Types

Activity	Class Method
Accessing the default notification center	+ defaultCenter
Adding and removing observers	– addObserver:selector:name:object: – removeObserver: – removeObserver:name:object:
Posting notifications	– postNotification: – postNotificationName:object: – postNotificationName:object:userInfo:

Class Methods

`defaultCenter`

```
+ (NSNotificationCenter *)defaultCenter
```

Returns the default notification center object; used for generic notifications.

Instance Methods

`addObserver:selector:name:object:`

```
– (void)addObserver:(id)anObserver
    selector:(SEL)aSelector
    name:(NSString *)notificationName object:(id)anObject
```

Registers `anObserver` and `aSelector` with the receiver so that `anObserver` receives an `aSelector` message when a notification of name `notificationName` is posted to the notification center by `anObject`. If

anObject is nil, the observer will get posted whatever the object is. If aName is nil, the observer will get posted for all notifications that match anObject. See also removeObserver:.

postNotification:

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

Posts aNotification to the notification center. Raises NSInvalidArgumentException if the name associated with aNotification is nil.

postNotificationName:object:

```
- (void)postNotificationName:(NSString *)aName object:(id)anObject
```

Creates a notification object that associates aName and anObject and posts it to the notification center.

postNotificationName:object:userInfo:

```
- (void)postNotificationName:(NSString *)aName
    object:(id)anObject userInfo:(NSDictionary *)userInfo
```

Creates a notification object that associates aName and anObject and posts it to the notification center. userInfo is a dictionary of arbitrary data that will be passed with the notification. userInfo may be nil.

removeObserver:

```
- (void)removeObserver:(id)anObserver
```

Removes anObserver as the observer of any notifications from any objects. If anObserver is nil, all observers are removed. See also removeObserver:name:object:, addObserver:selector:name:object:.

removeObserver:name:object:

```
- (void)removeObserver:(id)anObserver
    name:(NSString *)notificationName
    object:anObject
```

Removes anObserver as the observer of any notificationName notifications from anObject. If notificationName is nil, then anObserver is removed from observing any notifications from an anObject. If anObject is nil, then anObserver is removed from observing notificationName notifications from any object. If notificationName and anObject are both nil, then this method removes anObserver as an observer of any notifications from any objects (equivalent to removeObserver:). See also removeObserver:, addObserver:selector:name:object:.

NSNotificationQueue

Characteristic	Description
Inherits From:	NSObject
Conforms To:	NSObject (NSObject)
Declared In:	Foundation / NSNotificationQueue.h

Class Description

NSNotificationQueue objects (or simply, *notification queues*) act as buffers for notifications centers (instances of NSNotificationCenter). A notification queue maintains notifications (instances of NSNotification) generally in a First-In First-Out (FIFO). When a notification rises to the "top" of the queue, the queue posts it to the notification center, which in turn dispatches the notification to all objects registered as observers.

NSNotificationQueue contributes two important features to OpenStep's notification mechanism: asynchronous posting and the coalescing of notifications. With NSNotificationCenter's postNotification: and its variants, you can post a notification immediately to a notification center. However, the invocation of the method is synchronous: before the posting object can resume its thread of execution, it must wait until the notification center dispatches the notification to all observers and returns. With NSNotificationQueue's enqueueNotification:postingStyle: and enqueueNotification:postingStyle:coalesceMask:forModes:, however, you can post a notification asynchronously by putting it on the queue. These methods immediately return to the invoking object after putting the notification in the queue.

Posting to a notification queue can occur in one of three different styles. The posting style is an argument to both `enqueueNotification:...` methods:

- `NSPostWhenIdle`. The notification is posted when the run loop is idle.
- `NSPostASAP`. The notification is posted as soon as possible.
- `NSPostNow`. The notification is posted immediately to the notification center.

Note – See "Enqueuing with the Different Posting Styles," following, for details on and examples of enqueuing notifications with the three `postingStyle:` constants.

What is the difference between enqueuing notifications with `NSPostNow` and posting notifications (`postNotification:`)? Both post notifications immediately (but synchronously) to the notification center. The difference is that `enqueueNotification:...` (with `NSPostNow` as posting style) coalesces notifications in the queue before posting while `postNotification:` does not.

Coalescing is a process that removes notifications in the queue that are similar to the notification just enqueued (or posted, if posting style is `NSPostNow`. The notification queue scans the notifications in the queue for those with attributes matching the new notification and removes them, except for the notification that is topmost in the queue (closest to being posted). You indicate the criteria for similarity by specifying the `NSNotificationCoalescing` constants in the third argument of `enqueueNotification:postingStyle: coalesceMask:forModes:` (logically OR them in if multiple):

- `NSNotificationNoCoalescing`. Do not coalesce notifications in the queue.
- `NSNotificationCoalescingOnName`. Coalesce notifications with the same name.
- `NSNotificationCoalescingOnSender`. Coalesce notifications with the same sender.

Every thread has a default notification queue, which is associated with the thread's default notification center. You can create your own notification queues, and have multiple queues per center and thread; but you can have only one notification center per thread. `NSNotificationQueue` is a public, concrete class; instances of it are mutable.

Enqueuing with the Different Posting Styles

Any notification enqueued with the `NSPostASAP` posting style is posted to the notification center when the code executing in the current run loop callout completes. Callouts can be Application Kit event messages, file descriptor changes, timers, or another asynchronous notification. You'd typically use the `NSPostASAP` posting style for an expensive resource, like the Display PostScript server. When many clients draw on the window buffer during a callout, it is expensive to flush the buffer to the Display PostScript server after every draw operation. In this case, each `draw...` method enqueues some notification such as "FlushTheServer" with coalescing on name and sender specified, and a posting style of `NSPostASAP`. As a result, only one of those notifications is dispatched at the end of the current callout, and the window buffer is flushed only once.

A notification enqueued with the `NSPostWhenIdle` posting style is posted only when the run loop is in a wait state. In this state, there is nothing in the run loop's input channels, be it timers or other asynchronous notifications. A typical example of enqueuing with the `NSPostIdle` posting style occurs when the user types text, and the program displays the size of the text in bytes somewhere. It would be very expensive and not very useful to update the displayed size after each character the user types, especially if the user types fast. In this case, the program enqueues a notification after each character typed such as "`ChangeTheDisplayedSize`" with coalescing turned on and a posting style of `NSPostWhenIdle`. When the user stops typing, the single "`ChangeTheDisplayedSize`" notification in the queue (due to coalescing) is posted when the run loop is in a wait state and the display is updated.

A notification enqueued with `NSPostNow` is posted immediately to the notification center. You enqueue a notification with `NSPostNow` or post one with `NSNotificationCenter`'s `postNotification:` when you do not require asynchronous calling behavior. For many programming situations, synchronous behavior is not only allowable but desirable; you want the notification center to return after dispatching so you can be sure that observing objects have received the notification. Of course, you should enqueue with `NSPostNow` rather than use `postNotification:` when there are similar notifications in the queue that you want to remove through coalescing.

Method Types

Activity	Class Method
Creating notification queues	+ defaultQueue – init – initWithNotificationCenter:
Inserting and removing notifications from a queue	– dequeueNotificationsMatching:coalesceMask: – enqueueNotification:postingStyle: – enqueueNotification:postingStyle:coalesceMask: forModes:

Class Methods

defaultQueue

```
+ (NSNotificationQueue *)defaultQueue
```

Returns the default `NSNotificationQueue` object for the current thread. This object always uses the default notification-center object for the same thread.

Instance Methods

dequeueNotificationsMatching:coalesceMask:

```
– (void)dequeueNotificationsMatching:(NSNotification *)notification
    coalesceMask:(unsigned int)coalesceMask
```

Removes all notifications from the queue that match the `notification`'s attributes as specified by `coalesceMask`. The mask (set through `NSNotificationCoalescing` constants) can specify notification name, notification sender, or both name and sender.

enqueueNotification:postingStyle:

```
– (void)enqueueNotification:(NSNotification *)notification
    postingStyle:(NSPostingStyle)postingStyle
```

Puts a `notification` in the queue that the queue will post to the notification center at the time indicated by `postingStyle`. The notification queue posts in all runloop modes, and it coalesces only notifications in the queue that match both the name and sender of `notification`.

enqueueNotification:postingStyle:coalesceMask:forModes:

```
- (void)enqueueNotification:(NSNotification *)notification
    postingStyle:(NSPostingStyle)postingStyle
    coalesceMask:(unsigned int)coalesceMask
    forModes:(NSArray *)modes
```

Puts a `notification` in the queue that the queue will post to the notification center at the time indicated by `postingStyle`, but only if the runloop is in a mode identified by one of the string objects in the `modes` array. The notification queue coalesces related notifications in the queue as specified by `coalesceMask`. If `modes` is `nil`, all runloop modes are valid for posting.

init

```
- (id)init
```

Initializes and returns an `NSNotificationQueue` object that uses the default notification-center object.

initWithNotificationCenter:

```
- (id)initWithNotificationCenter:
    (NSNotificationCenter *)notificationCenter
```

Initializes and returns an `NSNotificationQueue` object that uses the notification-center object specified in `notificationCenter`.

NSNumber

Characteristic	Description
Inherits From:	NSValue : NSObject
Conforms To:	NSCoding, NSCopying (NSValue) NSObject (NSObject)
Declared In:	Foundation / NSValue.h

Class Description

`NSNumber` objects provide an object-oriented wrapper for the standard C-language number data types (`int`, `double`, etc.). The Foundation Kit's collection classes can store only objects, so this class provides a way to prepare numbers of various types for use with the collection classes.

`NSNumber`, which inherits from `NSValue`, provides methods for creating number objects that contain data of a specified type. It also provides methods for extracting data from a number object and casting the data to be of a particular type. For determining whether two number objects are equal, `NSNumber` provides the `compare:` method.

5

Method Types

Activity	Class Method
Allocating and initializing	+ numberWithBool: + numberWithChar: + numberWithDouble: + numberWithFloat: + numberWithInt: + numberWithLong: + numberWithLongLong: + numberWithShort: + numberWithUnsignedChar: + numberWithUnsignedInt: + numberWithUnsignedLong: + numberWithUnsignedLongLong: + numberWithUnsignedShort: – initWithBool: – initWithChar: – initWithDouble: – initWithFloat: – initWithInt: – initWithLong: – initWithLongLong: – initWithShort: – initWithUnsignedChar: – initWithUnsignedInt: – initWithUnsignedLong: – initWithUnsignedLongLong: – initWithUnsignedShort:
Accessing data	– boolValue – charValue – descriptionWithLocale: – doubleValue – floatValue – intValue – longLongValue – longValue – shortValue – stringValue – unsignedCharValue – unsignedIntValue – unsignedLongLongValue – unsignedLongValue – unsignedShortValue

Activity	Class Method
Comparing data	– compare: – isEqualToNumber:

Class Methods

numberWithBool:

```
+ (NSNumber *)numberWithBool:(BOOL)value
```

Creates and returns a number object representing `value` of the type `BOOL`.

numberWithChar:

```
+ (NSNumber *)numberWithChar:(char)value
```

Creates and returns a number object representing `value` of the type `char`.

numberWithDouble:

```
+ (NSNumber *)numberWithDouble:(double)value
```

Creates and returns a number object representing `value` of the type `double`.

numberWithFloat:

```
+ (NSNumber *)numberWithFloat:(float)value
```

Creates and returns a number object representing `value` of the type `float`.

numberWithInt:

```
+ (NSNumber *)numberWithInt:(int)value
```

Creates and returns a number object representing `value` of the type `int`.

numberWithLong:

```
+ (NSNumber *)numberWithLong:(long)value
```

Creates and returns a number object representing `value` of the type `long`.

numberWithLongLong:

```
+ (NSNumber *)numberWithLongLong:(long long)value
```

Creates and returns a number object representing `value` of the type `long long`.

numberWithShort:

```
+ (NSNumber *)numberWithShort:(short)value
```

Creates and returns a number object representing `value` of the type `short`.

numberWithUnsignedChar:

```
+ (NSNumber *)numberWithUnsignedChar:(unsigned char)value
```

Creates and returns a number object representing `value` of the type `unsigned char`.

numberWithUnsignedInt:

```
+ (NSNumber *)numberWithUnsignedInt:(unsigned int)value
```

Creates and returns a number object representing `value` of the type `unsigned int`.

numberWithUnsignedLong:

```
+ (NSNumber *)numberWithUnsignedLong:(unsigned long)value
```

Creates and returns a number object representing `value` of the type `unsigned long`.

numberWithUnsignedLongLong:

```
+ (NSNumber *)numberWithUnsignedLongLong:(unsigned long long)value
```

Creates and returns a number object representing `value` of the type `unsigned long long`.

numberWithUnsignedShort:

```
+ (NSNumber *)numberWithUnsignedShort:(unsigned short)value
```

Creates and returns a number object representing `value` of the type `unsigned short`.

Instance Methods

boolValue

```
- (BOOL)boolValue
```

Returns the receiver's value as a Boolean value.

charValue

```
- (char)charValue
```

Returns the receiver's value as a character value.

compare:

```
- (NSComparisonResult)compare:(NSNumber *)otherNumber
```

Compares the receiver to `otherNumber`, using ANSI C rules for type coercion, and returns an `NSComparisonResult` (see the "Searching" section of the Foundation Kit's Types and Constants chapter).

descriptionWithLocale:

```
- (NSString *)descriptionWithLocale:
    (NSDictionary *)localeDictionary
```

Returns a string representation of the `NSSet` object, including the keys and values that represent the locale data from `localeDictionary`.

doubleValue

```
- (double)doubleValue
```

Returns the receiver's value as a double-precision floating-point value.

floatValue

```
– (float)floatValue
```

Returns the receiver's value as a single-precision floating-point value.

initWithBool:

```
- (id)initWithBool:(BOOL)value
```

Initializes the receiving number object to `value`.

initWithChar:

```
- (id)initWithChar:(char)value
```

Initializes the receiving number object to `value`. See also `initWithUnsignedChar:`.

initWithDouble:

```
- (id)initWithDouble:(double)value
```

Initializes the receiving number object to `value`. See also `initWithFloat:`.

initWithFloat:

```
- (id)initWithFloat:(float)value
```

Initializes the receiving number object to `value` See also `initWithDouble:`.

initWithInt:

```
- (id)initWithInt:(int)value
```

Initializes the receiving number object to `value`. See also `initWithUnsignedInt:`.

initWithLong:

```
- (id)initWithLong:(long)value
```

Initializes the receiving number object to `value`. See also `initWithUnsignedLong:`.

initWithLongLong:

```
- (id)initWithLongLong:(long long)value
```

Initializes the receiving number object to `value`. See also `initWithUnsignedLongLong:`.

initWithShort:

```
- (id)initWithShort:(short)value
```

Initializes the receiving number object to `value`. See also `initWithUnsignedShort:`.

initWithUnsignedChar:

```
- (id)initWithUnsignedChar:(unsigned char)value
```

Initializes the receiving number object to `value`. See also `initWithChar:`.

initWithUnsignedInt:

```
- (id)initWithUnsignedInt:(unsigned int)value
```

Initializes the receiving number object to `value`. See also `initWithInt:`.

initWithUnsignedLong:

```
- (id)initWithUnsignedLong:(unsigned long)value
```

Initializes the receiving number object to `value`. See also `initWithLong:`.

initWithUnsignedLongLong:

```
- (id)initWithUnsignedLongLong:(unsigned long long)value
```

Initializes the receiving number object to `value`. See also `initWithLongLong:`.

initWithUnsignedShort:

```
- (id)initWithUnsignedShort:(unsigned short)value
```

Initializes the receiving number object to `value`. See also `initWithShort:`.

intValue

```
– (int)intValue
```

Returns the receiver's value as a integer value.

isEqualToNumber:

```
- (BOOL)isEqualToNumber:(NSNumber *)otherNumber
```

Returns YES if otherNumber is equal to the receiver, and returns NO otherwise.

longLongValue

```
– (long long)longLongValue
```

Returns the receiver's value as a `long long` double-precision floating-point value.

longValue

```
– (long)longValue
```

Returns the receiver's value as a `long` double-precision floating-point value.

shortValue

```
– (short)shortValue
```

Returns the receiver's value as a `short` integer value.

stringValue

```
- (NSString *)stringValue
```

Returns the receiver's value as a string contained in an `NSString` object.

unsignedCharValue

```
- (unsigned char)unsignedCharValue
```

Returns the receiver's value as an `unsigned char` value.

unsignedIntValue

```
- (unsigned int)unsignedIntValue
```

Returns the receiver's value as an unsigned integer value.

unsignedLongLongValue

```
- (unsigned long long)unsignedLongLongValue
```

Returns the receiver's value as an `unsigned long long` double-precision floating-point value.

unsignedLongValue

```
- (unsigned long)unsignedLongValue
```

Returns the receiver's value as an `unsigned long` double-precision floating-point value.

unsignedShortValue

```
- (unsigned short)unsignedShortValue
```

Returns the receiver's value as an `unsigned short` integer value.

NSObject

Characteristic	Description
Inherits From:	none (NSObject is the root class)
Conforms To:	NSObject
Declared In:	Foundation / NSObject.h Foundation / NSRunLoop.h

Class Description

`NSObject` is the root class of all ordinary Objective C inheritance hierarchies; it has no superclass. Its interface derives from two sources: the methods it declares directly and those declared in the `NSObject` protocol. Its interface is divided in this way so that objects inheriting from other root classes (notably `NSProxy`) can stand in for ordinary objects without having to inherit from `NSObject`. The following discussion makes no distinction between the methods declared in this class and those declared in the `NSObject` protocol.

From `NSObject`, other classes inherit a basic interface to the run-time system for the Objective C language. It is through `NSObject` that instances of all classes obtain their ability to behave as objects. Among other things, the `NSObject` class provides inheriting classes with a framework for creating, initializing, deallocating, comparing, and archiving objects, for performing methods selected at run-time, for querying an object about its methods and its position in the inheritance hierarchy, and for forwarding messages to other objects. For example, to ask an object what class it belongs to, you would send it a `class` message. To find out whether it implements a particular method, you would send it a `respondsToSelector:` message.

The `NSObject` class is an abstract class; programs use instances of classes that inherit from `NSObject`, but never of `NSObject` itself. See also the `NSObject` protocol (Application Kit's Types and Constants chapter).

Initializing an Object to Its Class

Every object is connected to the run-time system through its `isa` instance variable, inherited from the `NSObject` class. `isa` identifies the object's class; it points to a structure that is compiled from the class definition. Through `isa`, an object can find whatever information it needs at run time—such as its place

in the inheritance hierarchy, the size and structure of its instance variables, and the location of the method implementations it can perform in response to messages.

Because all ordinary objects inherit directly or indirectly from the `NSObject` class, they all have this variable. The defining characteristic of an "object" is that its first instance variable is an `isa` pointer to a class structure.

The installation of the class structure—the initialization of `isa`—is one of the responsibilities of the `alloc` and `allocWithZone:` methods, the same methods that create (allocate memory for) new instances of a class. In other words, class initialization is part of the process of creating an object; it's not left to the methods (such as `init`) that initialize individual objects with their particular characteristics.

Instance and Class Methods

Every object requires an interface to the run-time system, whether it's an instance object or a class object. For example, it should be possible to ask either an instance or a class whether it can respond to a particular message. So that this won't mean implementing every `NSObject` method twice, once as an instance method and again as a class method, the run-time system treats methods defined in the root class in a special way: *Instance methods defined in the root class can be performed both by instances and by class objects.*

A class object has access to class methods—those defined in the class and those inherited from the classes above it in the inheritance hierarchy—but generally not to instance methods. However, the run-time system gives all class objects access to the instance methods defined in the root class. Any class object can perform any root instance method, provided it doesn't have a class method with the same name.

For example, a class object could be sent messages to perform `NSObject`'s `respondsToSelector:` and `performSelector:withObject:` instance methods:

```
SEL method = @selector(riskAll:);

if ( [MyClass respondsToSelector:method] )
    [MyClass performSelector:method withObject:self];
```

When a class object receives a message, the run-time system looks first at the receiver's set of class methods. If it fails to find a class method that can respond to the message, it looks at the set of instance methods defined in the root class. If the root class has an instance method that can respond (as `NSObject` does for `respondsToSelector:` and `performSelector:withObject:`), the run-time system uses that implementation and the message succeeds.

Note that the only instance methods available to a class object are those defined in the root class. If `MyClass` in the example above had reimplemented either `respondsToSelector:` or `performSelector:withObject:`, those new versions of the methods would be available only to instances. The class object for `MyClass` could perform only the versions defined in the `NSObject` class. Of course, if `MyClass` had implemented `respondsToSelector:` or `performSelector:withObject:` as class methods rather than instance methods, the class would perform those new versions.

Method Types

Activity	Class Method
Initializing the class	+ initialize + load
Creating and destroying instances	+ alloc + allocWithZone: + new – copy + copyWithZone: – dealloc – init – mutableCopy + mutableCopyWithZone:
Identifying classes	+ class + superclass
Testing class functionality	+ instancesRespondToSelector:
Testing protocol conformance	+ conformsToProtocol:
Obtaining method information	+ instanceMethodForSelector: – methodForSelector: – methodSignatureForSelector:
Describing objects	+ description
Posing	+ poseAsClass:
Error handling	– doesNotRecognizeSelector:
Sending deferred messages	+ cancelPreviousPerformRequestsWithTarget: selector:object: – performSelector:object:afterDelay:
Forwarding messages	– forwardInvocation:
Archiving	– awakeAfterUsingCoder: – classForArchiver – classForCoder – replacementObjectForArchiver: – replacementObjectForCoder: + setVersion: + version

Class Methods

alloc

```
+ (id)alloc
```

Returns a new instance of the receiving class. The `isa` instance variable of the new object is initialized to a data structure that describes the class; memory for all other instance variables is set to 0. A version of the `init` method should be used to complete the initialization process. For example:

```
id newObject = [[TheClass alloc] init];
```

Other classes shouldn't override `alloc` to add code that initializes the new instance. Instead, class-specific versions of the `init` method should be implemented for that purpose. Versions of the `new` method can also be implemented to combine allocation and initialization. See also `allocWithZone:`, `new`.

allocWithZone:

```
+ (id)allocWithZone:(NSZone *)zone
```

Returns a new, uninitialized instance of the receiving class in `zone`. The `isa` instance variable of the new object is initialized to a data structure that describes the class; memory for all other instance variables is set to 0. A version of the `init` method should be used to complete the initialization process. For example:

```
id newObject = [[TheClass alloc] init];
```

Other classes shouldn't override `alloc` to add code that initializes the new instance. Instead, class-specific versions of the `init` method should be implemented for that purpose. Versions of the `new` method can also be implemented to combine allocation and initialization. See also `allocWithZone:`, `new`.

When one object creates another, it's often a good idea to make sure they're both allocated from the same region of memory. The `zone` (see the `NSObject` protocol) method can be used for this purpose; it returns the zone where the receiver is located. For example:

```
id myCompanion = [[TheClass allocFromZone:[self zone]] init];
```

See also alloc, new.

cancelPreviousPerformRequestsWithTarget: selector:object:

```
+ (void)cancelPreviousPerformRequestsWithTarget:(id)aTarget
    selector:(SEL)aSelector object:(id)anObject
```

Cancels previous perform requests having the same target and argument as determined by isEqual:, and having the same selector. This method removes timers only in the current run loop, not all run loops. See also isEqual: (NSObject protocol), performSelector:object:afterDelay:, NSTimer, NSRunLoop.

class

```
+ (Class)class
```

Returns self. Since this is a class method, it returns the class object. When a class is the receiver of a message, it can be referred to by name. In all other cases, the class object must be obtained through this, or a similar method. For example, here SomeClass is passed as an argument to the isKindOfClass: method (see NSObject protocol):

```
BOOL test = [self isKindOfClass:[SomeClass class]];
```

See also class (NSObject protocol), superclass.

conformsToProtocol:

```
+ (BOOL)conformsToProtocol:(Protocol *)aProtocol
```

Returns YES if the receiving class conforms to aProtocol, and NO if it doesn't. A class is said to "conform to" a protocol if it adopts the protocol or inherits from another class that adopts it. Protocols are adopted by listing them within angle brackets after the interface declaration. Here, for example, MyClass adopts the imaginary AffiliationRequests and Normalization protocols:

```
@interface MyClass : Object <AffiliationRequests, Normalization>
```

A class also conforms to any protocols that are incorporated in the protocols it adopts or inherits. Protocols incorporate other protocols in the same way that classes adopt them. For example, here the `AffiliationRequests` protocol incorporates the `Joining` protocol:

```
@protocol AffiliationRequests <Joining>
```

When a class adopts a protocol, it must implement all the methods the protocol declares. If it adopts a protocol that incorporates another protocol, it must also implement all the methods in the incorporated protocol or inherit those methods from a class that adopts it. In the previous example, `MyClass` must implement the methods in the `AffiliationRequests` and `Normalization` protocols and, in addition, either inherit from a class that adopts the `Joining` protocol or implement the `Joining` methods itself.

When these conventions are followed and all the methods in adopted and incorporated protocols are in fact implemented, the `conformsToProtocol:` (see `NSObject` protocol) test for a set of methods becomes roughly equivalent to the `respondsToSelector:` (see `NSObject` protocol) test for a single method. However, `conformsToProtocol:` judges conformance solely on the basis of the formal declarations in source code, as illustrated previously. It doesn't check to see whether the methods declared in the protocol are actually implemented. It's the programmer's responsibility to see that they are. The protocol object required as this method's argument can be specified using the `@protocol()` directive:

```
BOOL canJoin = [MyClass conformsToProtocol:@protocol(Joining)];
```

copyWithZone:

```
+ (id)copyWithZone:(NSZone *)zone
```

Returns a copy of the receiver, allocated within `zone`. The default implemenation returns `self`. See also `mutableCopyWithZone:`, `copy`, `NSZone`.

description

```
+ (NSString *)description
```

Subclasses override this method to return a human-readable string representation of the contents of the receiver. `NSObject`'s implementation simply prints the name of the receiver's class.

initialize

```
+ (void)initialize
```

Initializes the class before it is used (that is, before it receives its first message). The run-time system sends an `initialize` message to each class just before the class, or any class that inherits from it, is sent its first message from within the program. Each class object receives the `initialize` message just once. Superclasses receive it before subclasses do. For example, if the first message your program sends is this,

[NSApplication sharedApplication]

the run-time system will generate these three `initialize` messages,

[NSObject initialize];
[NSResponder initialize];
[NSApplication initialize];

since `NSApplication` is a subclass of `NSResponder`, and `NSResponder` is a subclass of `NSObject`. All the `initialize` messages precede the `new` message and are sent in the order of inheritance, as shown. If your program later begins to use the `NSText` class,

```
[NSText instancesRespondToSelector:someSelector]
```

the run-time system will generate these additional `initialize` messages,

[NSView initialize];
[NSText initialize];

since the `NSText` class inherits from `NSObject`, `NSResponder`, and `NSView`. The `instancesRespondToSelector:` message is sent only after all these classes are initialized. Note that the `initialize` messages to `NSObject` and `NSResponder` aren't repeated; each class is initialized only once. You can implement your own versions of `initialize` to provide class-specific initialization as needed.

Because `initialize` methods are inherited, it's possible for the same method to be invoked many times, once for the class that defines it and once for each inheriting class. To prevent code from being repeated each time the method is invoked, it can be bracketed as shown in the example below

```
+ initialize
{
    if ( self == [MyClass class] ) {
        /* put initialization code here */
    }
    return self;
}
```

Since the run-time system sends a class just one `initialize` message, the test shown in the previous example should prevent code from being invoked more than once. However, if for some reason an application also generates `initialize` messages, a more explicit test may be needed:

```
+ initialize
{
    static BOOL tooLate = NO;
    if ( !tooLate ) {
        /* put initialization code here */
        tooLate = YES;
    }
    return self;
}
```

See also `init`, `class`.

instanceMethodForSelector:

```
+ (IMP)instanceMethodForSelector:(SEL)aSelector
```

Locates and returns the address of the implementation of the `aSelector` instance method. Use this method to ask the class object for the implementation of an instance method. To ask the class for the implementation of a class method, use the instance method `methodForSelector:` instead of this one. `instanceMethodForSelector:`, and the function pointer it returns, are subject to the same constraints as those described for `methodForSelector:`. See also `methodForSelector:`, `methodSignatureForSelector:`, `instancesRespondToSelector:`.

instancesRespondToSelector:

```
+ (BOOL)instancesRespondToSelector:(SEL)aSelector
```

Returns YES if instances of the class are capable of responding to aSelector messages, and NO if they're not. To ask the class whether it, rather than its instances, can respond to a particular message, use the respondsToSelector: instance method instead. See also respondsToSelector: (NSObject protocol).

load

```
+ (void)load
```

Sent to classes that are added to the Objective-C runtime. Usually received before initialize. The order in which load messages are sent to classes is unspecified.

mutableCopyWithZone:

```
+ (id)mutableCopyWithZone:(NSZone *)zone
```

Returns a writeable copy of the receiver, allocated within zone. The default implemenation returns self. See also copyWithZone:, mutableCopy, NSZone (Foundation Kit Types and Constants).

new

```
+ (id)new
```

Allocates a new instance of the receiving class, sends it an init message, and returns the initialized object returned by init. This method is simply a convenient cover for the alloc and init methods. Like alloc, new initializes the isa instance variable of the new object so that it points to the class data structure. It then invokes the init method to complete the initialization process.

Unlike alloc, new is sometimes reimplemented in subclasses to invoke a class-specific initialization method. If the init method includes arguments, they're typically reflected in the new method as well. For example:

```
+ newArg:(int)tag arg:(struct info *)data
{
    return [[self alloc] initArg:tag arg:data];
}
```

However, there's little point in implementing a `new...` method if it's simply a shorthand for `alloc` and `init...`, like the one shown above. Often `new...` methods will do more than just allocation and initialization. In some classes, they manage a set of instances, returning the one with the requested properties if it already exists, allocating and initializing a new one only if necessary. For example:

```
+ newArg:(int)tag arg:(struct info *)data
{
    id theInstance;

    if ( theInstance = findTheObjectWithTheTag(tag) )
        return theInstance;
    return [[self alloc] initArg:tag arg:data];
}
```

Although it's appropriate to define new `new...` methods in this way, the `alloc` and `allocFromZone:` methods should never be augmented to include initialization code. See also `init`, `alloc`, `allocWithZone:`.

poseAsClass:

```
+ (void)poseAsClass:(Class)aClassObject
```

Causes the receiving class to "pose as" its superclass, the `aClassObject` class. The receiver takes the place of `aClassObject` in the inheritance hierarchy; all messages sent to `aClassObject` will actually be delivered to the receiver. The receiver must be defined as a subclass of `aClassObject`. It can't declare any new instance variables of its own, but it can define new methods and override methods defined in the superclass. The `poseAsClass:` message should be sent before any messages are sent to `aClassObject` and before any instances of `aClassObject` are created.

This facility allows you to add methods to an existing class by defining them in a subclass and having the subclass substitute for the existing class. The new method definitions will be inherited by all subclasses of the superclass. Care should be taken to ensure that this doesn't generate errors. A subclass that poses as its superclass still inherits from the superclass. Therefore, none of the functionality of the superclass is lost in the substitution. Posing doesn't alter the definition of either class.

Posing is useful as a debugging tool, but category definitions are a less complicated and more efficient way of augmenting existing classes. Posing admits only two possibilities that are absent for categories:

- A method defined by a posing class can override any method defined by its superclass. Methods defined in categories can replace methods defined in the class proper, but they cannot reliably replace methods defined in other categories. If two categories define the same method, one of the definitions will prevail, but there's no guarantee which one.
- A method defined by a posing class can, through a message to `super`, incorporate the superclass method it overrides. A method defined in a category can replace a method defined elsewhere by the class, but it can't incorporate the method it replaces.

If not successful, this method generates an error message and aborts.

`setVersion:`

```
+ (void)setVersion:(int)version
```

Sets the class version number to `version`. The version number is helpful when instances of the class are to be archived and reused later. The default version is 0. See also `version`.

`superclass`

```
+ (Class)superclass
```

Returns the class object for the receiver's superclass. See also `class`, `superclass` (`NSObject` protocol).

`version`

```
+ (int)version
```

Returns the version of the class definition. See also `setVersion:`.

Instance Methods

awakeAfterUsingCoder:

```
- (id)awakeAfterUsingCoder:(NSCoder *)aDecoder
```

Implemented by subclasses to reinitialize the receiving object after it has been unarchived by `aDecoder`. An `awakeAfterUsingCoder:` message is automatically sent to every object after it has been unarchived and after all the objects it refers to are in a usable state. The default version merely returns `self`.

Each implementation of `awakeAfterUsingCoder:` should limit the work it does to the scope of the class definition, and incorporate the initialization of classes farther up the inheritance hierarchy through a message to `super`. For example:

```
- awakeAfterUsingCoder:(NSCoder *) aDecoder
{
    [super awakeAfterUsingCoder:aDecoder];
    /* class-specific initialization goes here */
    return self;
}
```

All implementations should return `self`.

Note – Not all objects loaded from a nib file (created by Interface Builder) are unarchived; some are newly instantiated. Those that are unarchived receive an `awakeAfterUsingCoder:` message, but those that are instantiated do not.

See also `NSCoder`, `NSArchiver`.

classForArchiver

```
- (Class)classForArchiver
```

Returns the class used during archiving. `NSObject`'s implementation returns the object returned by `classForCoder:`. See also `NSArchiver`.

classForCoder

```
- (Class)classForCoder
```

Returns the class used during serialization. An `NSObject` returns its own class by default. See also `classForArchiver`, `NSCoder`.

copy

```
- (id)copy
```

Returns a new instance that's an exact copy of the receiver. This method creates only one new object. If the receiver has instance variables that point to other objects, the instance variables in the copy will point to the same objects. The values of the instance variables are copied, but the objects they point to are not.

This method does its work by invoking the `copyWithZone:` method and specifying that the copy should be allocated from the same memory zone as the receiver. If a subclass implements its own `copyWithZone:` method, this `copy` method will use it to copy instances of the subclass. Therefore, a class can support copying from both methods just by implementing a class-specific version of `copyFromZone:`. See also `copyWithZone:` (`NSCopying` protocol).

dealloc

```
- (void)dealloc
```

Deallocates the memory occupied by the receiver. Subsequent messages to the object will generate an error indicating that a message was sent to a deallocated object, provided that the freed memory hasn't been reused yet.

Subclasses must implement their own versions of `dealloc` to deallocate any additional memory consumed by the object—such as dynamically allocated storage for data, or other objects that are tightly coupled to the freed object and are of no use without it. After performing the class-specific deallocation, the subclass method should incorporate superclass versions of `dealloc` through a message to `super`. See also `alloc`, `allocWithZone:`, `new`.

doesNotRecognizeSelector:

```
- (void)doesNotRecognizeSelector:(SEL)aSelector
```

Handles `aSelector` messages that the receiver doesn't recognize. The run-time system invokes this method whenever an object receives an `aSelector` message that it can't respond to or forward. This method, in turn, invokes `NSLog()` to generate an error message and, raises an

NSInvalidArgumentException. doesNotRecognizeSelector: messages should be sent only by the run-time system. See also NSLog() (Foundation Kit's "Functions" chapter), NSException.

forwardInvocation:

```
- (void)forwardInvocation:(NSInvocation *)anInvocation
```

Implemented by subclasses to forward message invocations to other objects. When an object is sent a message, and the run-time system can't find an implementation of the message for the receiving object, it sends the object a forwardInvocation: message to give it an opportunity to delegate the message to another receiver. If the delegated receiver can't respond to the message either, it will also be given a chance to forward it. Thus the forwardInvocation: message allows an object to establish relationships with other objects that will, for certain messages, act on its behalf. The forwarding object is able to "inherit" some of the characteristics of the object it forwards the message to.

A forwardInvocation: message is generated only if anInvocation isn't implemented by the receiving object's class or by any of the classes it inherits from.

A forwardInvocation: method implementation has two tasks:

- To locate an object that can respond to anInvocation. This need not be the same object for all messages.
- To send anInvocation to that object.

The default implementation sends the doesNotRecognizeSelector: message.

init

```
- (id)init
```

Implemented by subclasses to initialize a new object (the receiver) immediately after memory for it has been allocated. An init message is generally coupled with an alloc or allocWithZone: message in the same line of code:

```
id newObject = [[TheClass alloc] init];
```

An object isn't ready to be used until it has been initialized. The version of the `init` method defined in the `NSObject` class does no initialization; it simply returns `self`.

Subclass versions of this method should return the new object (`self`) after it has been successfully initialized. If it can't be initialized, then it should free the object and return `nil`. In some cases, an `init` method might free the new object and return a substitute. Programs should therefore always use the object returned by `init`, and not the one returned by `alloc` or `allocWithZone:`.

Every class must guarantee that the `init` method returns a fully functional instance of the class. Typically this means overriding the method to add class-specific initialization code. Subclass versions of `init` need to incorporate the initialization code for the classes they inherit from, by sending a message to `super`. For example:

```
- init
{
   [super init];
    /* class-specific initialization goes here */
    return self;
}
```

Note that the message to `super` precedes the initialization code added in the method. This ensures that initialization proceeds in the order of inheritance.

Subclasses often add arguments to the `init` method to allow specific values to be set. The more arguments a method has, the more freedom it gives you to determine the character of initialized objects. Classes often have a set of `init...` methods, each with a different number of arguments. For example:

```
- init;
- initArg:(int)tag;
- initArg:(int)tag arg:(struct info *)data;
```

The convention is that at least one of these methods, usually the one with the most arguments, includes a message to `super` to incorporate the initialization of classes higher up the hierarchy. This method is the *designated initializer* for the class. The other `init...` methods defined in the class directly or indirectly invoke the designated initializer through messages to `self`. In this way, all `init...` methods are chained together. For example:

```
- init
{
    return [self initArg:-1];
}
```

```
- initArg:(int)tag
{
    return [self initArg:tag arg:NULL];
}

- initArg:(int)tag arg:(struct info *)data
{
    [super init. . .];
    /* class-specific initialization goes here */
}
```

In this example, the `initArg:arg:` method is the designated initializer for the class. If a subclass does any initialization of its own, it must define its own designated initializer. This method should begin by sending a message to `super` to perform the designated initializer of its superclass. For example, suppose the three methods illustrated above are defined in the `B` class. The `C` class, a subclass of `B`, might have this designated initializer:

```
- initArg:(int)tag arg:(struct info *)data arg:anObject
{
    [super initArg:tag arg:data];
    /* class-specific initialization goes here */
}
```

If inherited `init...` methods are to successfully initialize instances of the subclass, they must all be made to (directly or indirectly) invoke the new designated initializer. To accomplish this, the subclass is obliged to cover (override) only the designated initializer of the superclass. For example, in addition to its designated initializer, the `C` class would also implement this method:

```
- initArg:(int)tag arg:(struct info *)data
{
    return [self initArg:tag arg:data arg:nil];
}
```

This ensures that all three methods inherited from the B class also work for instances of the `C` class.

Often the designated initializer of the subclass overrides the designated initializer of the superclass. If so, the subclass need only implement the one `init...` method.

These conventions maintain a direct chain of `init...` links, and ensure that the `new` method and all inherited `init...` methods return usable, initialized objects. They also prevent the possibility of an infinite loop wherein a subclass method sends a message (to `super`) to perform a superclass method, which in turn sends a message (to `self`) to perform the subclass method.

This `init` method is the designated initializer for the `NSObject` class. Subclasses that do their own initialization should override it, as described above. See also `new`, `alloc`, `allocWithZone:`.

methodForSelector:

```
- (IMP)methodForSelector:(SEL)aSelector
```

Locates and returns the address of the receiver's implementation of the `aSelector` method, so that it can be called as a function. If the receiver is an instance, `aSelector` should refer to an instance method; if the receiver is a class, it should refer to a class method.

`aSelector` must be a valid, non-`NULL` selector. If in doubt, use the `respondsToSelector:` method (see the `NSObject` protocol) to check before passing the selector to `methodForSelector:`.

`IMP` is defined (in the `objc/objc.h` header file) as a pointer to a function that returns an `id` and takes a variable number of arguments in addition to the two "hidden" arguments—`self` and `_cmd`—that are passed to every method implementation:

```
typedef id (*IMP)(id, SEL, ...);
```

This definition serves as a prototype for the function pointer that `methodForSelector:` returns. It is sufficient for methods that return an object and take object arguments. However, if the `aSelector` method takes different argument types or returns anything but an `id`, its function counterpart will be inadequately prototyped. Lacking a prototype, the compiler will promote `float`s to `double`s and `char`s to `int`s, which the implementation won't expect. It will therefore behave differently (and erroneously) when called as a function than when performed as a method.

To remedy this situation, it is necessary to provide your own prototype. In the example below, the declaration of the `test` variable serves to prototype the implementation of the `isEqual:` method. `test` is defined as pointer to a

function that returns a BOOL and takes an id argument (in addition to the two "hidden" arguments). The value returned by methodForSelector: is then similarly cast to be a pointer to this same function type:

```
BOOL (*test)(id, SEL, id);
test = (BOOL (*)(id, SEL, id))[target
methodForSelector:@selector(isEqual:)];

while ( !test(target, @selector(isEqual:), someObject) ) {
    . . .
}
```

In some cases, it might be clearer to define a type (similar to IMP) that can be used both for declaring the variable and for casting the function pointer methodForSelector: returns. The example below defines the EqualIMP type for just this purpose:

```
typedef BOOL (*EqualIMP)(id, SEL, id);
EqualIMP test;
test = (EqualIMP)[target methodForSelector:@selector(isEqual:)];

while ( !test(target, @selector(isEqual:), someObject) ) {
    . . .
}
```

Either way, it's important to cast methodForSelector:'s return value to the appropriate function type. It's not sufficient to simply call the function returned by methodForSelector: and cast the result of that call to the desired type. This can result in errors.

Note – Turning a method into a function by obtaining the address of its implementation "unhides" the self and _cmd arguments. See also instanceMethodForSelector:, methodSignatureForSelector:.

---

methodSignatureForSelector:

```
- (NSMethodSignature *)methodSignatureForSelector:(SEL)aSelector
```

Returns an object that contains an encoded description of the aSelector method, or nil if the aSelector method can't be found. When the receiver is an instance, aSelector should be an instance method; when the receiver is a class, it should be a class method. See also NSMethodSignature.

mutableCopy

```
- (id)mutableCopy
```

Invokes `mutableCopyWithZone:` (see `NSMutableCopying` protocol). This method is implemented in `NSObject` as a convenience to subclasses. A subclass need override only `mutableCopyWithZone:` for both `mutableCopy` and `mutableCopyWithZone:` to operate correctly. See also `copy`.

performSelector:object:afterDelay:

```
- (void)performSelector:(SEL)aSelector object:(id)anObject
    afterDelay:(NSTimeInterval)delay
```

Sends the receiver an `aSelector` message, with `anObject` as its argument, after `delay`. If `delay` is 0, then `aSelector` is performed on the next event loop. `anObject` is retained until after the action is executed. See also `cancelPreviousPerformRequestsWithTarget: selector:object:`, `NSTimeInterval` (Foundation Kit's "Types and Constants" chapter).

replacementObjectForArchiver:

```
- (id)replacementObjectForArchiver:(NSArchiver *)anArchiver
```

Allows an object to substitute another object for itself during archiving. `NSObject`'s implementation returns the object returned by `replacementObjectForCoder:`. See also `NSArchiver`.

replacementObjectForCoder:

```
- (id)replacementObjectForCoder:(NSCoder *)anEncoder
```

Allows an object to substitute another object for itself during serialization. `NSObject`'s implementation returns `self`. See also `NSCoder`.

NSPosixFileDescriptor

Characteristic	Description
Inherits From:	NSObject
Conforms To:	NSObject(NSObject)
Declared In:	Foundation / NSPosixFileDescriptor.h

Class Description

An `NSPosixFileDescriptor` is an object that references an input/output stream. Use `NSPosixFileDescriptors` to open, read data from, and write data to various system entities:

- files
- devices (such as terminals)
- pipes
- sockets

You can also use `NSPosixFileDescriptors` to map files into virtual memory, to truncate files, and to monitor the activity of data through devices and sockets. Using `NSPosixFileDescriptor` methods to read from and write to files has some advantages over other OpenStep API for file access. For one, `NSPosixFileDescriptor` permits random access to the bytes within files. Another advantage is the ability to read files in (and write them out) incrementally, a feature especially useful when you have large files and limited memory. However, for most purposes it is more efficient to access file contents with "higher-level" OpenStep API, such as `NSData`'s `initWithContentsOfMappedFile:`.

Note – NSPosixFileDescriptor is not part of the OpenStep specification.

Method Types

Activity	Class Method
Getting a standard NSPosixFileDescriptor	– initWithStandardError – initWithStandardInput – initWithStandardOutput
Creating an NSPosixFileDescriptor	– initWithFileDescriptor: – initWithPath: – initWithPath:flags: – initWithPath:flags:createMode:
Getting a file descriptor	– fd
Reading from a file descriptor	– readEntireFile – readFileLength: – readFileRange: – readRestOfFile
Writing to a file descriptor	– writeData: – writeData:range:
Seeking within a file descriptor	– position – seekToEnd – seekToPosition:
Mapping files into memory	– mapFile – mapFileRange:
Truncating files	– truncateAtPosition: – synchronize
Setting and getting the delegate	– delegate – setDelegate:
Monitoring descriptors	– activity – ceaseMonitoringActivity – monitorActivity: – monitorActivity:delegate:
Methods implemented by the delegate	– activity:posixFileDescriptor:

Instance Methods

activity

```
- (NSPosixFileActivities)activity
```

Returns the current activity of the `NSPosixFileDescriptor` being monitored. Activities include reading, writing, exceptions, and no activity. See also `ceaseMonitoringActivity`, `monitorActivity:`, `monitorActivity:delegate:`.

ceaseMonitoringActivity

```
- (void)ceaseMonitoringActivity
```

Removes the receiver from the run loop monitoring descriptor file activity. See also `monitorActivity:`, `monitorActivity:delegate:`.

delegate

```
- delegate
```

Returns the delegate for the `NSPosixFileDescriptor`. See also `monitorActivity:delegate:`, `setDelegate:`.

fd

```
- (int)fd
```

Returns the descriptor integer associated with the receiver.

initWithFileDescriptor:

```
-(id)initWithFileDescriptor:(int)fileDescriptor
```

Returns an `NSPosixFileDescriptor` initialized with `fileDescriptor`. See also `initWithStandardError`, `initWithStandardInput`, `initWithStandardOutput`.

initWithPath:

```
- (id)initWithPath:(NSString *)path
```

Returns a read-only NSPosixFileDescriptor initialized to reference the file identified by path. See also initWithPath:flags:.

initWithPath:flags:

```
-(id)initWithPath:(NSString *)path flags:(int)flags
```

Returns an NSPosixFileDescriptor initialized to reference the file identified by path. The flags argument contains file-access attributes such as O_RDONLY (read-only), O_RDWR (read-write), or O_APPEND (append on each write). Compatible attributes can be OR'd together. However, never invoke this method with an O_CREAT flag. See open(2) in the man pages for a complete list of allowable flags. See also initWithPath:flags:createMode:.

initWithPath:flags:createMode:

```
-(id)initWithPath:(NSString *)path
    flags:(int)flags createMode:(int)mode
```

Returns an NSPosixFileDescriptor initialized to reference the file identified by path. The flags argument contains file-access attributes such as O_WRONLY (write-only), O_RDWR (read-write), or O_NDELAY (do not block on open). Compatible attributes can be OR'd together. See the system routine open(2) in the man pages for a complete list of allowable flags. mode specifies the file mode (that is, access permissions) for created NSPosixFileDescriptors (O_CREAT flag); see the system routine chmod() in the man pages for a list of allowable values. See also initWithPath:.

initWithStandardError

```
-(id)InitWithStandardError
```

Returns an NSPosixFileDescriptor initialized with the standard error device. See also initWithFileDescriptor:, initWithPath:.

initWithStandardInput

```
-(id)initWithStandardInput
```

Returns an NSPosixFileDescriptor initialized with the standard input device. See also: initWithFileDescriptor:, initWithPath:.

initWithStandardOutput

```
-(id)initWithStandardOutput
```

Returns an NSPosixFileDescriptor initialized with the standard output device. See also initWithStandardError, initWithFileDescriptor:, initWithPath:.

mapFile

```
- (NSData *)mapFile
```

Maps the receiver into memory. As bytes are accessed, the operating system brings new pages from disk to memory automatically. See also mapFileRange:.

mapFileRange:

```
- (NSData *)mapFileRange:(NSRange)range
```

Maps the range of bytes in the file referenced by the receiver into memory. As bytes are accessed, the operating system brings new pages from disk to memory automatically. If there is an error in mapping, the method returns nil. See also synchronize, mapFile.

monitorActivity:

```
- (void)monitorActivity:(NSPosixFileActivities)activity
```

Adds the receiver to the list of descriptors monitored in a run loop for activity. The delegate for the receiving NSPosixFileDescriptor (if one has been specified) is notified via the delegate method activity:posixFileDescriptor: when the descriptor has data for reading, can accept data for writing, or has an exceptional condition pending.

monitorActivity:delegate:

```
- (void)monitorActivity:(NSPosixFileActivities)activity
    delegate:(id)delegate
```

Adds the receiver to the list of descriptors monitored in a run loop for activity. The delegate for the receiving `NSPosixFileDescriptor` (specified in `delegate`) is notified via the delegate method `activity:posixFileDescriptor:` when the descriptor has data for reading, can accept data for writing, or has an exceptional condition pending.

position

```
- (unsigned)position
```

Returns the position of the file pointer within the file referenced by the receiver. See also `readEntireFile`, `seekToEnd`, `writeData:`.

readEntireFile

```
- (NSData *)readEntireFile
```

Returns the contents of the file referenced by the receiver. See also `readFileLength:`.

readFileLength:

```
- (NSData *)readFileLength:(unsigned int)length
```

Returns the contents of the file, socket, named pipe, or device referenced by the receiver up to the byte identified by `length`. See also `readFileRange:`.

readFileRange:

```
- (NSData *)readFileRange:(NSRange)range
```

Returns the byte `range` of date in the file referenced by the receiver. See also `readRestOfFile`.

readRestOfFile

```
- (NSData *)readRestOfFile
```

Returns the contents of the file, socket, named pipe, or device referenced by the receiver from the current file pointer. See also `position`, `seekToEnd`, `writeData:`.

seekToEnd

```
- (unsigned)seekToEnd
```

Puts the file pointer at the end of the file referenced by the receiver and returns the number of bytes the file pointer has advanced from the start of the file. See also `position`, `seekToPosition:`.

seekToPosition:

```
- (unsigned)seekToPosition:(unsigned)position
```

Moves the file pointer to the specified `position` within the file referenced by the receiver and returns the number of bytes the file pointer has advanced from the start of the file. See also `position`, `readEntireFile`, `writeData:`.

setDelegate:

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

Sets the delegate object for the receiver. See also `delegate`.

synchronize

```
- (void)synchronize
```

Synchronizes the in-core state of the mapped file referenced by the receiver with the on-disk image. See also `mapFileRange:`.

truncateAtPosition:

```
- (void)truncateAtPosition:(unsigned)position
```

Truncates the file referenced by the receiver at `position` within the file. See also `writeData:`.

writeData:

- (void)writeData:(NSData *)data

Writes `data` to the file or device referenced by the receiver. See also `position`, `readEntireFile`, `seekToEnd`, `writeData:range:`.

writeData:range:

```
- (void)writeData:(NSData *)data range:(NSRange)range
```

Writes the `range` of bytes from `data` to the current position within the file referenced by the receiver. See also `writeData:`.

Methods implemented by the delegate

activity:posixFileDescriptor:

```
- activity:(NSPosixFileActivities)activity
    posixFileDescriptor:(NSPosixFileDescriptor *)descriptor
```

Invoked to inform the delegate that the `NSPosixFileDescriptor` identified by `descriptor` (a socket, device, pipe, or named pipe) is manifesting the condition identified by `activity`. This condition can indicate a readiness for reading or writing data, or can be an exception that is pending.

NSProcessInfo

Characteristic	Description
Inherits From:	NSObject
Conforms To:	NSObject (NSObject)
Declared In:	Foundation / NSProcessinfo.h

Class Description

The `NSProcessInfo` class provides methods to access process-wide information. An `NSProcessInfo` object can return such information as the arguments, environment, host name, or process name. The `processInfo` class method returns an `NSProcessInfo` object. For example, the following code creates an `NSProcessInfo` object, which then provides the name of the current process:

```
[[NSProcessInfo processInfo] processName];
```

Method Types

Activity	Class Method
Getting an NSProcessInfo object	+ processInfo
Returning process information	– arguments – environment – hostName – processName – globallyUniqueString
Specifying a process name	– setProcessName:

Class Methods

processInfo

```
+ (NSProcessInfo *)processInfo
```

Returns the `NSProcessInfo` object for the process. It is already initialized. An `NSProcessInfo` object is created the first time this method is invoked, and that same object is returned on each subsequent invocation.

Instance Methods

arguments

```
- (NSArray *)arguments
```

Returns the arguments as an array of `NSString`s from the command line.

environment

```
- (NSDictionary *)environment
```

Returns a dictionary of variables defined for the environment from which the process was launched.

globallyUniqueString

```
- (NSString *)globallyUniqueString
```

Returns a globally unique string to identify the process. This method uses the host name, process ID, and a timestamp (in that order) to ensure that the string returned will be globally unique.

hostName

```
- (NSString *)hostName
```

Returns the name of the host system.

processName

```
- (NSString *)processName
```

Returns the name of the process under which this program's user defaults domain is created, and is the name used in error messages. It does not uniquely identify the process.

setProcessName:

```
- (void)setProcessName:(NSString *)newName
```

Sets the name of the process to `newName`. Aspects of the environment like user defaults might depend on the process name, so be very careful if you change this. Setting the process name this way is not thread-safe.

NSProxy

Characteristic	Description
Inherits From:	none (NXProxy is a root class)
Conforms To:	NSObject
Declared In:	Foundation / NSProxy

Class Description

A proxy object stands in for another (real) object. Proxies can stand in for real objects, which should be descendants of `NSObject`, that can exist in another process, perhaps on another machine across a network.

To the application, the proxy behaves like the real object, though the real object may not be directly accessible, and in general, instance variables of remote objects are not accessible.

`NSProxy` class defines few methods because proxies respond to few messages directly. Instead, when a proxy receives a message it doesn't respond to, it encodes the message, including the arguments, in an `NSInvocation`, and invokes `forwardInvocation:`. Specialized subclasses then direct further processing, such as forwarding the message to a real object in the same or another process.

Methods defined in this class are methods that the `NSProxy` class responds to directly. Unless otherwise noted, none of these methods are forwarded to the proxy's correspondent.

Your application in general doesn't instantiate `NSProxy` objects—they're created as instances of specialized subclasses. Proxies are reference-counted so that only a single `NSProxy` per connection is instantiated for any real object.

Method Types

Activity	Class Method
Creating and destroying instances	+ alloc + allocWithZone: – dealloc
Identifying classes	+ class
Obtaining method information	– methodSignatureForSelector:
Describing objects	– description
Forwarding messages	– forwardInvocation:

Class Methods

alloc

```
+ (id)alloc
```

Returns a new, uninitialized instance of the receiving class.

allocWithZone:

```
+ (id)allocWithZone:(NSZone *)zone
```

Returns a new, uninitialized instance of the receiving class in `zone`.

class

```
+ (Class)class
```

Returns `self`. Since this is a class method, it returns the class object.

Instance Methods

dealloc

```
- (void)dealloc
```

Deallocates the memory occupied by the receiver.

description

```
- (NSString *)description
```

Prints the name of receiver's class and the hexadecimal value of its `id`. See also `description` (`NSObject`, `NSArray`, `NSDictionary`).

forwardInvocation:

```
- (void)forwardInvocation:(NSInvocation *)invocation
```

Implemented by subclasses to forward messages to other objects. The `NSProxy` implementation of this method raises an `NSInvalidArgumentException` exception. See also `NSInvocation`.

methodSignatureForSelector:

```
- (NSMethodSignature *)methodSignatureForSelector:(SEL)aSelector
```

Implemented by subclasses to return an object that contains a description of the `aSelector` method, or to return `nil` if the `aSelector` method can't be found. The `NSProxy` implementation of this method raises an `NSInvalidArgumentException` exception.

NSRecursiveLock

Characteristic	Description
Inherits From:	NSObject
Conforms To:	NSLocking NSObject (NSObject)
Declared In:	Foundation / NSLock.h

Class Description

`NSRecursiveLock` is used for locks that need to be reacquired by the same thread. An `NSRecursiveLock` locks a critical section of code such that a single thread can reacquire the lock multiple times without deadlocking, while preventing access by other threads. (Note that this implies that a recursive lock

will not protect a critical section from a signal handler interrupting the thread holding the lock.) Here is an example where a recursive lock functions properly, but where other lock types would deadlock:

```
// create the lock only once!
NSRecursiveLock  *theLock = [NSRecursiveLock new];
/* ...other code... */
[theLock lock];

/* ... possibly a long time of fussing with global data... */
[theLock lock]; /* possibly invoked in a subroutine */
[theLock unlock];

[theLock unlock];
```

The `NSConditionLock`, `NSLock`, and `NSRecursiveLock` classes all implement the `NSLocking` protocol with various features and performance characteristics; see the `NSConditionLock` and `NSLock` class descriptions for more information.

Method Types

Activity	Class Method
Acquiring a lock	– tryLock

Instance Methods

`tryLock`

`- (BOOL)tryLock`

Attempts to acquire a lock. Returns `YES` if successful and `NO` otherwise. This method can be called repeatedly to produce nested locks.

NSRunLoop

Characteristic	Description
Inherits From:	NSObject
Conforms To:	NSObject (NSObject)
Declared In:	Foundation / NSRunLoop.h

Class Description

The `NSRunLoop` class declares the programmatic interface to objects that manage input sources. An `NSRunLoop` object processes input for sources such as mouse and keyboard events from the window system, `NSTimer`s, POSIX file descriptors, and `NSConnection`s, based on a *mode* argument. A given `NSRunLoop` object processes input for input sources associated with a particular mode.

In general, your application won't need to either create or explicitly manage `NSRunLoop` objects. Each thread has an `NSRunLoop` object automatically created for it. The `NSApplication` object creates a default thread and therefore creates a default run loop.

Applications wanting to perform their own explicit run loop management should send the `currentRunLoop` message to the `NSRunLoop` class object to obtain the `NSRunLoop` object for the current thread, then invoke one of the methods, described below in "Running a run loop" method category, to obtain input.

Currently defined modes are:

Table 5-3 Run Loop and Reply Modes

Mode	Use
NSDefaultRunLoopMode	Use this mode to deal with input sources other than NSConnections. Defined in the Foundation / NSRunLoop.h header file.
NSConnectionReplyMode	Use this mode to indicate NSConnections waiting for replies. Defined in the Foundation / NSConnection.h header file.

Method Types

Activity	Class Method
Accessing the current run loop	+ currentRunLoop – currentMode – limitDateForMode:
Adding timers	– addTimer:forMode:
Running a run loop	– acceptInputForMode:beforeDate: – run – runMode:beforeDate: – runUntilDate:

Class Methods

currentRunLoop

```
+ (NSRunLoop *)currentRunLoop
```

Returns the NSRunLoop for the current thread.

Instance Methods

acceptInputForMode:beforeDate:

```
– (void)acceptInputForMode:(NSString *)mode
    beforeDate:(NSDate *)limitDate
```

Runs the run loop, accepting input from the input sources for the mode specified by mode until the time specified by limitDate.

addTimer:forMode:

```
– (void)addTimer:(NSTimer *)aTimer forMode:(NSString *)mode
```

Registers the timer aTimer with input filter mode. The run loop causes the timer to fire at its scheduled fire date. Note that timers are removed from modes if they supply nil as their fire date. See also NSTimer.

currentMode

```
- (NSString *)currentMode
```

Returns the current run loop mode.

limitDateForMode:

```
- (NSDate *)limitDateForMode:(NSString *)mode
```

Polls timers and platform-specific input managers for their limit date (if any). Timers will fire if appropriate. Returns `nil` if there are no input sources for this mode.

run

```
- (void)run
```

Runs the run loop in the default mode until there is nothing to do.

runMode:beforeDate:

```
- (BOOL)runMode:(NSString *)mode beforeDate:(NSDate *)limitDate
```

Runs the run loop, accepting input from filter `mode` until `limitDate` or until the earliest limit date for input sources in this mode. Returns `NO` without starting the run loop if there are no limit dates set for input sources (that is, there's nothing to do).

runUntilDate:

```
- (void)runUntilDate:(NSDate *)limitDate
```

Runs the run loop until `limitDate` or until there are no limit dates set for input sources (that is, there's nothing to do).

NSScanner

Characteristic	Description
Inherits From:	NSObject
Conforms To:	NSCopying NSObject (NSObject)
Declared In:	Foundation / NSScanner.h

Class Description

The `NSScanner` class declares the programmatic interface to an object that is capable of scanning `NSString` objects (strings of characters in the Unicode character encoding), converting the scanned strings to various numeric representations, or scanning characters from a character set.

Generally, you instantiate a scanner object by sending one of `scannerWithString:` or `localizedScannerWithString:` methods to the `NSScanner` class object. Either method returns a scanner object initialized with the string you pass in.

`NSScanner` provides methods of configuring the behavior of the scan. `setCaseSensitive:` specifies whether the scanner will treat upper case and lower case letters as distinct. `setCharactersToBeSkipped:` determines the set of characters that will be skipped while scanning. The preset set of characters to skip are whitespace and newline characters. `setLocale:` specifies the locale to be used while scanning strings**.** `setScanLocation:` sets the index in the string object at that scanning will commence. Using this method, you can repeatedly scan portions of a string.

Scanning is performed using any of the `scan...` methods listed under “Scanning a string” method category.

Note – Floating-point numbers are assumed to be IEEE compliant.

Method Types

Activity	Class Method
Creating an NSScanner	+ localizedScannerWithString: + scannerWithString: – initWithString:
Getting an NSScanner's string	– string
Configuring an NSScanner	– caseSensitive – charactersToBeSkipped – locale – scanLocation – setCaseSensitive: – setCharactersToBeSkipped: – setLocale: – setScanLocation:
Scanning a string	– scanCharactersFromSet:intoString: – scanDouble: – scanFloat: – scanHexInt: – scanInt: – scanLongLong: – scanString:intoString: – scanUpToCharactersFromSet:intoString: – scanUpToString:intoString: – isAtEnd

Class Methods

`localizedScannerWithString:`

```
+ (id)localizedScannerWithString:(NSString *)aString
```

Creates and returns a scanner that scans `aString`. Invokes `initWithString:` and sets the locale to the user's default locale.

`scannerWithString:`

```
+ (id)scannerWithString:(NSString *)aString
```

Creates and returns a scanner that scans `aString`.

Instance Methods

caseSensitive

```
- (BOOL)caseSensitive
```

Returns YES if the scanner distinguishes case, and NO otherwise. Scanners are by default *not* case sensitive.

charactersToBeSkipped

```
- (NSCharacterSet *)charactersToBeSkipped
```

Returns a character set object containing those characters that the scanner ignores when looking for an element. The default set is the whitespace and newline character set.

initWithString:

```
- (id)initWithString:(NSString *)aString
```

Initializes the receiver, a newly allocated scanner, to scan `aString`. Returns `self`.

isAtEnd

```
- (BOOL)isAtEnd
```

Returns YES if the scanner has exhausted all characters in its string; returns NO if there are characters left to scan.

locale

```
- (NSDictionary *)locale
```

Returns a dictionary object containing locale information. Returns `nil` if the locale dictionary has not been set.

scanCharactersFromSet:intoString:

```
- (BOOL)scanCharactersFromSet:(NSCharacterSet *)aSet
    intoString:(NSString **)value
```

Scans the string as long as characters from `aSet` are encountered, accumulating characters into an optional string that's returned by reference in `value`. If any characters are scanned, returns `YES`; otherwise returns `NO`.

scanDouble:

```
- (BOOL)scanDouble:(double *)value
```

Scans a `double` into `value` if possible. Returns `YES` if a valid floating-point expression was scanned; returns `NO` otherwise. `HUGE_VAL` or `–HUGE_VAL` is put in `value` on `overflow`; 0.0 on underflow. Returns `YES` in overflow and underflow cases.

scanFloat:

```
- (BOOL)scanFloat:(float *)value
```

Scans a `float` into `value` if possible. Returns `YES` if a valid floating-point expression was scanned; returns `NO` otherwise. `HUGE_VAL` or `–HUGE_VAL` is put in `value` on overflow; 0.0 on underflow. Returns `YES` in overflow and underflow cases.

scanHexInt:

```
- (BOOL)scanHexInt:(unsigned *)value
```

Scans a hexadecimal integer into `value`, if possible. Returns `YES` if successful; returns `NO` otherwise.

scanInt:

```
- (BOOL)scanInt:(int *)value
```

Scans an `int` into `value` if possible. Returns `YES` if a valid integer expression was scanned; returns `NO` otherwise. `INT_MAX` or `INT_MIN` is put in `value` on overflow. Returns `YES` in overflow cases.

scanLocation

```
- (unsigned)scanLocation
```

Returns the character index at which the scanner will begin its next scanning operation.

scanLongLong:

```
- (BOOL)scanLongLong:(long long *)value
```

Scans a `long long int` into `value` if possible. Returns `YES` if a valid integer expression was scanned; returns `NO` otherwise. `LONG_LONG_MAX` or `LONG_LONG_MIN` is put in `value` on overflow. Returns `YES` in overflow cases.

scanString:intoString:

```
- (BOOL)scanString:(NSString *)aString
    intoString:(NSString **)value
```

Scans for `aString`, and if a match is found returns by reference in the optional `value` argument a string object equal to it. If `aString` matches the characters at the scan location, this method returns `YES`; otherwise returns `NO`.

scanUpToCharactersFromSet:intoString:

```
- (BOOL)scanUpToCharactersFromSet:(NSCharacterSet *)aSet
    intoString:(NSString **)value
```

Scans the string until a character from `aSet` is encountered, accumulating characters encountered into a string that's returned by reference in the optional `value` argument. If any characters are scanned, returns `YES`; otherwise returns `NO`.

scanUpToString:intoString:

```
- (BOOL)scanUpToString:(NSString *)aString
    intoString:(NSString **)value
```

Scans the string until `aString` is encountered, accumulating characters encountered into a string that's returned by reference in the optional `value` argument. If any characters are scanned, returns `YES`, otherwise returns `NO`.

setCaseSensitive:

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

If `flag` is `YES`, the scanner considers case when scanning characters. If `flag` is `NO`, it ignores case distinctions. `NSScanner`s are by default *not* case sensitive.

`setCharactersToBeSkipped:`

```
- (void)setCharactersToBeSkipped:(NSCharacterSet *)aSet
```

Sets the scanner to ignore characters from `aSet` when scanning its string.

`setLocale:`

```
- (void)setLocale:(NSDictionary *)localeDictionary
```

Sets the receiver's dictionary object containing locale information.

`setScanLocation:`

```
- (void)setScanLocation:(unsigned int)anIndex
```

Sets the location at which the next scan will begin to `anIndex`.

`string`

```
- (NSString *)string
```

Returns the string object that the scanner was created with.

NSSerializer

Characteristic	Description
Inherits From:	NSObject
Conforms To:	NSObject (NSObject)
Declared In:	Foundation / NSSerialization.h

Class Description

The `NSSerializer` class provides a mechanism for creating an abstract representation of a property list. (In OpenStep, property lists are defined to be—and to contain—objects of these classes: `NSDictionary`, `NSArray`, `NSString`, `NSData`). The `NSSerializer` class stores this representation in an

NSData object in an architecture-independent format, so that property lists can be used with distributed applications. NSSerializer's companion class NSDeserializer declares methods that take the abstract representation and recreate the property list in memory.

In contrast to archiving (see the NSArchiver class specification), the serialization process preserves only structural information, not class information. If a property list is serialized and then deserialized, the objects in the resulting property list might not be of the same class as the objects in the original property list. However, the structure and interrelationships of the data in the resulting property list are identical to that in the original, with one possible exception.

The exception is that when an object graph is serialized, the mutability of the container objects (NSDictionary and NSArray objects) is preserved only down to the highest node in the graph that has an immutable container. Thus, if an NSArray contains an NSMutableDictionary, the serialized version of this object graph would not preserve the mutability of the dictionary or any of the mutable objects it contained. Since serialization doesn't preserve class information or—in some cases—mutability, coding (as implemented by NSCoder and NSArchiver) is the preferred way to make object graphs persistent.

The NSSerializer class object provides the interface to the serialization process; you don't create instances of NSSerializer. You might subclass NSSerializer to modify the representation it creates, for example, to encrypt the data or add authentication information.

Other types of data besides property lists can be serialized using methods declared by the NSData and NSMutableData classes (see serializeDataAt:ofObjCType:context: and deserializeDataAt:ofObjCType:atCursor:context:), allowing these types to be represented in an architecture-independent format. Furthermore, the NSObjCTypeSerializationCallBack protocol allows you to serialize and deserialize objects that aren't property lists.

Method Types

Activity	Class Method
Serialization of property lists	+ serializePropertyList: + serializePropertyList:intoData:

Class Methods

serializePropertyList:

```
+ (NSData *)serializePropertyList:(id)aPropertyList
```

Creates a data object, serializes `aPropertyList` into it, and returns the data object. `aPropertyList` must be a kind of `NSData`, `NSString`, `NSArray`, or `NSDictionary`.

serializePropertyList:intoData:

```
+ (void)serializePropertyList:(id)aPropertyList
    intoData:(NSMutableData *)mdata
```

Serializes the property list `aPropertyList` in the mutable data object `mdata`. `aPropertyList` must be a kind of `NSData`, `NSString`, `NSArray`, or `NSDictionary`.

NSSet

Characteristic	Description
Inherits From:	NSObject
Conforms To:	NSCoding, NSCopying, NSMutableCopying NSObject (NSObject)
Declared In:	Foundation / NSSet.h

Class Description

The `NSSet` class declares the programmatic interface to an object that manages an immutable set of objects. `NSSet` provides support for the mathematical concept of a *set*. A set, both in its mathematical sense and in the OpenStep

implementation of NSSet, is an *unordered* collection of distinct elements. OpenStep provides the NSMutableSet class for sets whose contents may be altered, and also provides the NSCountedSet class for sets that can contain multiple instances of the same element.

Use set objects as an alternative to array objects when the order of elements is not important, but performance in testing whether an object is contained in the set *is* a consideration—while arrays are ordered, testing for membership is slower than with sets. For example, the NSSet method containsObject: operates in O(1) time when applied to a set, while containsObject: operates in O(N) time when applied to an array.

Objects in a set must respond to hash and isEqual: methods. See the NSObject protocol for details on hash and isEqual:.

Generally, you instantiate an NSSet object by sending one of the set… methods to the NSSet class object. These methods return an NSSet object containing the elements (if any) you pass in as arguments. The set method is a "convenience" method to create an empty set. Newly created instances of NSSet created by invoking the set method can be populated with objects using any of the init… methods. initWithObjects: is the designated initializer for the NSSet class. Objects added to the set are not copied; rather, each object receives a retain message before it is added to the set.

NSSet provides methods for querying the elements of the set. allObjects returns an array containing all objects in the set. anyObject returns some object in the set. count returns the number of objects currently in the set. member: returns the object in the set that is equal to a specified object. Additionally, the intersectsSet: tests for set intersection, isEqualToSet: tests for set equality, and isSubsetOfSet: tests for one set being a subset of the specified set object.

The objectEnumerator method provides for traversing elements of the set one by one. NSSet's makeObjectsPerform: and makeObjectsPerform:withObject: methods provide for sending messages to individual objects in the set.

Exceptions

NSSet implements the encodeWithCoder: method, which raises NSInternalInconsistencyException if the number of objects enumerated for encoding turns out to be unequal to the number of objects in the set.

Method Types

Activity	Class Method
Allocating and initializing a set	+ allocWithZone: + set + setWithArray: + setWithObject: + setWithObjects: – initWithArray: – initWithObjects: – initWithObjects:count: – initWithSet: – initWithSet:copyItems:
Querying the set	– allObjects – anyObject – containsObject: – count – member: – objectEnumerator
Sending messages to elements of the set	– makeObjectsPerform: – makeObjectsPerform:withObject:
Comparing sets	– intersectsSet: – isEqualToSet: – isSubsetOfSet:
Creating a string description of the set	– description – descriptionWithLocale:

Class Methods

allocWithZone:

```
+ (id)allocWithZone:(NSZone *)zone
```

Creates and returns an uninitialized set object in `zone`.

set

```
+ (id)set
```

Creates and returns an empty set object.

setWithArray:

```
+ (id)setWithArray:(NSArray *)array
```

Creates and returns a set object containing the objects in `array`.

setWithObject:

```
+ (id)setWithObject:(id)anObject
```

Creates and returns a set object containing the single element `anObject`.

setWithObjects:

```
+ (id)setWithObjects:(id)firstObj,...
```

Creates and returns a set object containing the objects in the argument list. The object list is comma-separated and ends with `nil`.

Instance Methods

allObjects

```
- (NSArray *)allObjects
```

Returns an array containing all the objects in the set.

anyObject

```
- (id)anyObject
```

Returns some object in the set, or `nil` if the set is empty.

containsObject:

```
- (BOOL)containsObject:(id)anObject
```

Returns `YES` if `anObject` is present in the set.

count

```
- (unsigned int)count
```

Returns the number of objects currently in the set.

description

```
- (NSString *)description
```

Returns a string object that describes the contents of the receiver. See also `description` (`NSArray`, `NSDictionary`, `NSObject`).

descriptionWithLocale:

```
- (NSString *)descriptionWithLocale:
    (NSDictionary *)localeDictionary
```

Returns a string representation of the `NSSet` object, including the keys and values that represent the locale data from `localeDictionary`.

initWithArray:

```
- (id)initWithArray:(NSArray *)array
```

Initializes a newly allocated set object by placing in it the objects contained in `array`.

initWithObjects:

```
- (id)initWithObjects:(id)firstObj,...
```

Initializes a newly allocated set object by placing in it the objects in the argument list. The object list is comma-separated and ends with `nil`.

initWithObjects:count:

```
- (id)initWithObjects:(id *)objects count:(unsigned int)count
```

Initializes a newly allocated set object by placing in it `count` objects from the `objects` array.

initWithSet:

```
- (id)initWithSet:(NSSet *)anotherSet
```

Initializes a newly allocated set object by placing in it the objects contained in `anotherSet`.

initWithSet:copyItems:

```
- (id)initWithSet:(NSSet *)set copyItems:(BOOL)flag
```

Initializes a newly allocated set object by placing in it the objects contained in `anotherSet` (or immutable copies of them, if `flag` is YES).

intersectsSet:

```
- (BOOL)intersectsSet:(NSSet *)otherSet
```

Returns YES if there is any object in the receiving set that's equal to an object in `otherSet`.

isEqualToSet:

```
- (BOOL)isEqualToSet:(NSSet *)otherSet
```

Returns YES if every object in the receiving set is equal to an object in `otherSet`, and the two sets contain the same number of objects.

isSubsetOfSet:

```
- (BOOL)isSubsetOfSet:(NSSet *)otherSet
```

Returns YES if every object in the receiving set is equal to an object in `otherSet`, and the receiving set contains no more objects than `otherSet` does.

makeObjectsPerform:

```
- (void)makeObjectsPerform:(SEL)aSelector
```

Sends an `aSelector` message to each object in the set.

makeObjectsPerform:withObject:

```
- (void)makeObjectsPerform:(SEL)aSelector withObject:(id)anObject
```

Sends an `aSelector` message to each object in the set, with `anObject` as an argument.

member:

```
- (id)member:(id)anObject
```

Return the object in the set that is equal to `anObject`, or `nil` if none is equal.

objectEnumerator

```
- (NSEnumerator *)objectEnumerator
```

Returns an enumerator object that lets you access each object in the set.

NSString

Characteristic	Description
Inherits From:	NSObject
Conforms To:	NSCoding, NSCopying, NSMutableCopying NSObject (NSObject)
Declared In:	Foundation / NSString.h Foundation / NSPathUtilities.h Foundation / NSUtilities.h

Class Description

`NSString` declares the programmatic interface for objects that create and manage immutable character strings in a *representation-independent* format.

`NSString` and `NSMutableString` are abstract classes for string manipulation. `NSString` provides methods for read-only access, while `NSMutableString` allows for changing the contents of the string. `NSString` and `NSMutableString` provide factory methods that return autoreleased instances of unspecified subclasses of strings.

While the actual representation of character strings stored in `NSString` and `NSMutableString` is independent of any particular implementation, you can in general think of the contents of `NSString` and `NSMutableString` objects as being Unicode characters defined by the `unichar` data type. Methods that use the terms “character”, “range”, and “length”, refer to strings of `unichar`s and ranges and lengths of `unichar` strings. This is important because conversion between `unichar`s and other character encodings is not necessarily one-to-one. For instance, an ISO Latin1 encoded string of a given length might contain fewer or more characters when encoded as `unichar`s. Another important point is that `unichar`s don't necessarily correspond one-to-one with what is normally thought of as “letters” in a string; if you need to go through a string in terms of ”letters”, use `rangeOfComposedCharacterSequenceAtIndex:`.

Methods that take “CString” arguments deal with the default eight-bit encoding of the environment, which could be, for instance, EUC or ISOLatin1. You can also explicitly convert to and from any encoding by using methods such as `initWithData:usingEncoding:` and `dataUsingEncoding:`.

Constant `NSString`s can be created with the `@"..."` option. Such strings should contain only ASCII characters, and nothing more.

Strings are provided with generic coding behavior when used for storage or distribution. This behavior is to copy the contents and provide a generic `NSString` implementation, losing class but preserving mutability.

In general, you instantiate `NSString` objects sending one of the `stringWith…` methods or the `localizedStringWithFormat:` method to the `NSString` class object. For `NSString` objects that were allocated “manually”, use any of the `initWith…` methods to initialize the contents of the string object.

The primitive methods to `NSString` are `length` and `characterAtIndex:`.

UNIX-style file system path names can be manipulated using the collection of `stringBy…` methods described under “Manipulating file system paths” method category in the following table.

Method Types

Activity	Class Method
Creating temporary strings	+ localizedStringWithFormat: + stringWithCString: + stringWithCString:length: + stringWithCharacters:length: + stringWithContentsOfFile: + stringWithFormat:
Initializing newly allocated strings	– init – initWithCString: – initWithCString:length: – initWithCStringNoCopy:length:freeWhenDone: – initWithCharacters:length: – initWithCharactersNoCopy:length:freeWhenDone: – initWithContentsOfFile: – initWithData:encoding: – initWithFormat: – initWithFormat:arguments: – initWithFormat:locale:arguments: – initWithString:
Getting a string's length	– length
Accessing characters	– characterAtIndex: – getCharacters: – getCharacters:range:
Combining strings	– stringByAppendingFormat: – stringByAppendingString:
Dividing strings into substrings	– componentsSeparatedByString: – substringFromIndex: – substringWithRange: – substringToIndex:
Finding ranges of characters and substrings	– rangeOfCharacterFromSet: – rangeOfCharacterFromSet:options: – rangeOfCharacterFromSet:options:range: – rangeOfString: – rangeOfString:options: – rangeOfString:options:range:
Determining composed character sequences	– rangeOfComposedCharacterSequenceAtIndex:

Activity	Class Method
Indentifying and comparing strings	– caseInsensitiveCompare: – compare: – compare:options: – compare:options:range: – hasPrefix: – hasSuffix: – hash – isEqualToString:
Storing the string	– description – writeToFile:atomically:
Getting a shared prefix	– commonPrefixWithString:options:
Changing case	– capitalizedString – lowercaseString – uppercaseString
Getting C strings	– cString – cStringLength – getCString: – getCString:maxLength: – getCString:maxLength:range:remainingRange: – lossyCString
Getting numeric values	– doubleValue – floatValue – intValue

Activity	Class Method
Working with encodings	+ availableStringEncodings + defaultCStringEncoding + localizedNameOfStringEncoding: – canBeConvertedToEncoding: – dataUsingEncoding: – dataUsingEncoding:allowLossyConversion: – fastestEncoding – smallestEncoding
Converting string contents into a property list	– propertyList – propertyListFromStringsFileFormat
Manipulating file system paths	– completePathIntoString:caseSensitive: matchesIntoArray:filterTypes: – fileSystemRepresentation – getFileSystemRepresentation:maxLength: – lastPathComponent – pathExtension – stringByAbbreviatingWithTildeInPath – stringByAppendingPathComponent: – stringByAppendingPathExtension: – stringByDeletingLastPathComponent – stringByDeletingPathExtension – stringByExpandingTildeInPath – stringByResolvingSymlinksInPath – stringByStandardizingPath – stringsByAppendingPaths:

Class Methods

`availableStringEncodings`

```
+ (NSStringEncoding *)availableStringEncodings
```

Returns a null terminated array of available string encodings. See the “String” section of the Foundation Kit’s “Types and Constants” chapter for a list of available string encodings. See also `defaultCStringEncoding`, `localizedNameOfStringEncoding:`, `canBeConvertedToEncoding:`, `dataUsingEncoding:`, `dataUsingEncoding:allowLossyConversion:`, `fastestEncoding`, `smallestEncoding`.

defaultCStringEncoding

```
+ (NSStringEncoding)defaultCStringEncoding
```

Returns the C string encoding assumed for any method accepting a C string as an argument. See the "String" section of the Foundation Kit's "Types and Constants" chapter for a list of available string encodings. See also `availableStringEncodings`.

localizedNameOfStringEncoding:

```
+(NSString *)localizedNameOfStringEncoding:
 (NSStringEncoding)encoding
```

Returns the localized name of the string encoding specified by `encoding`. See the "String" section of the Foundation Kit's "Types and Constants" chapter for a list of available string encodings. See also `availableStringEncodings`.

localizedStringWithFormat:

```
+ (id)localizedStringWithFormat:(NSString *)format,...
```

Returns a string created by using `format` as a `printf()` style format string, and the following arguments as values to be substituted into the format string. The user's default locale is used for format information. See also `stringWithFormat:`, `availableStringEncodings`.

stringWithCString:

```
+ (id)stringWithCString:(const char *)byteString
```

Returns a string containing the characters in `byteString`, which must be null-terminated. `byteString` should contain characters in the default C string encoding. This method sends the message `stringWithCString:byteString length:strlen(byteString)`. See also `stringWithCString:length:`, `availableStringEncodings`.

stringWithCString:length:

```
+ (id)stringWithCString:(const char *)byteString
    length:(unsigned int)length
```

Returns a string containing characters from `byteString`. `byteString` should contain characters in the default C string encoding. `length` bytes are copied into the string, regardless of whether a null byte exists in `byteString`. Raises `NSInvalidArgumentException` if `byteString` is `NULL`. See also `stringWithCString:`, `availableStringEncodings`.

`stringWithCharacters:length:`

```
+ (id)stringWithCharacters:(const unichar *)chars
    length:(unsigned int)length
```

Returns a string containing `chars`. `length` characters are copied into the string, regardless of whether a null character exists in `chars`. See also `availableStringEncodings`.

`stringWithContentsOfFile:`

```
+ (id)stringWithContentsOfFile:(NSString *)path
```

Returns a string containing the contents of the file specified by `path`, or `nil` if unsuccessful. This method attempts to determine the encoding for the file. The string is assumed to be in Unicode encoding, but if the encoding is determined not to be Unicode, the default C-string encoding is used instead. See also `availableStringEncodings`.

`stringWithFormat:`

```
+ (id)stringWithFormat:(NSString *)format,...
```

Returns a string created by using `format` as a `printf()` style format string (for example `%s`), and the following arguments as values to be substituted into the format string. Note that the `p` and `n` format specifiers are not supported; valid format specifiers are `c s O o X x D d U u I i E e G g f I`. In addition, `@` can be used to specify arbitrary objects; in this case, any arguments to the object format specifier (for example field width and precision) are ignored. Solaris `sprintf`-style argument reordering (`$`) is also supported. See also `localizedStringWithFormat:`, `availableStringEncodings`.

Instance Methods

canBeConvertedToEncoding:

```
- (BOOL)canBeConvertedToEncoding:(NSStringEncoding)encoding
```

Returns YES if the receiver can be converted to encoding without loss of information, and NO otherwise. See also availableStringEncodings.

capitalizedString

```
- (NSString *)capitalizedString
```

Returns a string with the first character of each word changed to its corresponding uppercase value. See also lowercaseString, uppercaseString.

caseInsensitiveCompare:

```
- (NSComparisonResult)caseInsensitiveCompare:(NSString *)aString
```

Invokes compare:options: with the option NSCaseInsensitiveSearch. See also compare:options:, compare:options:range:.

characterAtIndex:

```
- (unichar)characterAtIndex:(unsigned int)index
```

Returns the character at the array position given by index. This method raises an NSStringBoundsError exception if index lies beyond the end of the string. See also getCharacters:.

commonPrefixWithString:options:

```
- (NSString *)commonPrefixWithString:(NSString *)aString
    options:(unsigned int)mask
```

Returns the substring of the receiver containing characters that the receiver and aString have in common. mask can be any combination (using the C bitwise OR operator |) of NSCaseInsensitiveSearch and NSLiteralSearch (character-by-character search). See the String section of the Foundation Kit's Types and Constants chapter for a list of search masks.

compare:

```
- (NSComparisonResult)compare:(NSString *)aString
```

Invokes `compare:options:` with no options. See the "String section" of the Foundation Kit's "Types and Constants" chapter for a list of search masks. See also `compare:options:`, `compare:options:range:`, `caseInsensitiveCompare:`, `hasPrefix:`, `hasSuffix:`, `hash`, `isEqualToString:`.

compare:options:

```
- (NSComparisonResult)compare:(NSString *)aString
    options:(unsigned int)mask
```

Invokes `compare:options:range:` with `mask` as the options and the receiver's full extent as the range. See also `compare:`.

compare:options:range:

```
- (NSComparisonResult)compare:(NSString *)aString
    options:(unsigned int)mask range:(NSRange)aRange
```

Compares `aString` to the receiver and returns their lexical ordering. The comparison is restricted to `aRange` and uses `mask` options, which may be `NSCaseInsensitiveSearch` and `NSLiteralSearch`. One of the following values is returned:

- `NSOrderedAscending`
- `NSOrderedSame`
- `NSOrderedDescending`

See also `compare:`.

completePathIntoString:caseSensitive: matchesIntoArray:filterTypes:

```
- (unsigned int)completePathIntoString:(NSString **)outputName
    caseSensitive:(BOOL)flag
    matchesIntoArray:(NSArray **)outputArray
    filterTypes:(NSArray *)filterTypes
```

Regards the receiver as containing a partial filename and returns in `outputName` the longest matching path name. Case is considered if `flag` is `YES`. If `outputArray` is given, all matching file names are returned in `outputArray`. If `filterTypes` is provided, this method considers only those paths that match one of the types. Returns 0 if no matches are found; otherwise, the return value is positive. See also `lastPathComponent`, `pathExtension`, `stringByAbbreviatingWithTildeInPath`.

componentsSeparatedByString:

```
- (NSArray *)componentsSeparatedByString:(NSString *)separator
```

Finds the substrings in the receiver that are delimited by `separator` and returns them as the elements of an `NSArray`. The strings in the array appear in the order they appeared in the receiver. See also `substringFromIndex:`.

cString

```
- (const char *)cString
```

Returns a representation of the receiver as a C string in the default C-string encoding. See also `cStringLength`, `getCString:`.

cStringLength

```
- (unsigned int)cStringLength
```

Returns the length, in bytes, of the receiver's C string representation. See also `cString`.

dataUsingEncoding:

```
- (NSData *)dataUsingEncoding:(NSStringEncoding)encoding
```

Invokes `dataUsingEncoding:allowLossyConversion:` with `NO` as the argument to allow lossy conversion. See also `availableStringEncodings`.

dataUsingEncoding:allowLossyConversion:

```
- (NSData *)dataUsingEncoding:(NSStringEncoding)encoding
    allowLossyConversion:(BOOL)flag
```

Returns an NSData object containing a representation of the receiver in encoding. If flag is NO, and the receiver can't be converted without losing some information (such as accents or case), this method returns nil. If flag is YES and the receiver can't be converted without losing some information, some characters may be removed or altered in the conversion. See also dataUsingEncoding:, availableStringEncodings.

description

```
- (NSString *)description
```

Returns the string (self). See also writeToFile:atomically:.

doubleValue

```
- (double)doubleValue
```

Returns the double-precision floating-point value of the receiver's text. Whitespace at the beginning of the string is skipped. If the receiver begins with a valid text representation of a floating-point number, that number's value is returned, otherwise 0.0 is returned. HUGE_VAL or –HUGE_VAL is returned on overflow. 0.0 is returned on underflow. Characters following the number are ignored. See also floatValue, intValue.

fastestEncoding

```
- (NSStringEncoding)fastestEncoding
```

Encoding in which this string can be expressed with lossless conversion most quickly. See also smallestEncoding, availableStringEncodings.

floatValue

```
- (float)floatValue
```

Returns the floating-point value of the receiver's text. Whitespace at the beginning of the string is skipped. If the receiver begins with a valid text representation of a floating-point number, that number's value is returned, otherwise 0.0 is returned. HUGE_VAL or –HUGE_VAL is returned on overflow. 0.0 is returned on underflow. Characters following the number are ignored. See also doubleValue, intValue.

fileSystemRepresentation

```
- (const char *)fileSystemRepresentation
```

Returns a file system specific representation of the receiver, as described for `getFileSystemRepresentation:maxLength:`. The returned C string will be automatically freed just as a returned object would be released; your code should copy the representation or use `getFileSystemRepresentation:maxLength:` if it needs to store the representation outside of the autorelease context in which the representation is created. See also `getFileSystemRepresentation:maxLength:`.

getCharacters:

```
– (void)getCharacters:(unichar *)buffer
```

Invokes `getCharacters:range:` with the provided `buffer` and the entire extent of the receiver as the range. See also `characterAtIndex:`.

getCharacters:range:

```
– (void)getCharacters:(unichar *)buffer range:(NSRange)aRange
```

Copies characters from `aRange` in the receiver into `buffer`, which must be large enough to contain them. This method does *not* add a null character. This method raises an `NSStringBoundsError` exception if any part of `aRange` lies beyond the end of the string. See also `getCharacters:`, `characterAtIndex:`.

getCString:

```
– (void)getCString:(char *)buffer
```

Invokes `getCString:maxLength:range:remainingRange:` with `NSMaximumStringLength` as the maximum length, the receiver's entire extent as the range, and `NULL` for the remaining range. `buffer` must be large enough to contain the resulting C string plus a terminating null characte, which this method adds. See also `cString`, `cStringLength`, `getCString:maxLength:`, `getCString:maxLength:range:remainingRange:`.

getCString:maxLength:

```
- (void)getCString:(char *)buffer maxLength:(unsigned int)maxLength
```

Invokes `getCString:maxLength:range:remainingRange:` with `maxLength` as the maximum length, the receiver's entire extent as the range, and `NULL` for the remaining range. `buffer` must be large enough to contain the resulting C string plus a terminating null character (which this method adds). See also `getCString:`.

getCString:maxLength:range:remainingRange:

```
- (void)getCString:(char *)buffer maxLength:(unsigned int)maxLength
    range:(NSRange)aRange remainingRange:(NSRange *)leftoverRange
```

Copies the receiver's characters, in the default C-string encoding, as bytes into `buffer`. `buffer` must be large enough to contain `maxLength` bytes plus a terminating null character which this method adds. Characters are copied from `aRange`; if not all characters can be copied, the range of those not copied is put into `leftoverRange`. This method raises an `NSStringBoundsError` exception if any part of `aRange` lies beyond the end of the string. See also `getCString:`.

getFileSystemRepresentation:maxLength:

```
- (BOOL)getFileSystemRepresentation:(char *)c maxLength:(unsigned)m
```

Interprets the receiver as a system-independent path, filling buffer with a C string in a format and encoding suitable for use with file system calls. This is done by replacing the abstract path and extension separator characters ('/' and '.' respectively) with their equivalents for the operating system. For example, on Microsoft Windows 95 the receiver "C:/Working/Sample.tiff" is returned as the C string "C:\Working\Sample.tiff". Returns NO if the receiver cannot be converted to a C string or if it is an empty string object. See also `fileSystemRepresentation`.

hasPrefix:

```
- (BOOL)hasPrefix:(NSString *)aString
```

Returns `YES` if `aString` matches the beginning characters of the receiver, and returns `NO` otherwise. See also `hasSuffix:`, `compare:`.

hasSuffix:

```
- (BOOL)hasSuffix:(NSString *)aString
```

Returns YES if aString matches the ending characters of the receiver, and returns NO otherwise. See also hasPrefix:, compare:.

hash

```
- (unsigned int)hash
```

Returns an unsigned integer that can be used as a table address in a hash table structure. If two string objects are equal, they must have the same hash value. See also isEqualToString:, compare:.

init

```
- (id)init
```

Initializes the receiver, a newly allocated NSString, to contain no characters. This is the only initialization method that a subclass of NSString should invoke. See also initWithCString:.

initWithCString:

```
- (id)initWithCString:(const char *)byteString
```

Initializes the receiver, a newly allocated NSString, by converting the one-byte characters in byteString into Unicode characters. byteString must be a null-terminated C string in the default C string encoding. See also init, initWithCString:length:, initWithCharacters:length:, initWithString:.

initWithCString:length:

```
- (id)initWithCString:(const char *)byteString
    length:(unsigned int)length
```

Initializes the receiver, a newly allocated NSString, by converting length one-byte characters in byteString into Unicode characters. This method doesn't stop at a null byte. See also initWithCStringNoCopy:length:freeWhenDone:, initWithCString:.

initWithCStringNoCopy:length:freeWhenDone:

```
- (id)initWithCStringNoCopy:(char *)byteString
    length:(unsigned int)length
    freeWhenDone:(BOOL)flag
```

Initializes the receiver, a newly allocated NSString, by converting length one-byte characters in byteString into Unicode characters. This method doesn't stop at a null byte. The receiver becomes the owner of byteString; if flag is YES it will free the memory when it no longer needs it, but if flag is NO it won't. See also initWithCString:, initWithCString:length:.

initWithCharacters:length:

```
- (id)initWithCharacters:(const unichar *)chars
    length:(unsigned int)length
```

Initializes the receiver, a newly allocated NSString, by copying length characters from chars. This method doesn't stop at a null character. See also initWithCharactersNoCopy:length:freeWhenDone:, initWithCString:.

initWithCharactersNoCopy:length:freeWhenDone:

```
- (id)initWithCharactersNoCopy:(unichar *)chars
    length:(unsigned int)length freeWhenDone:(BOOL)flag
```

Initializes the receiver, a newly allocated NSString, to contain length characters from chars. This method doesn't stop at a null character. The receiver becomes the owner of chars; if flag is YES the receiver will free the memory when it no longer needs them, but if flag is NO it won't. Note that the NO case could be dangerous if used with memory that could be freed. The NO flag should be used only when the provided backing store is permanent. See also initWithCharacters:length:, initWithCString:.

initWithContentsOfFile:

```
- (id)initWithContentsOfFile:(NSString *)path
```

Initializes the receiver, a newly allocated NSString, by reading characters from the file whose name is given by path. This method attempts to determine the encoding for the file. The string is assumed to be in Unicode encoding, but

if the encoding is determined not to be Unicode, the default C string encoding is used instead. See also `writeToFile:atomically:`, `initWithCString:`, `initWithString:`, `initWithFormat:`, `initWithData:encoding:`.

initWithData:encoding:

```
- (id)initWithData:(NSData *)data
    encoding:(NSStringEncoding)encoding
```

Initializes the receiver, a newly allocated `NSString`, by converting the bytes in `data` into Unicode characters. `data` must be an `NSData` object containing bytes in `encoding` and in the default "plain text" format for that encoding. See also `initWithCString:`, `initWithContentsOfFile:`, `initWithCharacters:length:`, `initWithFormat:`, `initWithString:`.

initWithFormat:

```
- (id)initWithFormat:(NSString *)format,...
```

Initializes the receiver, a newly allocated `NSString`, by constructing a string from `format` and following string objects in the manner of `printf()`. See the `stringWithFormat:` description for a list of valid format specifiers. See also `initWithFormat:arguments:`, `initWithFormat:locale:`, `initWithFormat:locale:arguments:`, `initWithCString:`.

initWithFormat:arguments:

```
- (id)initWithFormat:(NSString *)format arguments:(va_list)argList
```

Initializes the receiver, a newly allocated `NSString`, by constructing a string from `format` and `argList` in the manner of `vprintf()`. See also `initWithFormat:`.

initWithFormat:locale:

```
- (id)initWithFormat:(NSString *)format
    locale:(NSDictionary *)dictionary,...
```

Initializes the receiver, a newly allocated `NSString`, by constructing a string from `format` and the formatting information in the dictionary in the manner of `printf()`. See also `initWithFormat:`.

initWithFormat:locale:arguments:

```
- (id)initWithFormat:(NSString *)format
    locale:(NSDictionary *)dictionary
    arguments:(va_list)argList
```

Initializes the receiver, a newly allocated NSString, by constructing a string from format and format information in dictionary and argList in the manner of vprintf(). See also initWithFormat:.

initWithString:

```
- (id)initWithString:(NSString *)string
```

Initializes the receiver, a newly allocated NSString, by copying the characters from string. See also initWithCString:, initWithFormat:, initWithData:encoding:.

intValue

```
- (int)intValue
```

Returns the integer value of the receiver's text. White space at the beginning of the string is skipped. If the receiver begins with a valid representation of an integer, that number's value is returned; otherwise 0 is returned. INT_MAX or INT_MIN is returned on overflow. Characters following the number are ignored. See also doubleValue, floatValue.

isEqualToString:

```
- (BOOL)isEqualToString:(NSString *)aString
```

Returns YES if aString is equivalent to the receiver if they have the same id or if they compare as NSOrderedSame. Returns NO otherwise. See also compare:.

lastPathComponent

```
- (NSString *)lastPathComponent
```

Returns the last component of the receiver's path representation. Given the path `/Images/Bloggs.tiff`, this method returns a string containing `Bloggs.tiff`. See also `pathExtension`, `completePathIntoString:caseSensitive:matchesIntoArray:filterTypes:`.

`length`

```
- (unsigned int)length
```

Returns the number of characters in the receiver. This number includes the individual characters of composed character sequences. See also `cStringLength`.

`lossyCString`

```
- (const char *)lossyCString
```

Returns a lossy C string version of the receiver.

`lowercaseString`

```
- (NSString *)lowercaseString
```

Returns a string with each character changed to its corresponding lowercase value. See also `uppercaseString`, `capitalizedString`.

`pathExtension`

```
- (NSString *)pathExtension
```

Returns the extension of the receiver's path representation. Given the path `/Images/Bloggs.tiff`, this method returns a string containing `tiff`. See also `completePathIntoString:caseSensitive:matchesIntoArray:filterTypes:`, `lastPathComponent`.

`propertyList`

```
- (id)propertyList
```

Depending on the format of the receiver's contents, returns a string, data, array, or dictionary object represention of those contents. See also `propertyListFromStringsFileFormat`.

`propertyListFromStringsFileFormat`

```
- (NSDictionary *)propertyListFromStringsFileFormat
```

Returns a dictionary object initialized with the keys and values found in the receiver. The receiver's format must be that used for ".string" files. See also `NSDictionary`, `propertyList`.

`rangeOfCharacterFromSet:`

```
- (NSRange)rangeOfCharacterFromSet:(NSCharacterSet *)aSet
```

Invokes `rangeOfCharacterFromSet:options:` with no options. See also `rangeOfCharacterFromSet:options:`, `rangeOfCharacterFromSet:options:range:`, `rangeOfString:`.

`rangeOfCharacterFromSet:options:`

```
- (NSRange)rangeOfCharacterFromSet:(NSCharacterSet *)aSet
    options:(unsigned int)mask
```

Invokes `rangeOfCharacterFromSet:options:range:` with `mask` and the entire extent of the receiver as the range. See also `rangeOfCharacterFromSet:`.

`rangeOfCharacterFromSet:options:range:`

```
- (NSRange)rangeOfCharacterFromSet:(NSCharacterSet *)aSet
    options:(unsigned int)mask range:(NSRange)aRange
```

Returns the range of the first character found from `aSet`. The search is restricted to `aRange` with `mask` options. `mask` can be any combination (using the C bitwise OR operator `|`) of `NSCaseInsensitiveSearch`, `NSLiteralSearch`, and `NSBackwardsSearch` (see theFoundation Kit's Type and Constants chapter). See also `rangeOfCharacterFromSet:`.

rangeOfComposedCharacterSequenceAtIndex:

```
- (NSRange)rangeOfComposedCharacterSequenceAtIndex:
    (unsigned int)anIndex
```

Returns an NSRange giving the location and length in the receiver of the composed character sequence located at anIndex. This method raises an NSStringBoundsError exception if anIndex lies beyond the end of the string. See also NSRange (Foundation Kit's "Types and Constants" chapter).

rangeOfString:

```
- (NSRange)rangeOfString:(NSString *)string
```

Invokes rangeOfString:options: with no options. See also NSRange (Foundation Kit's "Types and Constants" chapter), rangeOfString:options:, rangeOfString:options:range:, rangeOfCharacterFromSet:.

rangeOfString:options:

```
- (NSRange)rangeOfString:(NSString *)string
    options:(unsigned int)mask
```

Invokes rangeOfString:options:range: with mask options and the entire extent of the receiver as the range. See also NSRange (Foundation Kit's "Types and Constants" chapter), rangeOfString:.

rangeOfString:options:range:

```
- (NSRange)rangeOfString:(NSString *)aString
    options:(unsigned int)mask
    range:(NSRange)aRange
```

Returns the range giving the location and length in the receiver of aString. The search is restricted to aRange with mask options. mask can be any combination (using the C bitwise OR operator |) of the following values:

- NSCaseInsensitiveSearch
- NSLiteralSearch
- NSBackwardsSearch
- NSAnchoredSearch

See the “Searching” section of the Foundation Kit’s “Types and Constants” chapter for more information on the searching flags. See also `rangeOfString:`, `NSRange`.

`smallestEncoding`

```
- (NSStringEncoding)smallestEncoding
```

Encoding in which this string can be expressed with lossless conversion in the most space-efficient manner. See the “String” section of the Foundation Kit’s “Types and Constants” chapter for a list of encodings. See also `fastestEncoding`.

`stringByAbbreviatingWithTildeInPath`

```
- (NSString *)stringByAbbreviatingWithTildeInPath
```

Returns a string in which the user’s home directory path is replaced by ~. If the user’s home directory is not detected in the receiver, this method attempts to find any other user’s home directory path, which, if found, is replaced with ~. If no home directory component is found within the receiving string, a copy of the receiving string is returned. See also `stringByExpandingTildeInPath`.

`stringByAppendingFormat:`

```
- (NSString *)stringByAppendingFormat:(NSString *)format,...
```

Returns a string made by using `format` as a `printf()` style format string, and the following arguments as values to be substituted into the format string. See also `stringByAppendingString:`.

`stringByAppendingPathComponent:`

```
- (NSString *)stringByAppendingPathComponent:(NSString *)aString
```

Returns a string representing `aString` concatenated with the receiver. The following table illustrates this method's behavior.

Receiver	aString	Result
/	/New.tiff	/New.tiff
/Dir	New.tiff	/Dir/New.tiff
/Dir/	New.tiff	/Dir/New.tiff
@""	New.tiff	New.tiff

See also `stringByAppendingPathExtension:`, `stringByDeletingLastPathComponent`.

`stringByAppendingPathExtension:`

```
- (NSString *)stringByAppendingPathExtension:(NSString *)aString
```

Returns a string representing the receiver with the addition of the extension `aString`. The following table illustrates this method's behavior.

Receiver	aString	Result
/Dir/New.x	tiff	/Dir/New.x.tiff
/Dir/	tiff	/Dir/.tiff
New	tiff	New.tiff

See also `stringByDeletingPathExtension`, `stringByAppendingPathComponent:`.

`stringByAppendingString:`

```
- (NSString *)stringByAppendingString:(NSString *)aString
```

Returns a string formed by appending `aString` to the receiver. If `aString`'s length is 0, a copy of the receiver is returned. If the receiver's length is 0, a copy of `aString` is returned. See also `stringByAppendingFormat:`.

`stringByDeletingLastPathComponent`

```
- (NSString *)stringByDeletingLastPathComponent
```

Returns the receiver's path representation minus the last component. The following table illustrates this method's behavior.

Receiver	Result
/Dir/New.tiff	/Dir
/Dir/	/
/	/
New	@""

See also `stringByAppendingPathComponent:`, `stringByDeletingPathExtension`.

stringByDeletingPathExtension

```
- (NSString *)stringByDeletingPathExtension
```

Returns the receiver's path representation minus the extension on the last component. Given the path `/Images/Bloggs.tiff`, this method returns a string containing `/Images/Bloggs`. The following table illustrates this method's behavior.

Receiver	Result
/Dir/New.tiff	/Dir/New
/Dir/	/Dir
/	/

See also `stringByAppendingPathExtension:`, `stringByDeletingLastPathComponent`.

stringByExpandingTildeInPath

```
- (NSString *)stringByExpandingTildeInPath
```

Returns a string in which paths of the form `~user/path` or `~/path` are expanded to their full path equivalent. If no tilde is found, a copy of the receiver is returned. See also `stringByAbbreviatingWithTildeInPath`.

stringByResolvingSymlinksInPath

```
- (NSString *)stringByResolvingSymlinksInPath
```

Returns a string identical to the receiver's path except that any symbolic links have been resolved. If symbolic links can't be resolved, the empty string (@"") is returned. See also `stringByAbbreviatingWithTildeInPath`.

stringByStandardizingPath

```
- (NSString *)stringByStandardizingPath
```

Returns a string containing a "standardized" path, one in which tildes are expanded and redundant elements (for example, "/ /") eliminated. Returns the empty string (@"") if the path cannot be standardized. See also `stringByResolvingSymlinksInPath`.

stringsByAppendingPaths:

```
- (NSArray *)stringsByAppendingPaths:(NSArray *)paths
```

Appends each element of `paths` to the receiver and returns the array of resulting paths. See also `stringByAppendingPathComponent:`.

substringFromIndex:

```
- (NSString *)substringFromIndex:(unsigned int)index
```

Returns a string object containing the characters from `index` to the end of the receiver. This method raises an `NSStringBoundsError` exception if `index` lies beyond the end of the string. See also `substringWithRange:`, `substringToIndex:`.

substringWithRange:

```
- (NSString *)substringWithRange:(NSRange)aRange
```

Returns a string object containing the receiver characters that lie within `aRange`. This method raises an `NSStringBoundsError` exception if any part of `aRange` lies beyond the end of the string. See also `substringFromIndex:`, `substringToIndex:`.

substringToIndex:

```
- (NSString *)substringToIndex:(unsigned int)index
```

Returns a string object containing the characters of the receiver up to, but not including, the character at `index`. This method raises an `NSStringBoundsError` exception if `index` lies beyond the end of the string. See also `substringFromIndex:`, `substringWithRange:`.

uppercaseString

```
- (NSString *)uppercaseString
```

Returns a string with each character changed to its corresponding uppercase value. See also `lowercaseString`, `capitalizedString`.

writeToFile:atomically:

```
- (BOOL)writeToFile:(NSString *)filename
    atomically:(BOOL)useAuxiliaryFile
```

Writes a textual description of the receiver to `filename`. If `useAuxiliaryFile` is `YES`, the data is written to a backup file and then, assuming no errors occur, the backup file is renamed to the intended file name. The string is written in the default C string encoding if the contents can be converted to that encoding. If not, the string is stored in the Unicode encoding. See also `description`.

NSThread

Characteristic	Description
Inherits From:	NSObject
Conforms To:	NSObject (NSObject)
Declared In:	Foundation / NSThread.h

Class Description

An `NSThread` object controls a thread of execution. Use an `NSThread` when you want to terminate or delay a thread or you want a new thread.

A *thread* is an executable unit. A *task* is made up of one or more threads. Each thread has its own execution stack and is capable of independent I/O. All threads share the virtual memory address space and communication rights of their task. When a thread is started, it is *detached* from its initiating thread. The new thread runs independently. That is, the initiating thread does not know the new thread's state.

To obtain an `NSThread` object that represents your current thread of execution, use the `currentThread` method. To obtain an `NSThread` object that will create a new thread of execution, use `detachNewThreadSelector:toTarget:withObject:`. This method sends the specified Objective C message to the specified object in its own thread of execution. You use the `NSThread` object returned by these methods if you ever need to delay or terminate that thread of execution.

When you use `detachNewThreadSelector:toTarget:withObject:`, your application becomes multithreaded. At any time, you can send `isMultiThreaded` to find out if the application is multithreaded, that is, if a thread was ever detached from the current thread. `isMultiThreaded` returns `YES` even if the detached thread has completed execution.

Method Types

Activity	Class Method
Creating an NSThread	+ currentThread + detachNewThreadSelector:toTarget:withObject:
Querying a thread	+ isMultiThreaded – threadDictionary
Delaying a thread	+ sleepUntilDate:
Terminating a thread	+ exit

Class Methods

`currentThread`

```
+ (NSThread *)currentThread
```

Returns an object representing the current thread of execution.

detachNewThreadSelector:toTarget:withObject:

```
+ (void)detachNewThreadSelector:(SEL)aSelector toTarget:(id)aTarget
    withObject:(id)anArgument
```

Creates and starts a new NSThread for the message [aTarget aSelector:anArgument]. The method aSelector may take only one argument and may not have a return value. If this is the first thread detached from the current thread, this method posts the notification NSBecomingMultiThreaded with the nil object to the default notification center.

exit

```
+ (void)exit
```

Terminates the thread represented by the calling object. Before exiting that thread, this method posts the NSThreadExiting notification with the thread being exited to the default notification center.

isMultiThreaded

```
+ (BOOL)isMultiThreaded
```

Returns YES if a thread was ever detached whether or not the detached thread is still running.

sleepUntilDate:

```
+ (void)sleepUntilDate:(NSDate *)date
```

Sends the receiving NSThread to sleep until the time specified by date. No input or timers will be processed in this interval.

Instance Methods

threadDictionary

```
- (NSMutableDictionary *)threadDictionary
```

Returns the NSThread's dictionary, allowing you to add data specific to the receiving NSThread. This allows user-defined NSThread variables.

5

NSTimer

Characteristic	Description
Inherits From:	NSObject
Conforms To:	NSObject (NSObject)
Declared In:	Foundation / NSTimer.h

Class Description

`NSTimer` creates timer objects. A timer object waits until a certain time interval has elapsed and then fires, sending a specified message to a specified object. For example, you could create an `NSTimer` that sends a message to a window, telling it to update itself after a certain time interval.

`NSTimer` objects work in conjunction with `NSRunLoop` objects. `NSRunLoop`s control loops that wait for input, and they use `NSTimer`s to help determine the maximum amount of time they should wait. When the `NSTimer`'s time limit has elapsed, the `NSRunLoop` fires the `NSTimer` (causing its message to be sent), then checks for new input.

There are several ways to create an `NSTimer` object. The `scheduledTimerWithTimeInterval...` class methods automatically register the new `NSTimer` with the current `NSRunLoop` object in default mode. The `timerWithTimeInterval...` class methods create `NSTimer`s that the user may register at a later time by sending the message `addTimer:forMode:` to the `NSRunLoop`. If you specify that the `NSTimer` should repeat, it will automatically reschedule itself after it fires. If a delay occurs when a timer is scheduled to fire, the timer will not fire. For example, suppose you used the following statement to create a timer:

```
timer = [NSTimer scheduledTimerWithTimeInterval:0.5
         invocation:anInvocation repeats:YES];
```

This statement creates a timer that will schedule itself to fire after 0.5 seconds, 1 second, 1.5 seconds, and so on from the time this statement is executed. Suppose there was a 2-second delay because `NSRunLoop` was busy processing input. The timer takes this delay into consideration and will skip intervals that were already missed when computing the next scheduled fire date.

There is no method that removes the association of an NSTimer from an NSRunLoop—send the NSTimer the invalidate message instead. invalidate disables the NSTimer, so it will no longer affect the NSRunLoop.

See the NSRunLoop class description for more information on NSRunLoops.

As a consequence of being a subclass of NSObject, NSTimer conforms to the NSCoding protocol. In practice, however, NSTimers are neither encoded nor archived.

Method Types

Activity	Class Method
Creating a timer object	+ scheduledTimerWithTimeInterval: invocation:repeats: + scheduledTimerWithTimeInterval:target: selector:userInfo:repeats: + timerWithTimeInterval:invocation:repeats: + timerWithTimeInterval:target: selector:userInfo:repeats:
Firing the timer	– fire
Stopping the timer	– invalidate
Getting information about the NSTimer	– fireDate – isValid – userInfo

Class Methods

scheduledTimerWithTimeInterval: invocation:repeats:

```
+ (NSTimer *)scheduledTimerWithTimeInterval:(NSTimeInterval)seconds
    invocation:(NSInvocation *)anInvocation repeats:(BOOL)repeats
```

Returns a new NSTimer object and registers it with the current NSRunLoop in the default mode. After seconds seconds have elapsed, the NSTimer fires, sending anInvocation's message to its target. If repeats is YES, the NSTimer will repeatedly reschedule itself.

scheduledTimerWithTimeInterval:target:selector:userInfo:repeats:

```
+ (NSTimer *)scheduledTimerWithTimeInterval:(NSTimeInterval)seconds
    target:(id)anObject selector:(SEL)aSelector
    userInfo:(id)anArgument repeats:(BOOL)repeats
```

Returns a new NSTimer object and registers it with the current NSRunLoop in the default mode. After seconds seconds have elapsed, the NSTimer fires, sending the message [anObject aSelector:self]. If anObject needs more information, it can send the NSTimer a userData message to retrieve anArgument. If repeats is YES, the NSTimer will repeatedly reschedule itself.

timerWithTimeInterval:invocation:repeats:

```
+ (NSTimer *)timerWithTimeInterval:(NSTimeInterval)seconds
    invocation:(NSInvocation *)anInvocation repeats:(BOOL)repeats
```

Returns a new NSTimer that, if registered, will fire after seconds seconds. Upon firing, the NSTimer sends anInvocation's message to its target. If repeats is YES, the NSTimer will repeatedly reschedule itself.

timerWithTimeInterval:target:selector:userInfo:repeats:

```
+ (NSTimer *)timerWithTimeInterval:(NSTimeInterval)seconds
    target:(id)anObject selector:(SEL)aSelector
    userInfo:(id)anArgument repeats:(BOOL)repeats
```

Returns a new NSTimer that, if registered, will fire after seconds seconds. Upon firing, the NSTimer sends the message [anObject aSelector:self]. If anObject needs more information, it can send the NSTimer a userData message to retrieve anArgument. If repeats is YES, the NSTimer will repeatedly reschedule itself.

Instance Methods

fire

```
- (void)fire
```

Causes the NSTimer's message to be dispatched to its target.

fireDate

```
- (NSDate *)fireDate
```

Returns the date that the NSTimer will next fire. Returns nil if the timer object is not valid.

invalidate

```
- (void)invalidate
```

Stops the NSTimer from ever firing again. See also isValid.

isValid

```
- (BOOL)isValid
```

Returns YES if the timer is valid, and returns NO otherwise. See also invalidate.

userInfo

```
- userInfo
```

Additional data that the object receiving NSTimer's message can use. The default implementation returns nil.

5

NSTimeZone

Characteristic	Description
Inherits From:	NSObject
Conforms To:	NSCoding, NSCopying NSObject (NSObject)
Declared In:	Foundation/NSDate.h

Class Description

`NSTimeZone` is an abstract class that defines the behavior of time-zone objects. By itself, `NSDate` represents dates as *universal time*. Universal time treats a date and time value as identical in, for instance, Redwood City and New York City. `NSDate` has no provision for locale adjustment of time-zone information. Provision for locale is critical for string descriptions and other expressions of conventional dates and times. `NSTimeZone` is used to affect the apparent value of date objects so that they reflect time-zone related locale information.

`NSTimeZoneDetail`, a public subclass of `NSTimeZone`, augments the behavior of `NSTimeZone` by providing the commonly known attributes of a time zone in effect for a date within a time zone geopolitical area. These attributes are abbreviation, the offset from Greenwhich Mean Time (GMT), and an indication of whether Daylight Savings Time (DST) is in effect.

Time-zone objects represent geopolitical regions and use names to denote the various regions. For example, "US/Pacific" identifies the geopolitical time zone for San Francisco and Los Angeles, which falls in the same general latitude as that for the time zone "Canada/Pacific." The US/Pacific time-zone has specific `NSTimeZoneDetail` instances that specify Pacific Standard Time (PST) and Pacific Daylight Time (PDT), which have slightly different offsets from GMT.

You typically associate the objects returned by `NSTimeZone` and, by extension, `NSTimeZoneDetail` with date objects to affect their behavior. Time-zone objects can be of various types:

- time zones with hour and minute offsets from GMT
- time zones with a single abbreviation and offset
- time zones that vary according to Standard Time and DST

The system should supply the various choices for time zones along with time-zone information. These choices should be restricted to subsets based on latitude. You can access these choices through the `timeZoneArray` class method. Another restriction is the choice of time zone available when there is an ambiguous abbreviation; these choices are available through the class method `abbreviationDictionary`. Despite these restrictions, you can obtain an `NSTimeZone` object from an arbitrary file through the class method `timeZoneWithName`.

Note – By itself, the `NSTimeZone` class only *names* a time zone. It does not associate an abbreviation or a temporal offset with a time zone; that is done by `NSTimeZoneDetail`. An instance of `NSTimeZone`, however, "knows" about the set of time-zone detail objects related to it.

`NSTimeZone` provides several class methods to get time-zone objects, with or without detail: `timeZoneWithName:`, `timeZoneWithAbbreviation:`, and `timeZoneForSecondsFromGMT:`. The class also permits you to set the default time zone used by your application for your locale (`setDefaultTimeZone:`) You can access this default time zone at any time by the `defaultTimeZone` method, and, with the `localTimeZone` class method, you can also get a relative time-zone object that will decode itself to become the default time zone for any locale in which it finds itself.

`NSCalendarDate` methods return date objects that are automatically bound with time-zone detail objects. These date objects use the functionality of `NSTimeZone` to adjust dates for the proper locale. Unless you specify otherwise, objects returned from `NSCalendarDate` are bound to the default time zone for the current locale. A useful instance method is `timeZoneDetailForDate:`, which returns a time-zone detail object associated with a specific date.

Method Types

Activity	Class Method
Creating and initializing an NSTimeZone	+ defaultTimeZone + localTimeZone + timeZoneForSecondsFromGMT: + timeZoneWithAbbreviation: + timeZoneWithName: – timeZoneDetailForDate:
Managing time zones	+ setDefaultTimeZone:
Getting time zone information	+ abbreviationDictionary – timeZoneName
Getting arrays of time zones	+ timeZoneArray – timeZoneDetailArray

Class Methods

abbreviationDictionary

```
+ (NSDictionary *)abbreviationDictionary
```

Returns a dictionary that maps abbreviations to region names, for example "PST" is the key for "US/Pacific". If you know a region name for a key, you can obtain a valid abbreviation from the dictionary and use it to obtain a detail time-zone object using `timeZoneWithAbbreviation:`. See also `NSDictionary`.

defaultTimeZone

```
+ (NSTimeZoneDetail *)defaultTimeZone
```

Returns the default time zone as set for the current locale. Default time-zone objects remain constant as they travel around the globe. For example, if you create a default time-zone object in California, under PST, and send that object to New York, the object still represents PST. See also `localTimeZone`, `NSTimeZoneDetail`.

localTimeZone

```
+ (NSTimeZone *)localTimeZone
```

Returns an NSTimeZone that behaves as the current default time zone in any given locale. The local time-zone objects "change" as they travel around the globe. For example, if you create a local time-zone object in California, under PST, and send that object to New York, the object will then represent EST. See also defaultTimeZone.

setDefaultTimeZone:

```
+ (void)setDefaultTimeZone:(NSTimeZone *)aTimeZone
```

Sets aTimeZone as the time zone appropriate for the current locale. This new time zone replaces the previous default time zone.

timeZoneArray

```
+ (NSArray *)timeZoneArray
```

Returns an array of string object arrays, each containing strings that show current geopolitical names for each time zone. The subarrays are grouped by latitudinal region. See also NSArray.

timeZoneForSecondsFromGMT:

```
+ (NSTimeZone *)timeZoneForSecondsFromGMT:(int)seconds
```

Returns an NSTimeZone representing the time zone with seconds offset from GMT. If there is no object matching the offset, this method creates and returns a new NSTimeZone bearing the value seconds as a name.

timeZoneWithAbbreviation:

```
+ (NSTimeZoneDetail *)timeZoneWithAbbreviation:
    (NSString *)abbreviation
```

Returns the time-zone object identified by the abbreviation abbreviation. If there is no match, this method returns nil.

timeZoneWithName:

```
+ (NSTimeZone *)timeZoneWithName:(NSString *)aTimeZoneName
```

Returns the time-zone object with the name that corresponds to the geopolitical region `aTimeZoneName`. It searches the region's dictionary for matching names. If there is no match on the name, this method returns `nil`.

Instance Methods

timeZoneDetailForDate:

```
- (NSTimeZoneDetail *)timeZoneDetailForDate:(NSDate *)date
```

Returns the correct time-zone detail object associated with a date object. You invoke this method when a region's time zone (that is, its offset value from GMT) varies over the year, as happens between Standard Time and Daylight Savings Time.

timeZoneName

```
- (NSString *)timeZoneName
```

Returns the geopolitical name of the time zone.

timeZoneDetailArray

```
- (NSArray *)timeZoneDetailArray
```

Returns an array of `NSTimeZoneDetail` objects that are associated with the receiving `NSTimeZone` object.

NSTimeZoneDetail

Characteristic	Description
Inherits From:	NSTimeZone : NSObject
Conforms To:	NSCoding, NSCopying (NSTimeZone) NSObject (NSObject)
Declared In:	Foundation / NSDate.h

Class Description

`NSTimeZoneDetail` is an abstract class that refines the behavior provided by `NSTimeZone`. `NSTimeZone` identifies a geopolitical area with a name (such as US / Pacific). `NSTimeZoneDetail` augments this region name with more specific information appropriate for a particular date within its geopolitical region: an abbreviation, an offset (in seconds) from Greenwich Mean Time (GMT), and an indication of whether Daylight Savings Time is in effect. The specificity afforded through `NSTimeZoneDetail` helps to resolve conflicts between abbreviations and offsets that can arise within regions.

Even though it is a concrete subclass of `NSTimeZone`, `NSTimeZoneDetail` does *not* have "factory" class methods that create and return time-zone objects. See the specification of `NSTimeZone` for methods that provide this ability.

However, `NSTimeZoneDetail` does have methods that allow you to get the abbreviation and temporal offset of a time-zone object, as well as determine whether Daylight Savings Time is in effect.

Method Types

Activity	Class Method
Querying an NSTimeZoneDetail	– isDaylightSavingTimeZone – timeZoneAbbreviation – timeZoneSecondsFromGMT

Instance Methods

isDaylightSavingTimeZone

```
– (BOOL)isDaylightSavingTimeZone
```

Returns YES if the time-zone detail object is used in the representation of dates during Daylight Savings Time, and returns NO otherwise.

timeZoneAbbreviation

```
– (NSString *)timeZoneAbbreviation
```

Returns the abbreviation of the time-zone detail object, such as EDT (Eastern Daylight Time).

timeZoneSecondsFromGMT

```
– (int)timeZoneSecondsFromGMT
```

Returns the difference in seconds between the receiving time-zone detail object and GMT. The offset can be a positive or negative value.

NSUnarchiver

Characteristic	Description
Inherits From:	NSCoder : NSObject
Conforms To:	NSObject (NSObject)
Declared In:	Foundation / NSArchiver.h

Class Description

`NSUnarchiver`, a concrete subclass of `NSCoder`, defines objects that can decode a data structure, such as a graph of Objective C objects, from an archive. Such archives are produced by objects of the `NSArchiver` class. See the `NSArchiver` specification for an introduction to archiving.

General Exception Conditions

While unarchiving, `NSUnarchiver` performs a variety of consistency checks on the incoming data stream. `NSUnarchiver` raises an `NSInconsistentArchiveException` for a variety of reasons. Possible data errors leading to this exception are: unknown type descriptors in the data file; an array type descriptor is incorrectly terminated (that is, a missing “]”); excess characters in a type descriptor; a null class found where a concrete class was expected; class not loaded.

Method Types

Activity	Class Method
Initializing an NSUnarchiver	– initForReadingWithData:
Decoding objects	+ unarchiveObjectWithData: + unarchiveObjectWithFile: – decodeArrayOfObjCType:count:at:
Managing an NSUnarchiver	– isAtEnd – objectZone – setObjectZone: – systemVersion
Substituting one class for another	+ classNameDecodedForArchiveClassName: + decodeClassName:asClassName: – classNameDecodedForArchiveClassName: – decodeClassName:asClassName:

Class Methods

classNameDecodedForArchiveClassName:

```
+ (NSString *)classNameDecodedForArchiveClassName:
    (NSString *)nameInArchive
```

Returns the class name used to archive instances of the class (`nameInArchive`). This may not be the original class name but another name encoded with `NSArchiver`'s `encodeClassName:intoClassName:`.

decodeClassName:asClassName:

```
+ (void)decodeClassName:(NSString *)nameInArchive
    asClassName:(NSString *)trueName
```

Decodes from the archived data a class name (`nameInArchive`) substituted for the real class name (`trueName`). This method enables easy conversion of unarchived data when there are name changes in classes.

unarchiveObjectWithData:

```
+ (id)unarchiveObjectWithData:(NSData *)data
```

Decodes an archived object stored in `data`.

`unarchiveObjectWithFile:`

```
+ (id)unarchiveObjectWithFile:(NSString *)path
```

Decodes an archived object stored in the file `path`.

Instance Methods

`classNameDecodedForArchiveClassName:`

```
- (NSString *)classNameDecodedForArchiveClassName:
    (NSString *)nameInArchive
```

Returns the class name used to archive instances of the class (`nameInArchive`). This may not be the original class name but another name encoded with `NSArchiver`'s `encodeClassName:intoClassName:`.

`decodeArrayOfObjCType:count:at:`

```
- (void)decodeArrayOfObjCType:(const char *)itemType
    count:(unsigned int)count at:(void *)array
```

Decodes an `array` of `count` data elements of the same Objective C data `itemType`. It is your responsibility to release any objects derived in this way. `itemType` can be some combination of the type descriptors in the following table.

Descriptor	Type
id	@
Class	#
SEL	:
char	c
unsigned char	C
short	s
unsigned short	S

Descriptor	Type
int	i
unsigned int	I
long	l
unsigned long	L
float	f
double	d
bitfield	b
void	v
undefined	?
pointer	^
char *	*
array	[<count><types>]
union	(<types>)
structure	{<types>}

For example, if `itemType` were "{sic*@}", the array to be decoded would contain structures containing a short, an int, a char, a char *, and an object. See also `encodeArrayOfObjCType:count:at:` (`NSArchiver`).

`decodeClassName:asClassName:`

```
- (void)decodeClassName:(NSString *)nameInArchive
    asClassName:(NSString *)trueName
```

Decodes from the archived data a class name (`nameInArchive`) substituted for the real class name (`trueName`). This method enables easy conversion of unarchived data when there are name changes in classes.

`initForReadingWithData:`

```
- (id)initForReadingWithData:(NSData *)data
```

Initializes an `NSUnarchiver` object from data object `data`. Raises `NSInvalidArgumentException` if the `data` argument is `nil`.

isAtEnd

```
- (BOOL)isAtEnd
```

Returns YES if the end of data is reached, NO if more data follows.

objectZone

```
- (NSZone *)objectZone
```

Returns the allocation zone for the unarchiver object.

setObjectZone:

```
- (void)setObjectZone:(NSZone *)zone
```

Sets the allocation zone for the unarchiver object to zone. If zone is nil, it sets it to the default zone.

systemVersion

```
- (unsigned int)systemVersion
```

Returns the system version number for the unarchived data.

NSUserDefaults

Characteristic	Description
Inherits From:	NSObject
Conforms To:	NSObject (NSObject)
Declared In:	Foundation / NSUserDefaults.h

Class Description

The NSUserDefaults class allows an application to query and manipulate a user's defaults settings. Defaults are grouped in domains. For example, there's a domain for application-specific defaults and another for global defaults. Each domain has a name and stores defaults as key-value pairs in an

NSDictionary object. A default is identified by a string key, and its value can be any property-list object (NSData, NSString, NSArray, or NSDictionary). The standard domains are:

Table 5-4 Domains for User Defaults

Domain	Identifier
Argument	NSArgumentDomain
Application	Identified by the application's name
Global	NSGlobalDomain
Languages	Identified by the language names
Registration	NSRegistrationDomain

Above identifiers starting with "NS" are global constants.

The argument domain is composed of defaults parsed from the application's arguments. The application domain contains the defaults set by the application. It is identified by the name of the application, as returned by this message:

```
NSString *applicationName = [[NSProcessInfo processInfo]
                                processName];
```

The global domain contains defaults that are meant to be seen by all applications. The registration domain is a set of temporary defaults whose values can be set by the application to ensure that searches for default values will always be successful. Applications can create additional domains as needed.

A search for the value of a given default proceeds through the domains listed in an NSUserDefault object's *search list*. Only domains in the search list are searched. The standard search list contains the domains from the table above, in the order listed. A search ends when the default is found. If multiple domains contain the same default, only the domain nearest the beginning of the search list provides the default's value. Using the searchList method, you can reorder the default search list or set up one that is a subset of all the user's domains.

Typically, you use this class by invoking the `standardUserDefaults` class method to get an `NSUserDefaults` object. This method returns a global `NSUserDefaults` object with a search list already initialized. Then use the `setObject:forKey:` and `objectForKey:` methods to set and access user defaults.

The rest of the methods allow more complex defaults management. You can create your own domains, modify any domain, set up a custom search list, and even control the synchronization of the in-memory and on-disk defaults representations. The `synchronize` method saves any modifications to the persistent domains and updates all persistent domains that were not modified to what is on disk. `synchronize` is automatically invoked at periodic intervals.

You can create either persistent or volatile domains. Persistent domains are permanent and last past the life of the `NSUserDefaults` object. Any changes to the persistent domains are committed to disk. Volatile domains last only as long as the `NSUserDefaults` object exists. The `NSGlobalDomain` domain is persistent; the `NSArgumentDomain` is volatile.

Be warned that:

- User defaults are not thread safe.
- Automatic saving of changes to disk (through `synchronize`) depends on a run-loop being present.
- You should synchronize any domain you have altered before exiting a process.

Method Types

Activity	Class Method
Getting the shared instance	+ standardUserDefaults
Getting and setting a default	– arrayForKey: – boolForKey: – dataForKey: – dictionaryForKey: – floatForKey: – integerForKey: – objectForKey: – removeObjectForKey: – setBool:forKey: – setFloat:forKey: – setInteger:forKey: – setObject:forKey: – stringArrayForKey: – stringForKey:
Initializing the user defaults	– init – initWithUser:
Returning the search list	– searchList – setSearchList:
Maintaining peristent domains	– persistentDomainForName: – persistentDomainNames – removePersistentDomainForName: – setPersistentDomain:forName: – synchronize
Maintaining volatile domains	– removeVolatileDomainForName: – setVolatileDomain:forName: – volatileDomainForName: – volatileDomainNames
Making advanced use of defaults	– dictionaryRepresentation – registerDefaults:

Class Methods

standardUserDefaults

```
+ (NSUserDefaults *)standardUserDefaults
```

Returns the shared defaults object. If it doesn't exist yet, it's created with a search list containing the names of the following domains, in order:

- `NSArgumentDomain` (consisting of defaults parsed from the application's arguments)
- A domain with the process' name
- Separate domains for each of the user's preferred languages
- the `NSGlobalDomain` (consisting of defaults meant to be seen by all applications)
- the `NSRegistrationDomain` (a set of temporary defaults whose values can be set by the application to ensure that searches will always be successful)

The defaults are initialized for the current user. Subsequent modifications to the standard search list remain in effect even when this method is invoked again—the search list is guaranteed to be standard only the first time this method is invoked. The shared instance is provided as a convenience; other instances may also be created.

Instance Methods

arrayForKey:

```
- (NSArray *)arrayForKey:(NSString *)defaultName
```

Invokes `objectForKey:` with key `defaultName`. Returns the corresponding value if it's an `NSArray` object (according to the `isKindOfClass:` test) and `nil` otherwise.

boolForKey:

```
- (BOOL)boolForKey:(NSString *)defaultName
```

Invokes `stringForKey:` with key `defaultName`. Returns `YES` if the corresponding value is an `NSString` containing uppercase or lowercase `YES` or responds to the `intValue` message by returning a nonzero value. Otherwise, returns `NO`.

dataForKey:

```
- (NSData *)dataForKey:(NSString *)defaultName
```

Invokes `objectForKey:` with key `defaultName`. Returns the corresponding value if it's an `NSData` object (according to the `isKindOfClass:` test) and `nil` otherwise.

dictionaryForKey:

```
- (NSDictionary *)dictionaryForKey:(NSString *)defaultName
```

Invokes `objectForKey:` with key `defaultName`. Returns the corresponding value if it's an `NSDictionary` object (according to the `isKindOfClass:` test) and `nil` otherwise.

dictionaryRepresentation

```
- (NSDictionary *)dictionaryRepresentation
```

Returns a dictionary that contains a union of all key-value pairs in the domains in the search list. As with `objectForKey:`, key-value pairs in domains that are earlier in the search list take precedence. The combined result doesn't preserve information about which domain each entry came from.

floatForKey:

```
- (float)floatForKey:(NSString *)defaultName
```

Invokes `stringForKey:` with key `defaultName`. Returns 0 if no string is returned. Otherwise, the resulting string is sent a `floatValue` message, which provides this method's return value.

init

```
- (id)init
```

Initializes defaults for the current user, who is identified by examining the environment. This method doesn't put anything in the search list. Invoke it only if you've allocated your own `NSUserDefaults` object instead of using the shared one. Returns `self`.

initWithUser:

```
- (id)initWithUser:(NSString *)userName
```

Like `init`, but initializes defaults for the specified user.

integerForKey:

```
- (int)integerForKey:(NSString *)defaultName
```

Invokes `stringForKey:` with key `defaultName`. Returns 0 if no string is returned. Otherwise, the resulting string is sent an `intValue` message, which provides this method's return value.

objectForKey:

```
- (id)objectForKey:(NSString *)defaultName
```

Returns the value of the first occurrence of the specified default, searching the domains included in the search list. Returns `nil` if the default isn't found.

persistentDomainForName:

```
- (NSDictionary *)persistentDomainForName:(NSString *)domainName
```

Returns a dictionary corresponding to the specified persistent domain. The keys in the dictionary are names of defaults, and the value corresponding to each key is a property list data object.

persistentDomainNames

```
- (NSArray *)persistentDomainNames
```

Returns an array containing the names of the persistent domains. Each domain can then be retrieved by invoking `persistentDomainForName:`.

registerDefaults:

```
- (void)registerDefaults:(NSDictionary *)dictionary
```

Adds the contents of `dictionary` to the registration domain. If there is no registration domain yet, it's created using `dictionary`, and `NSRegistrationDomain` is added to the end of the search list.

removeObjectForKey:

```
- (void)removeObjectForKey:(NSString *)defaultName
```

Removes the value for the given default in the standard application domain. Removing a default has no effect on the value returned by the `objectForKey:` method if the same key exists in a domain that precedes the standard application domain in the search list.

removePersistentDomainForName:

```
- (void)removePersistentDomainForName:(NSString *)domainName
```

Removes the named persistent domain from the user's defaults. The first time that a persistent domain is changed after `synchronize`, an `NSUserDefaultsChanged` notification is posted.

removeVolatileDomainForName:

```
- (void)removeVolatileDomainForName:(NSString *)domainName
```

Removes the named volatile domain from the user's defaults.

searchList

```
- (NSArray *)searchList
```

Returns an array of domain names that `objectForKey:` will search. Non-existent domain names in the list are ignored. See also `setSearchList:`, `objectForKey:`.

setBool:forKey:

```
- (void)setBool:(BOOL)value forKey:(NSString *)defaultName
```

Sets the value of the specified default to a string representation of YES or NO, depending on value. Invokes setObject:forKey: as part of its implementation.

setFloat:forKey:

```
- (void)setFloat:(float)value forKey:(NSString *)defaultName
```

Sets the value of the specified default to a string representation of value. Invokes setObject:forKey: as part of its implementation.

setInteger:forKey:

```
- (void)setInteger:(int)value forKey:(NSString *)defaultName
```

Sets the value of the specified default to a string representation of value. Invokes setObject:forKey: as part of its implementation.

setObject:forKey:

```
- (void)setObject:(id)value
```

Sets the value of the specified default in the standard application domain. Setting a default has no effect on the value returned by the objectForKey: method if the same key exists in a domain that precedes the application domain in the search list.

setPersistentDomain:forName:

```
- (void)setPersistentDomain:(NSDictionary *)domain
    forName:(NSString *)domainName
```

Sets the dictionary for the persistent domain named domainName; raises an NSInvalidArgumentException if a volatile domain with domainName already exists. The first time that a persistent domain is changed after synchronize, an NSUserDefaultsChanged notification is posted.

setSearchList:

```
- (void)setSearchList:(NSArray *)newSearchList
```

Sets the domain name list searched by objectForKey:. See also `searchList`, `objectForKey:`.

`setVolatileDomain:forName:`

```
- (void)setVolatileDomain:(NSDictionary *)domain
    forName:(NSString *)domainName
```

Sets the dictionary to `domain` for the volatile domain named `domainName`. This method raises an `NSInvalidArgumentException` if a persistent domain with `domainName` already exists.

`stringArrayForKey:`

```
- (NSArray *)stringArrayForKey:(NSString *)defaultName
```

Invokes `objectForKey:` with key `defaultName`. Returns the corresponding value if it's an `NSArray` object containing `NSString`s, and returns `nil` otherwise. The class of each object is determined using the `isKindOfClass:` test.

`stringForKey:`

```
- (NSString *)stringForKey:(NSString *)defaultName
```

Invokes `objectForKey:` with key `defaultName`. Returns the corresponding value if it's an `NSString` object according to the `isKindOfClass:` test, and returns `nil` otherwise.

`synchronize`

```
- (BOOL)synchronize
```

Saves any modifications to the persistent domains and updates all persistent domains that were not modified to what is on disk. Returns `NO` if it could not save data to disk. Since the `synchronize` method is automatically invoked at periodic intervals, use this method only if you cannot wait for the automatic synchronization (for example if your application is about to exit), or if you want to update user defaults to what is on disk even though you have not made any changes.

volatileDomainForName:

```
- (NSDictionary *)volatileDomainForName:(NSString *)domainName
```

Returns a dictionary corresponding to the specified volatile domain. The keys in the dictionary are names of defaults, and the value corresponding to each key is a property list data object.

volatileDomainNames

```
- (NSArray *)volatileDomainNames
```

Returns an array containing the names of the volatile domains. Each domain can then be retrieved by calling `volatileDomainForName:`.

NSValue

Characteristic	Description
Inherits From:	NSObject
Conforms To:	NSCoding, NSCopying NSObject (NSObject)
Declared In:	Foundation / NSValue.h Foundation / NSGeometry.h

Class Description

`NSValue` objects provide an object-oriented wrapper for the data types defined in standard C and Objective C. The `NSValue` class is often used to put Objective C and standard C data types into collections that require objects, such as `NSArray` objects. When a value object is instantiated, it is encoded with the specified data type.

The `NSValue` class declares the programmatic interface to an object that contains a C data type. It provides methods for creating value objects that contain values of a specified data type, pointers, and other objects. Use `NSValue` objects to put C types into collections. Use `NSNumber` objects to put numbers into collections.

The following code puts an NSRange into an NSArray, using the Objective C @encode directive to get a character string that encodes the type structure of NSRange:

```
[myArray insertObject:[NSValue value:&range
                      withObjCType:@encode(NSRange)] atIndex:n]
```

To get the value back, you would do this:

```
[[myArray objectAtIndex:n] getValue:&range]
```

NSValue objects are provided with generic coding and copying behavior. To subclass NSValue and preserve class when encoding or copying, override classForCoder, initWithCoder:, encodeWithCoder: (for encoding), and copyWithZone: (for copying).

General Exception Conditions

NSValue can raise NSInternalInconsistencyException in a variety of cases where an unknown Objective C type is found. In addition, NSValue's implementation of encodeWithCoder: can raise NSInvalidArgumentException if an attempt is made to encode void.

Method Types

Activity	Class Method
Allocating and initializing value objects	+ valueWithBytes:objCType: + value:withObjCType: + valueWithNonretainedObject: + valueWithPointer: – initWithBytes:objCType:
Allocating and initializing geometry value objects	+ valueWithPoint: + valueWithRect: + valueWithSize:
Accessing data in value objects	– getValue: – nonretainedObjectValue – objCType – pointerValue
Accessing data in value geometry objects	– pointValue – rectValue – sizeValue
Equality	– isEqualToValue:

Class Methods

valueWithBytes:objCType:

```
+ (NSValue *)valueWithBytes:(const void *)value
    objCType:(const char *)type
```

Creates and returns a value object initialized to `value` and of specified Objective C `type`. See also `value:withObjCType:`, `initWithBytes:objCType:`.

valueWithNonretainedObject:

```
+ (NSValue *)valueWithNonretainedObject: (id)anObject
```

Creates and returns a value object containing the object `anObject`, without retaining `anObject`. This is provided as a convenience method. The following statement

```
[NSValue valueWithNonretainedOject:anObject]
```

is equivalent to the statement

```
[NSValue value:&anObject withObjCType:@encode(void *)].
```

value:withObjCType:

```
+ (NSValue *)value:(const void *)value
    withObjCType:(const char *)type
```

Creates and returns a value object containing the value `value` of the Objective C type `type`.

valueWithPoint:

```
+ (NSValue *)valueWithPoint:(NSPoint)point
```

Creates and returns a value object that contains the specified `NSPoint` structure which represents a geometrical point in two dimensions. See also `NSPoint`.

valueWithPointer:

```
+ (NSValue *)valueWithPointer:(const void *)pointer
```

Creates and returns a value object that contains the specified pointer. This is provided as a convenience method. The following statement

```
[NSValue valueWithPointer:pointer]
```

is equivalent to the statement

```
[NSValue value:&pointer withObjCType:@encode(void *)].
```

valueWithRect:

```
+ (NSValue *)valueWithRect:(NSRect)rect
```

Creates and returns a value object that contains the specified `NSRect` structure, representing a rectangle. See also `NSRect`.

valueWithSize:

```
+ (NSValue *)valueWithSize:(NSSize)size
```

Creates and returns a value object that contains the specified `NSSize` structure which stores a width and a height. See also `NSSize`.

Instance Methods

initWithBytes:objCType:

```
- (id) initWithBytes:(const void *)value objCType:(const char *)type
```

Returns an Objective C `type`, initialized to value. See also `valueWithBytes:objCType:`.

isEqualToValue:

```
- (BOOL)isEqualToValue:(NSValue *)otherValue
```

Returns `YES` if the receiver is equal to `otherValue`.

getValue:

```
– (void)getValue:(void *)value
```

Copies the receiver's data into `value`.

nonretainedObjectValue

```
– (id)nonretainedObjectValue
```

Returns the nonretained object that is contained in the receiver. It is an error to send this message to an `NSValue` object that doesn't store a nonretained object.

objCType

```
– (const char *)objCType
```

Returns the Objective C type of the data contained in the receiver.

pointValue

```
– (NSPoint)pointValue
```

Returns the NSPoint structure that is contained in the receiver. See also NSPoint.

pointerValue

```
- (void *)pointerValue
```

Returns the value pointed to by a pointer contained in a value object. It's an error to send this message to an NSValue that doesn't store a pointer.

rectValue

```
- (NSRect)rectValue
```

Returns the rectangle structure that is contained in the receiver. See also NSRect.

sizeValue

```
- (NSSize)sizeValue
```

Returns the size structure that's contained in the receiver. See also NSSize.

NSCoding

Characteristic	Description
Adopted By:	NSObject
Declared In:	Foundation / NSObject.h

Protocol Description

The `NSCoding` protocol declares the two methods that a class must implement so that objects of that class can be encoded and decoded. This capability provides the basis for *archiving* (where objects and other structures are stored on disk) and *distribution* (where objects are copied to different address spaces).

When an object receives an `encodeWithCoder:` message, it should write its instance variables (and, through a message to `super`, the instance variables that it inherits) to the supplied `NSCoder`. Similarly, when an object receives an `initWithCoder:` message, it should initialize its instance variables (and inherited instance variables, again through a message to `super`) from the data in the supplied `NSCoder`. See the `NSCoder` and `NSArchiver` class specifications for more complete information.

Instance Methods

encodeWithCoder:

```
- (void)encodeWithCoder:(NSCoder *)aCoder
```

Encodes the receiver using `aCoder`.

initWithCoder:

```
- (id)initWithCoder:(NSCoder *)aDecoder
```

Initializes and returns a new instance from data in `aDecoder`.

NSCopying

Characteristic	Description
Adopted By:	Various OpenStep classes
Declared In:	Foundation / NSObject.h

Protocol Description

A class whose instances provide functional copies of themselves should adopt the `NSCopying` protocol. The exact meaning of "copy" can vary from class to class, but a copy must be a functionally independent object, identical to the original at the time the copy was made. Where the concept "immutable vs. mutable" applies to an object, this protocol produces immutable copies; see the `NSMutableCopying` protocol for details on making mutable copies. Property list classes (`NSString`, `NSData`, `NSArray`, and `NSDictionary`) guarantee immutable returned values.

In most cases, to produce a copy that's independent of the original, a *deep copy* must be made. In a deep copy every instance variable of the receiver is duplicated, instead of referencing the variable in the original object. If the receiver's instance variables themselves have instance variables, those too must be duplicated, and so on. A deep copy is thus a completely separate object from the original; changes to it don't affect the original, and changes to the original don't affect it. Further, for an immutable copy, no part at any level may be changed, making a copy a "snapshot" of the original object.

Making a complete deep copy isn't always needed. Some objects can reasonably share instance variables among themselves—a static string object that gets replaced but not modified, for example. In such cases your class can implement `NSCopying` more cheaply than it might otherwise need to.

The typical usage of `NSCopying` is to create "passing by value" value objects.

Note – Contrary to most methods, the returned object is owned by the caller, which is responsible for releasing it.

Instance Methods

`copyWithZone:`

```
- (id)copyWithZone:(NSZone *)zone
```

Returns a new instance that's a functional copy of the receiver. Memory for the new instance is allocated from zone. For collections, creates a deep (recursive) copy. The copy returned is immutable if the consideration "immutable vs. mutable" applies to the receiving object; otherwise the exact nature of the copy is determined by the class. The returned object is owned by the caller, who is responsible for releasing it.

NSLocking

Characteristic	Description
Adopted By:	NSConditionLock NSLock NSRecursiveLock
Declared In:	Foundation / NSLock.h

Protocol Description

`NSLocking` protocol is used by classes that provide lock objects. The lock objects provided by OpenStep are used only for protecting critical sections of code: sections that manipulate shared data and that can be executed simultaneously in several threads. Lock objects—except for `NSConditionLock` objects—contain no useful data.

Although an object that isn't a lock could adopt the NSLocking protocol, it may be more desirable to design the object so that all locking is handled internally, through normal use rather than requiring that the object be explicitly locked and unlocked.

In order to enable clients to only have locks when processes become multithreaded, it is permissible to unlock a lock freshly created (i.e. that has not been locked)—unless it is a recursive lock. Three classes conform to the NSLocking protocol:

Table 6-1 Classes That Conform to NSLocking Protocol

Class	Use
NSLock	Protect critical sections of code.
NSConditionLock	Protects critical sections of code, but can also be used to postpone entry to a critical section until a condition is met. This class is functionally a superset of the NSLock class, though unlocking is slightly more expensive.
NSRecursiveLock	Protects critical sections from access by multiple threads, but allows a single thread to acquire a lock several times without deadlocking.

None of these classes busy-waits while the lock is unavailable. All classes may all be efficiently used for long sections of atomic code. See the class specifications for these classes for further information on their behavior and usage.

Instance Methods

lock

```
- (void)lock
```

Acquires a lock. Applications generally do this when entering a critical section of their code. A thread will sleep if it can’t immediately acquire the lock.

unlock

```
- (void)unlock
```

Releases a lock. Applications generally do this when exiting a critical section of their code.

NSMutableCopying

Characteristic	Description
Adopted By:	various OpenStep classes
Declared In:	Foundation / NSObject.h

Protocol Description

A class that defines an “immutable vs. mutable” distinction adopts this protocol to allow mutable copies of its instances to be made. A mutable copy of an object is usually a *shallow copy* (as opposed to the *deep copy* defined in the `NSCopying` protocol specification). The original and its copy share references to the same instance variables, so that if a component of the copy is changed, for example, that change is reflected in the original.

A class that doesn’t define an “immutable vs. mutable” distinction but that needs to offer both deep and shallow copying shouldn’t adopt this protocol. The `NSCopying` methods should by default be assumed to produce deep copies; the class can then also implement methods to produce shallow copies.

Note – Contrary to most methods, the returned value is owned by the caller, which is responsible for releasing it.

Instance Methods

`mutableCopyWithZone:`

```
- (id)mutableCopyWithZone:(NSZone *)zone
```

Returns a new instance that’s a top level, mutable copy of the receiver. For a collection, objects in the collection are retained. Memory for the new instance is allocated from `zone`. The returned object is owned by the caller, which is responsible for releasing it.

NSObjCTypeSerializationCallBack

Characteristic	Description
Adopted By:	No OpenStep classes
Declared In:	Foundation / NSSerialization.h

Protocol Description

An object conforms to the `NSObjCTypeSerializationCallBack` protocol so that it can intervene in the serialization and deserialization process. The primary purpose of this protocol is to allow for the serialization of objects and other data types that aren't directly supported by OpenStep's serialization facility. (See the `NSSerializer` class specification for information on serialization.)

`NSMutableData` declares the method that's used to begin the serialization process:

```
- (void)serializeDataAt:(const void *)data
  ofObjCType:(const char *)type
  context:(id <NSObjCTypeSerializationCallBack>)callback
```

This method can serialized all standard Objective C types (`int`, `float`, character strings, and so on) except for objects, `union`, and `void *`. If, during the serialization process, an object is encountered, the object passed as the callback argument above is asked to provide the serialization.

Suppose that the type being serialized is a structure of this description:

```
struct stockRecord {
  NSString *stockName;
  float value;
};
```

The Objective C type code for this structure is `{@f}`, so the serialization process begins with this message: (Assume that `theData` is the `NSMutableData` object that's doing the serialization and `helper` is an object that conforms to the `NSObjCTypeSerializationCallBack` protocol.)

```
struct stockRecord aRecord = {@"aCompany", 34.7};
[theData serializeDataAt:&aRecord
    ofObjCType:"{@f}" context:helper];
```

Since the first field of the structure is an unsupported type, the helper object is sent a `serializeObjectAt:ofObjCType:intoData:` message, letting it serialize the object. `helper` might implement the method in this way:

```
- (void)serializeObjectAt:(id *)objectPtr
    ofObjCType:(const char *)type
    intoData:(NSMutableData *)theMutableData
{
    NSString *nameObject;
    char *companyName

    nameObject = *objectPtr;
    companyName = [nameObject cString];

    [theData serializeDataAt:&companyName
        ofObjCType:@encode(typeof(companyName))
        context:nil]
}
```

The callback object is free to serialize the target object as it wishes. In this case, `helper` simply extracts the company name from the `NSString` object and then has that character string serialized. Once this callback method finishes executing, the original method (`serializeDataAt:ofObjCType:context:`) resumes execution and serializes the second field of the structure. Since this second field contains a supported type (`float`), the callback method is not invoked again.

Deserialization follows a similar pattern, except in this case `NSData` declares the central method `deserializeDataAt:ofObjCType:atCursor:context:`. The deserialization of the example structure starts with a message to the `NSData` object that contains the serialized data:

```
(unsigned *)cursor = 0;

[theData deserializeDataAt:&aRecord ofObjCType:"{@f}"
    cursor:&cursor context:helper];
```

(The cursor argument is a pointer to zero since we're starting at the beginning of the data in the `NSData` object.)

When this method is invoked, the callback object receives a `deserializeObjectAt:ofObjCType:fromData:atCursor:` message, as declared in this protocol. The callback object can then reestablish the first field of the structure. For example, `helper` might implement the method in this way:

```
- (void) deserializeObjectAt:(id *)objectPtr
  ofObjCType:(const char *)type
  fromData:(NSData *)data
  atCursor:(unsigned *)cursor
{
  char *companyName;

  [theData deserializeDataAt:&companyName ofObjCType:"*"
    atCursor:cursor context:nil];
  *objectPtr = [[NSString stringWithCString:companyName] retain];
}
```

Instance Methods

`deserializeObjectAt:ofObjCType: fromData:atCursor:`

```
– (void)deserializeObjectAt:(id *)object
    ofObjCType:(const char *)type fromData:(NSData *)data
    atCursor:(unsigned int*)cursor
```

The implementor of this method decodes the referenced `object` (which should always be of `type` "@") located at the `cursor` position in the `data` object. The decoded object is *not* autoreleased. See the description of `NSData` method `deserializeDataAt:ofObjCType:context:`.

`serializeObjectAt:ofObjCType:intoData:`

```
– (void)serializeObjectAt:(id *)object
    ofObjCType:(const char *)type
    intoData:(NSMutableData *)data
```

The implementor of this method encodes the referenced `object` (which should always be of `type` "@") in the `data` object. See the description of `NSMutableData` method `serializeDataAt:ofObjCType:context:`.

NSObject

Characteristic	Description
Adopted By:	NSObject
Declared In:	Foundation / NSObject.h

Protocol Description

The `NSObject` protocol declares methods that all objects should implement within OpenStep, no matter which root class they descend from (`NSObject`, `NSProxy`, or another root class). Some of the methods in this protocol reveal an object's primary attributes: its position in the class hierarchy, its conformance to other protocols, and whether it responds to specific messages. Other methods let the object be manipulated in various ways. For example, it can be asked to perform methods that are detemined at runtime (using the `performSelector:...` methods) or to participate in OpenStep's automatic deallocation scheme (using the `retain`, `release`, and `autorelease` methods). By conforming to this protocol, an object advertises that it has the basic behaviors necessary to work with the OpenStep container classes (such as `NSArray` and `NSDictionary`).

Instance Methods

autorelease

```
- (id)autorelease
```

As defined in the `NSObject` class, decrements the receiver's reference count. When the count reaches 0, adds the object to the current autorelease pool. Returns `self`. Objects in the pool are released later, typically at the top of the event loop.

class

```
- (Class)class
```

Returns the receiver's class object. See also `superclass`.

conformsToProtocol:

```
- (BOOL) conformsToProtocol:(Protocol *aProtocol)
```

Returns YES if the receiver's class conforms to aProtocol.

description

```
- (NSString)description
```

Returns text information about the receiver.

hash

```
- (unsigned int)hash
```

Returns an unsigned int that can be used as a table address in a hash table structure. Two objects that are equal must hash to the same value.

isEqual:

```
- (BOOL)isEqual:(id)anObject
```

Returns YES if the receiver and anObject have equal values, and returns NO otherwise.

isKindOfClass:

```
- (BOOL)isKindOfClass:(Class)aClass
```

Returns YES if the receiver is an instance of aClass or an instance of any class that inherits from aClass. Returns NO otherwise.

isMemberOfClass:

```
- (BOOL)isMemberOfClass:(Class)aClass
```

Returns YES if the receiver is an instance of aClass. Returns NO otherwise.

isProxy

```
- (BOOL)isProxy
```

Returns YES is an NSProxy, rather than an object that descends from the NSObject class. Returns NO otherwise.

performSelector:

```
- (id)performSelector:(SEL)selector
```

Sends an aSelector messsage to the receiver and returns the result of the message. If aSelector is NULL, and NSInvalidArgumentException is raised.

performSelector:withObject:

```
- (id)performSelector:(SEL)selector withObject:(id)anObject
```

Sends an aSelector messsage to the receiver with anObject as an argument, and returns the message result. If aSelector is NULL, and NSInvalidArgumentException is raised.

performSelector:withObject:withObject:

```
- (id)performSelector:(SEL)selector withObject:(id)anObject
    withObject:anotherObject
```

Sends an aSelector messsage to the receiver with anObject and anotherObject as arguments, and returns the message result. If aSelector is NULL, and NSInvalidArgumentException is raised.

release

```
- (void)release
```

Decrements the receiver's reference count. When the count reaches 0, the object is automatically deallocated immediately.

respondsToSelector:

```
- (BOOL)respondsToSelector:(SEL)aSelector
```

Returns YES if the receiver implements or inherits a method that can respond to aSelector messages. Returns NO otherwise.

retain

```
- (id)retain
```

As defined in the `NSObject` class, increments the receiver's reference count. Send an object a `retain` message when you want to prevent it from being deallocated without your permission. Returns self as a convenience. See also `retainCount`, `release`, `autorelease`.

retainCount

```
- (unsigned int)retainCount
```

Returns the receiver's reference count. This is useful for debugging.

self

```
- (id)self
```

Returns the receiver.

superclass

```
- (Class)superclass
```

Returns the class object for the receiver's superclass. See also `class`.

zone

```
- (NSZone *)zone
```

Returns a pointer to memory zone from which the receiver was allocated.

Functions 7

Memory Allocation Functions

Get the Virtual Memory Page Size

NSPageSize()

```
unsigned NSPageSize(void)
```

Returns the number of bytes in a page.

NSLogPageSize()

```
unsigned NSLogPageSize(void)
```

Returns the binary log of the page size.

NSRoundDownToMultipleOfPageSize()

```
unsigned NSRoundDownToMultipleOfPageSize(unsigned byteCount)
```

Returns the multiple of the page size that is closest to, but not greater than, `byteCount`. See also `NSRoundUpToMultipleOfPageSize()`.

NSRoundUpToMultipleOfPageSize()

```
unsigned NSRoundUpToMultipleOfPageSize(unsigned byteCount)
```

Returns the multiple of the page size that is closest to, but not less than, `byteCount`. See also `NSRoundDownToMultipleOfPageSize()`.

Get the Amount of Real Memory

NSRealMemoryAvailable()

```
unsigned NSRealMemoryAvailable(void)
```

Returns the number of bytes available in the RAM hardware.

Allocate or Free Virtual Memory

NSAllocateMemoryPages()

```
void *NSAllocateMemoryPages(unsigned byteCount)
```

Allocates the integral number of pages whose total size is closest to, but not less than, `byteCount`, with the pages guaranteed to be zero-filled. See also `NSDeallocateMemoryPages()`.

NSDeallocateMemoryPages()

```
void NSDeallocateMemoryPages(void *pointer, unsigned byteCount)
```

Deallocates `byteCount` of memory, pointed to by `pointer`, that was allocated with `NSAllocateMemoryPages()`.

NSCopyMemoryPages()

```
void NSCopyMemoryPages(const void *source, void *destination,
          unsigned byteCount)
```

Copies (or copies-on-write) `byteCount` bytes from `source` to `destination`.

Get a Zone

NSCreateZone()

```
NSZone *NSCreateZone(unsigned startSize, unsigned granularity,
    BOOL canFree)
```

Creates and returns a pointer to a new zone of `startSize` bytes, which will grow and shrink by `granularity` bytes. If `canFree` is `NO`, the allocator will never free memory, and `malloc()` will be fast. See also `NSDefaultMallocZone()`.

NSDefaultMallocZone()

```
NSZone *NSDefaultMallocZone(void)
```

Returns the default zone, which is created automatically at startup. This is the zone used by the standard C function `malloc()`.

NSZoneFromPointer()

```
NSZone *NSZoneFromPointer(void *pointer)
```

Returns the zone for the `pointer` block of memory, or `NULL` if the block was not allocated from a zone. The `pointer` must be one that was returned by a prior call to an allocation function. See also `NSCreateZone()`.

Allocate or Free Memory in a Zone

The following methods should be used instead of `malloc()`. Note that if the `zone` argument is given as `(NSZone *)0`, the default zone is used.

NSZoneMalloc()

```
void *NSZoneMalloc(NSZone *zone, unsigned size)
```

Allocates `size` bytes in `zone`, and returns a pointer to the allocated memory. See also `NSZoneCalloc()`, `NSZoneRealloc()`.

NSZoneCalloc()

```
void *NSZoneCalloc(NSZone *zone, unsigned numElems,
    unsigned numBytes)
```

Allocates enough memory from `zone` for `numElems` elements, each with a size of `numBytes` bytes, and returns a pointer to the allocated memory. The memory is initialized with zeros. See also `NSZoneMalloc()`, `NSZoneRealloc()`.

NSZoneRealloc()

```
void *NSZoneRealloc(NSZone *zone, void *pointer, unsigned size)
```

Changes the size of the block of memory pointed to by `pointer` to `size` bytes. It may allocate new memory to replace the old, in which case it moves the contents of the old memory block to the new block, up to a maximum of `size` bytes. The `pointer` may be `NULL`. See also `NSZoneMalloc()`, `NSZoneCalloc()`.

NSRecycleZone()

```
void NSRecycleZone(NSZone *zone)
```

Frees `zone` after adding any of its pointers still in use to the default zone. (This strategy prevents retained objects from being inadvertently destroyed.)

NSZoneFree()

```
void NSZoneFree(NSZone *zone, void *pointer)
```

Returns memory to the `zone` from which it was allocated. The standard C function `free()` does the same, but spends time finding which zone the memory belongs to.

Name a Zone

NSSetZoneName()

```
void NSSetZoneName(NSZone *zone, NSString *name)
```

Sets the specified zone's name to name, which can aid in debugging. See also NSZoneName().

NSZoneName()

```
NSString *NSZoneName(NSZone *zone)
```

Returns the zone's name. See also NSSetZoneName().

Object Allocation Functions

Allocate or Free an Object

NSAllocateObject()

```
NSObject *NSAllocateObject(Class aClass, unsigned extraBytes,
    NSZone *zone)
```

Allocates and returns a pointer to an instance of aClass, created in the specified zone (or in the default zone, if zone is NULL). The extraBytes argument (usually zero) states the number of extra bytes required for indexed instance variables. See also NSCopyObject(), NSDeallocateObject().

NSCopyObject()

```
NSObject *NSCopyObject(NSObject *anObject, unsigned extraBytes,
    NSZone *zone)
```

Creates and returns a new object that's an exact copy of anObject. The second and third arguments have the same meaning as in NSAllocateObject(). See also NSDeallocateObject().

NSDeallocateObject()

```
void NSDeallocateObject(NSObject *anObject)
```

Deallocates anObject, which must have been allocated using NSAllocateObject(). See also NSCopyObject().

Decide Whether to Retain an Object

NSShouldRetainWithZone()

```
BOOL NSShouldRetainWithZone(NSObject *anObject,
    NSZone *requestedZone)
```

Returns YES if `requestedZone` is `NULL`, the default zone, or the zone in which `anObject` was allocated. This function is typically called from inside an `NSObject`'s `copyWithZone:` method, when deciding whether to retain `anObject` as opposed to making a copy of it.

Modify the Number of References to an Object

NSDecrementExtraRefCountWasZero()

```
BOOL NSDecrementExtraRefCountWasZero(id anObject)
```

Returns YES if the externally maintained "extra reference count" for `anObject` is zero; otherwise, this function decrements the count and returns `NO`.

NSExtraRefCount()

```
unsigned int NSExtraRefCount(id anObject)
```

Returns the externally maintained "extra reference count".

NSIncrementExtraRefCount()

```
void NSIncrementExtraRefCount(id anObject)
```

Increments the externally maintained "extra reference count" for anObject. The first reference (typically done in `NSObject`'s `alloc` method) isn't maintained externally, so there's no need to call this function for that first reference.

Error-Handling Functions

Change the Top-Level Error Handler

NSGetUncaughtExceptionHandler()

```
NSUncaughtExceptionHandler *NSGetUncaughtExceptionHandler(void)
```

Returns a pointer to the function serving as the top-level error handler. This handler will process exceptions raised outside of any exception-handling domain.

NSSetUncaughtExceptionHandler()

```
void NSSetUncaughtExceptionHandler(NSUncaughtExceptionHandler
    *handler)
```

Sets the top-level error-handling function to handler. If handler is `NULL` or this function is never invoked, the default top-level handler is used.

Macros to Handle an Exception

NS_DURING

```
NS_DURING
```

Marks the beginning of an exception-handling domain (a portion of code delimited by NS_DURING and NS_HANDLER). When an error is raised anywhere within the exception-handling domain, program execution jumps to the first line of code in the exception handler. It's illegal to exit the exception-handling domain by any other means than NS_VALUERETURN, NS_VOIDRETURN, or falling out the bottom.

NS_ENDHANDLER

```
NS_ENDHANDLER
```

Marks the ending of an exception handler (a portion of code delimited by NS_HANDLER and NS_ENDHANDLER).

NS_HANDLER

```
NS_ENDHANDLER
```

Marks the ending of an exception-handling domain and the beginning of the corresponding exception handler. Within the scope of the handler, a local variable called *localException* (of type `NSException *`) stores the raised exception. Code delimited by `NS_HANDLER` and `NS_ENDHANDLER` is never executed except when an error is raised in the preceding exception-handling domain.

NS_VALUERETURN

```
value NS_VALUERETURN(value, type)
```

Causes the method (or function) in which this macro occurs to immediately return `value` of type `type`. This macro can only be placed within an exception-handling domain.

NS_VOIDRETURN

```
NS_VOIDRETURN
```

Causes the method (or function) in which this macro occurs to return immediately, with no return value. This macro can only be placed within an exception-handling domain.

Call the Assertion Handler from the Body of an Objective-C Method

NSAssert()

```
NSAssert(BOOL condition, NSString *description)
```

Calls the `NSAssertionHandler` object for the current thread if `condition` is false. The `description` should explain the error, formatted as for the standard C function `printf()`; it need not include the object's class and method name, since they're passed automatically to the handler.

NSAssert1()

```
NSAssert1(BOOL condition, NSString *description, arg)
```

Like `NSAssert()`, but the format string `description` includes a conversion specification (such as `%s` or `%d`) for the argument `arg`, in the style of `printf()`. You can pass an object in `arg` by specifying `%@`, which gets replaced by the string that the object's description method returns.

NSAssert2()

```
NSAssert2(BOOL condition, NSString *description, arg1, arg2)
```

Like `NSAssert1()`, but with two arguments.

NSAssert3()

```
NSAssert3(BOOL condition, NSString *description, arg1, arg2, arg3)
```

Like `NSAssert1()`, but with three arguments.

NSAssert4()

```
NSAssert4(BOOL condition, NSString *description, arg1, arg2,
    arg3, arg4)
```

Like `NSAssert1()`, but with four arguments.

NSAssert5()

```
NSAssert5(BOOL condition, NSString *description, arg1, arg2,
    arg3, arg4, arg5)
```

Like `NSAssert1()`, but with five arguments.

Call the Assertion Handler from the Body of a C Function

NSCAssert()

```
NSCAssert(BOOL condition, NSString *description)
```

Calls the `NSAssertionHandler` object for the current thread if `condition` is false. The `description` should explain the error, formatted as for the standard C function `printf()`; it need not include the function name, which is passed automatically to the handler.

NSCAssert1()

```
NSCAssert1(BOOL condition, NSString *description, arg)
```

Like `NSCAssert()`, but the format string description includes a conversion specification (such as `%s` or `%d`) for the argument `arg`, in the style of `printf()`. See also `NSAssert()`.

NSCAssert2()

```
NSCAssert2(BOOL condition, NSString *description, arg1, arg2)
```

Like `NSCAssert1()`, but with two arguments.

NSCAssert3()

```
NSCAssert3(BOOL condition, NSString *description, arg1, arg2, arg3)
```

Like `NSCAssert1()`, but with three arguments.

NSCAssert4()

```
NSCAssert4(BOOL condition, NSString *description, arg1, arg2,
    arg3, arg4)
```

Like NSCAssert1(), but with four arguments.

NSCAssert5()

```
NSCAssert5(BOOL condition, NSString *description, arg1, arg2,
    arg3, arg4, arg5)
```

Like NSCAssert1(), but with five arguments.

Validate a Parameter

NSParameterAssert()

```
NSParameterAssert(BOOL condition)
```

Like NSAssert(), but the description passed to the assertion handler is "Invalid parameter not satisfying: " followed by the text of condition (which can be any boolean expression). See also NSCParameterAssert.

NSCParameterAssert

```
NSCParameterAssert(BOOL condition)
```

Like NSParameterAssert(), but to be called from the body of a C function.

Geometric Functions

Create Basic Structures

NSMakePoint()

```
NSPoint NSMakePoint(float x, float y)
```

Create an NSPoint having the coordinates x and y.

NSMakeSize()

```
NSSize NSMakeSize(float width, float height)
```

Create an NSSize having the specified width and height.

NSMakeRect()

```
NSRect NSMakeRect(float x, float y, float width, float height)
```

Create an NSRect having the specified origin and size.

NSMakeRange()

```
NSRange NSMakeRange(unsigned int location, unsigned int length)
```

Create an NSRange having the specified location and length.

Get a Rectangle's Coordinates

NSMaxX()

```
float NSMaxX(NSRect aRect)
```

Returns the largest x-coordinate value within aRect. See also NSMidX(), NSMinX(), NSMaxY().

NSMaxY()

```
float NSMaxY(NSRect aRect)
```

Returns the largest y-coordinate value within aRect. See also NSMidY(), NSMinY(), NSMaxX(),

NSMidX()

```
float NSMidX(NSRect aRect)
```

Returns the x-coordinate of the rectangle's center point. See also NSMaxX(), NSMinX(), NSMidY().

NSMidY()

```
float NSMidY(NSRect aRect)
```

Returns the y-coordinate of the rectangle's center point. See also NSMaxY(), NSMinY(), NSMidX().

NSMinX()

```
float NSMinX(NSRect aRect)
```

Returns the smallest x-coordinate value within aRect. See also NSMaxX(), NSMidX(), NSMinY().

NSMinY()

```
float NSMinY(NSRect aRect)
```

Returns the smallest y-coordinate value within aRect. See also NSMaxY(), NSMidY(), NSMinX().

NSWidth()

```
float NSWidth(NSRect aRect)
```

Returns the width of aRect. See also NSHeight().

NSHeight()

```
float NSHeight(NSRect aRect)
```

Returns the height of aRect. See also NSWidth().

Modify a Copy of a Rectangle

NSInsetRect()

```
NSRect NSInsetRect(NSRect aRect, float dX, float dY)
```

Returns a copy of the rectangle aRect, altered by moving the two sides that are parallel to the y-axis inwards by dX, and the two sides parallel to the x-axis inwards by dY. See also NSOffsetRect().

NSOffsetRect()

```
NSRect NSOffsetRect(NSRect aRect, float dX, float dY)
```

Returns a copy of the rectangle aRect, with its location shifted by dX along the x-axis and by dY along the y-axis. See also NSInsetRect().

NSDivideRect()

```
void NSDivideRect(NSRect inRect, NSRect *slice, NSRect *remainder,
          float amount, NSRectEdge edge)
```

Creates two rectangles, `slice` and `remainder`, from `inRect`, by dividing `inRect` with a line that's parallel to one of `inRect`'s sides (namely, the side specified by edge—either `NSMinXEdge`, `NSMinYEdge`, `NSMaxXEdge`, or `NSMaxYEdge`). The size of slice is determined by amount, which measures the distance from edge. See also `NSIntegralRect()`.

NSIntegralRect()

```
NSRect NSIntegralRect(NSRect aRect)
```

Returns a copy of the rectangle `aRect`, expanded outwards just enough to ensure that none of its four defining values (x, y, width, and height) have fractional parts. If `aRect`'s width or height is zero or negative, this function returns a rectangle with origin at (0.0, 0.0) and with zero width and height. See also `NSDivideRect()`.

Compute a Third Rectangle from Two Rectangles

NSUnionRect()

```
NSRect NSUnionRect(NSRect aRect, NSRect bRect)
```

Returns the smallest rectangle that completely encloses both `aRect` and `bRect`. If one of the rectangles has zero (or negative) width or height, a copy of the other rectangle is returned; but if both have zero (or negative) width or height, the returned rectangle has its origin at (0.0, 0.0) and has zero width and height. See also `NSIntersectionRect()`.

NSIntersectionRect()

```
NSRect NSIntersectionRect(NSRect aRect, NSRect bRect)
```

Returns the graphic intersection of `aRect` and `bRect`. If the two rectangles don't overlap, the returned rectangle has its origin at (0.0, 0.0) and zero width and height. (This includes situations where the intersection is a point or a line segment.) See also `NSUnionRect()`.

Test Geometric Relationships

NSContainsRect()

```
BOOL NSContainsRect(NSRect aRect, NSRect bRect)
```

Returns YES if aRect is equal to or completely encloses bRect, and neither aRect nor bRect is "empty".

NSEqualPoints()

```
BOOL NSEqualPoints(NSPoint aPoint, NSPoint bPoint)
```

Returns YES if the two points aPoint and bPoint are identical, and NO otherwise.

NSEqualRects()

```
BOOL NSEqualRects(NSRect aRect, NSRect bRect)
```

Returns YES if the two rectangles aRect and bRect are identical, and NO otherwise.

NSEqualSizes()

```
BOOL NSEqualSizes(NSSize aSize, NSSize bSize)
```

Returns YES if the two sizes aSize and bSize are identical, and NO otherwise.

NSIntersectsRect()

```
BOOL NSIntersectsRect(NSRect aRect, NSRect bRect)
```

Returns YES if aRect and bRect intersect, and returns NO otherwise. See also NSIntersectionRect().

NSIsEmptyRect()

```
BOOL NSIsEmptyRect(NSRect aRect)
```

Returns YES if aRect encloses no area at all—that is, if its width or height is zero or negative.

NSMouseInRect()

```
BOOL NSMouseInRect(NSPoint aPoint, NSRect aRect, BOOL flipped)
```

Returns YES if the point represented by `aPoint` is located within the rectangle represented by `aRect`. It assumes an unscaled and unrotated coordinate system; the argument flipped should be YES if the coordinate system has been flipped so that the positive y-axis extends downward. This function is used to determine whether the hot spot of the cursor lies inside a given rectangle. See also `NSPointInRect()`.

NSPointInRect()

```
BOOL NSPointInRect(NSPoint aPoint, NSRect aRect)
```

Performs the same test as `NSMouseInRect()`, but assumes a flipped coordinate system.

Conversion To and From String Representations

NSStringFromPoint()

```
NSString *NSStringFromPoint(NSPoint aPoint)
```

Returns a string of the form "{x=a; y=b}", where a and b are the x and y coordinates of `aPoint`.

NSPointFromString()

```
NSPoint NSStringFromPoint(NSString *aString)
```

Returns a point from a string containing the substrings "x = a" and "y = b", where "a" and "b" specify the x and y point coordinates. Returns a point with coordinates {0.0, 0.0} if the string does not convert.

NSStringFromRect()

```
NSString *NSStringFromRect(NSRect aRect)
```

Returns a string of the form "{x=a; y=b; width=c; height=d}", where a, b, c, and d are the x- and y-coordinates and the width and height, respectively, of `aRect`.

NSRectFromString()

```
NSRect NSRectFromString(NSString *aString)
```

Returns a rectangle object from a string containing the following substrings “x = a”, “y = b”, “width = c”, “height = d”; where a, b, c, and d are the x and y coordinates, and the width and height of the rectangle. Returns a rectangle object of {0.0, 0.0, 0.0, 0.0} if the string does not convert.

NSStringFromSize()

```
NSString *NSStringFromSize(NSSize aSize)
```

Returns a string of the form “{width=a; height=b}”, where a and b are the width and height of `aSize`.

NSSizeFromString()

```
NSSize NSSizeFromString(NSString* aString)
```

Returns a size object from a string containing the substrings “width = a”, and “height = b”, where “a” and “b” specify the width and height of the size object. Returns a size object with dimensions {0.0, 0.0} if the string does not convert.

7

Range Functions

Query a Range

NSEqualRanges()

```
BOOL NSEqualRanges(NSRange range1, NSRange range2)
```

Returns YES if `range1` and `range2` have the same locations and lengths.

NSLocationInRange()

```
BOOL NSLocationInRange(unsigned location, NSRange range)
```

Returns YES if `location` is in range (that is, if location is greater than or equal to `range.location` and `location` is less than `NSMaxRange(range)`).

NSMaxRange()

```
unsigned NSMaxRange(NSRange range)
```

Returns `range.location` + `range.length`—in other words, the number one greater than the maximum value within the range.

Compute a Range from Two Other Ranges

NSIntersectionRange()

```
NSRange NSIntersectionRange(NSRange range1, NSRange range2)
```

Returns a range whose maximum value is the lesser of `range1`'s and `range2`'s maximum values, and whose location is the greater of the two range's locations. However, if the two ranges don't intersect, the returned range has a location and length of zero.

NSUnionRange()

```
NSRange NSUnionRange(NSRange range1, NSRange range2)
```

Returns a range whose maximum value is the greater of `range1`'s and `range2`'s maximum values, and whose location is the lesser of the two range's locations.

Convrsion To and From a String Representation

`NSStringFromRange()`

```
NSString *NSStringFromRange(NSRange range)
```

Returns a string of the form: "{location = a; length = b}", where a and b are non-negative integers.

`NSRangeFromString()`

```
NSRange NSRangeFromString(NSString* aString)
```

Returns range object from a string containing the substrings "location = a", and "length = b", where a and b are origin and length of the range object. Returns a range object with values of {0,0} if the string does not convert.

Hash Table Functions

Create a Table

`NSCreateHashTable()`

```
NSHashTable *NSCreateHashTable(NSHashTableCallBacks callBacks,
    unsigned capacity)
```

Creates, and returns a pointer to, an `NSHashTable` object in the default zone; the table's size is dependent on (but generally not equal to) `capacity`. If `capacity` is 0, a small hash table is created. The `NSHashTableCallBacks` structure `callBacks` has five pointers to functions (documented under "Types and Constants"), with the following defaults:

- Pointer hashing, if `hash()` is `NULL`;
- Pointer equality, if `isEqual()` is `NULL`;
- No call-back upon adding an element, if `retain()` is `NULL`;

- No call-back upon removing an element, if `release()` is `NULL`;
- A function returning a pointer's hexadecimal value as a string, if `describe()` is `NULL`.

The hashing function must be defined such that if two data elements are equal, as defined by the comparison function, the values produced by hashing on these elements must also be equal. Also, data elements must remain invariant if the value of the hashing function depends on them; for example, if the hashing function operates directly on the characters of a string, that string can't change. See also `NSCreateHashTableWithZone()`.

NSCreateHashTableWithZone()

```
NSHashTable *NSCreateHashTableWithZone(
    NSHashTableCallBacks callBacks, unsigned capacity, NSZone *zone)
```

Like `NSCreateHashTable()`, but creates the hash table in `zone` instead of in the default zone. (If `zone` is `NULL`, the default zone is used.) See also `NSCopyHashTableWithZone()`.

NSCopyHashTableWithZone()

```
NSHashTable *NSCopyHashTableWithZone(NSHashTable *table,
    NSZone *zone)
```

Returns a pointer to a new copy of `table`, created in `zone`, and containing copies of `table`'s pointers to data elements. If `zone` is `NULL`, the default zone is used. See also `NSCreateHashTableWithZone()`.

Free a Table

NSFreeHashTable()

```
void NSFreeHashTable(NSHashTable *table)
```

Releases each element of the specified hash `table` and frees the `table` itself.

NSResetHashTable()

```
void NSResetHashTable(NSHashTable *table)
```

Releases each element but doesn't deallocate the `table`. This is useful for preserving the table's capacity.

Compare Two Tables

NSCompareHashTables()

```
BOOL NSCompareHashTables(NSHashTable *table1, NSHashTable *table2)
```

Returns `YES` if the two hash tables are equal—that is, if each element of `table1` is in `table2`, and the two tables are the same size.

Get the Number of Items

NSCountHashTable()

```
unsigned NSCountHashTable(NSHashTable *table)
```

Returns the number of elements in the hash `table`.

Retrieve Items

NSHashGet()

```
void *NSHashGet(NSHashTable *table, const void *pointer)
```

Returns the pointer in the table that matches `pointer` (as defined by the `isEqual()` call-back function) within `table`. If there is no matching element, the function returns `NULL`.

NSAllHashTableObjects()

```
NSArray *NSAllHashTableObjects(NSHashTable *table)
```

Returns an array object containing all the elements of `table`. This function should be called only when the `table` elements are objects, not when they’re any other data type.

NSEnumerateHashTable()

```
NSHashEnumerator NSEnumerateHashTable(NSHashTable *table)
```

Returns an `NSHashEnumerator` structure that will cause successive elements of table to be returned each time this enumerator is passed to `NSNextHashEnumeratorItem()`.

NSNextHashEnumeratorItem()

```
void *NSNextHashEnumeratorItem(NSHashEnumerator *enumerator)
```

Returns the next element in the table that `enumerator` is associated with, or `NULL` if `enumerator` has already iterated over all the elements.

Add or Remove an Item

NSHashInsert()

```
void NSHashInsert(NSHashTable *table, const void *pointer)
```

Inserts `pointer`, which must not be `NULL`, into `table`. If `pointer` matches an item already in the table, the previous pointer is released using the `release()` call-back function that was specified when the `table` was created. See also `NSHashInsertKnownAbsent()`.

NSHashInsertKnownAbsent()

```
void NSHashInsertKnownAbsent(NSHashTable *table,
    const void *pointer)
```

Inserts `pointer`, which must not be `NULL`, into table. Unike `NSHashInsert()`, this function raises `NSInvalidArgumentException` if table already includes an element that matches pointer.

NSHashInsertIfAbsent()

```
void *NSHashInsertIfAbsent(NSHashTable *table, const void *pointer)
```

If `pointer` matches an item already in table, this function returns the pre-existing pointer; otherwise, it adds `pointer` to the table and returns `NULL`.

NSHashRemove()

```
void NSHashRemove(NSHashTable *table, const void *pointer)
```

If `pointer` matches an item already in `table`, this function releases the pre-existing item.

Get a String Representation

NSStringFromHashTable()

```
NSString *NSStringFromHashTable(NSHashTable *table)
```

Returns a string describing the hash table's contents. The function iterates over the `table`'s elements, and for each element appends the string returned by the `describe()` call-back function. If `NULL` was specified for the call-back function, the hexadecimal value of each pointer is added to the string.

Map Table Functions

Create a Table

NSCreateMapTable()

```
NSMapTable *NSCreateMapTable(NSMapTableKeyCallBacks keyCallBacks,
    NSMapTableValueCallBacks valueCallBacks, unsigned capacity)
```

Creates, and returns a pointer to, an `NSMapTable` in the default zone; the table's size is dependent on (but generally not equal to) `capacity`. If capacity is 0, a small map table is created. The `NSMapTableKeyCallBacks` arguments are structures (documented under "Types and Constants") that are very similar to the call-back structure used by `NSCreateHashTable()`; in fact, they have the same defaults as documented for that function. See also `NSCreateHashTable()`.

NSCreateMapTableWithZone()

```
NSMapTable *NSCreateMapTableWithZone(
    NSMapTableKeyCallBacks keyCallBacks,
    NSMapTableValueCallBacks valueCallBacks,
    unsigned capacity, NSZone *zone)
```

Like NSCreateMapTable(), but creates the map table in zone instead of in the default zone. If zone is NULL, the default zone is used.

NSCopyMapTableWithZone()

```
NSMapTable *NSCopyMapTableWithZone(NSMapTable *table, NSZone *zone)
```

Returns a pointer to a new copy of table, created in zone and containing copies of table's key and value pointers. If zone is NULL, the default zone is used.

Free a Table

NSFreeMapTable()

```
void NSFreeMapTable(NSMapTable *table)
```

Releases each key and value of the specified map table and frees the table itself. See also NSResetMapTable().

NSResetMapTable()

```
void NSResetMapTable(NSMapTable *table)
```

Releases each key and value but doesn't deallocate the table. This is useful for preserving the table's capacity. See also NSFreeMapTable().

Compare Two Tables:

NSCompareMapTables()

```
BOOL NSCompareMapTables(NSMapTable *table1, NSMapTable *table2)
```

Returns YES if each key of table1 is in table2, and the two tables are the same size. Note that this function does not compare values, only keys.

Get the Number of Items

NSCountMapTable()

```
unsigned NSCountMapTable(NSMapTable *table)
```

Returns the number of key/value pairs in table.

Retrieve Items

NSMapMember()

```
BOOL NSMapMember(NSMapTable *table, const void *key,
    void **originalKey, void **value)
```

Returns YES if table contains a key equal to key. If so, originalKey is set to key, and value is set to the value that the table maps to key.

NSMapGet()

```
void *NSMapGet(NSMapTable *table, const void *key)
```

Returns the value that table maps to key, or NULL if the table doesn't contain key.

NSEnumerateMapTable()

```
NSMapEnumerator NSEnumerateMapTable(NSMapTable *table)
```

Returns an NSMapEnumerator structure that will cause successive key/value pairs of table to be visited each time this enumerator is passed to NSNextMapEnumeratorPair().

NSNextMapEnumeratorPair()

```
BOOL NSNextMapEnumeratorPair(NSMapEnumerator *enumerator,
    void **key, void **value)
```

Returns NO if enumerator has already iterated over all the elements in the table that enumerator is associated with. Otherwise, this function sets key and value to match the next key/value pair in the table, and returns YES.

NSAllMapTableKeys()

```
NSArray *NSAllMapTableKeys(NSMapTable *table)
```

Returns an array object containing all the keys in table. This function should be called only when the table keys are objects, not when they're any other type of pointer.

NSAllMapTableValues()

```
NSArray *NSAllMapTableValues(NSMapTable *table)
```

Returns an array object containing all the values in table. This function should be called only when the table values are objects, not when they're any other type of pointer.

Add or Remove an Item

NSMapInsert()

```
void NSMapInsert(NSMapTable *table, const void *key,
    const void *value)
```

Inserts key and value into table. If key matches a key already in the table, value is retained and the previous value is released, using the retain and release call-back functions that were specified when the table was created. Raises NSInvalidArgumentException if key is equal to the notAKeyMarker field of the table's NSMapTableKeyCallBacks structure. See also NSMapInsertIfAbsent(), NSMapInsertIfAbsent()).

NSMapInsertIfAbsent()

```
void *NSMapInsertIfAbsent(NSMapTable *table, const void *key,
    const void *value)
```

If key matches a key already in table, this function returns the pre-existing key; otherwise, it adds key and value to the table and returns NULL. Raises NSInvalidArgumentException if key is equal to the notAKeyMarker field of the table's NSMapTableKeyCallBacks structure.

NSMapInsertKnownAbsent()

```
void NSMapInsertKnownAbsent(NSMapTable *table, const void *key,
    const void *value)
```

Inserts key (which must not be notAKeyMarker) and value into table. Unike NSMapInsert(), this function raises NSInvalidArgumentException if table already includes a key that matches key.

NSMapRemove()

```
void NSMapRemove(NSMapTable *table, const void *key)
```

If key matches a key already in table, this function releases the pre-existing key and its corresponding value.

NSStringFromMapTable()

```
NSString *NSStringFromMapTable(NSMapTable *table)
```

Returns a string describing the map table's contents. The function iterates over the table's key/value pairs, and for each one appends the string "a = b;\n", where a and b are the key and value strings returned by the corresponding describe() call-back functions. If NULL was specified for the call-back function, a and b are the key and value pointers, expressed as hexadecimal numbers.

Miscellaneous Functions

Get Information about a User

NSUserName()

```
NSString *NSUserName(void)
```

Returns the user's name. See also `NSHomeDirectory()`.

NSHomeDirectory()

```
NSString *NSHomeDirectory(void)
```

Returns the user's home directory.

NSHomeDirectoryForUser()

```
NSString *NSHomeDirectoryForUser(NSString * userName)
```

Returns the home directory for the specified `userName`. See also `NSHomeDirectory()`.

Log an Error Message

NSLog()

```
void NSLog(NSString *format, ...)
```

Writes to `stderr` an error message of the form: "time processName processID format". The `format` argument to `NSLog()` is a `printf()`-style format string, followed by an arbitrary number of arguments that match conversion specifications (such as `%s` or `%d`) in the format string. You can pass an object in the list of arguments by specifying `%@` in the format string—this conversion specification gets replaced by the string that the object's `description` method returns. See also `NSLogv()`.

NSLogv()

```
void NSLogv(NSString *format, va_list args)
```

Like `NSLog()`, but the arguments to the format string are passed in a single `va_list`, in the manner of `vprintf()`.

Get Localized Versions of Strings

NSLocalizedString()

```
NSString *NSLocalizedString(NSString *key, NSString *comment)
```

Returns a localized version of the string designated by `key`. The default string table (`Localizable.strings`) in the main bundle is searched for `key`. `comment` is ignored, but can provide information for translators. See also `NSLocalizedStringFromTable()`.

NSLocalizedStringFromTable()

```
NSString *NSLocalizedStringFromTable(NSString *key,
    NSString *tableName, NSString *comment)
```

Like `NSLocalizedString()`, but searches the specified table. See also `NSLocalizedStringFromTableInBundle()`.

NSLocalizedStringFromTableInBundle()

```
NSString *NSLocalizedStringFromTableInBundle(NSString *key,
    NSString *tableName, NSBundle *aBundle, NSString *comment)
```

Like `NSLocalizedStringFromTable()`, but uses the specified bundle instead of the application's main bundle. See also `NSLocalizedString()`.

Convert to and from a String

NSClassFromString()

```
Class NSClassFromString(NSString *aClassName)
```

Returns the class object named by `aClassName`, or `nil` if none by this name is currently loaded.

NSSelectorFromString()

```
SEL NSSelectorFromString(NSString *aSelectorName)
```

Returns the selector named by `aSelectorName`, or zero if none by this name exists.

NSStringFromClass()

```
NSString *NSStringFromClass(Class aClass)
```

Returns an `NSString` object containing the name of `aClass`.

NSStringFromSelector()

```
NSString *NSStringFromSelector(SEL aSelector)
```

Returns an `NSString` object containing the name of `aSelector`.

Get an Objective C Type's Size and Alignment

NSGetSizeAndAlignment()

```
const char *NSGetSizeAndAlignment (const char *typePtr,
    unsigned int *sizep, unsigned int *alignp)
```

Returns the size and alignment of the Objective C type that `typePtr` points to in `sizep` and `alignp`. Returns `typePtr`. See also `NSMethodSignature`, `NSConnection`.

Types and Constants 8

Bundle Notification

```
NSString *NSBundleDidLoadNotification
```

After a bundle dynamically loads its code, the bundle sends out this notification. `NSBundleDidLoadNotification`'s user information dictionary contains an array of strings which are the names of the classes loaded. The key for this dictionary entry is `@"NSLoadedClasses"`.

Exception Handling

Exception Handler

```
typedef struct _NSHandler NSHandler;
```

Exception handler information.

Uncaught Exception Handler

```
typedef volatile void
    NSUncaughtExceptionHandler
    (NSException *exception);
```

Registers an uncaught exception handler.

Inconsistent Archive Exception

```
NSString *NSInconsistentArchiveException;
```

Consistency error in archive file.

Generic Exception

```
NSString *NSGenericException;
```

General programming error.

Internal Inconsistency Exception

```
NSString *NSInternalInconsistencyException;
```

Some item that should be invariant changed.

Invalid Argument Exception

```
NSString *NSInvalidArgumentException;
```

Invalid argument.

Malloc Exception

```
NSString *NSMallocException;
```

No memory left to allocate.

Range Exception

```
NSString *NSRangeException;
```

Attempt to access an element beyond the limit of an array or similar structure.

Character Conversion Exception

```
NSString *NSCharacterConversionException
```

Raised when conversion to a C string fails.

Geometry

NSPoint

```
typedef struct _NSPoint {
    float x;
    float y;
} NSPoint;
```

Graphical point definition.

NSSize

```
typedef struct _NSSize {
    float width;
    float height;
} NSSize;
```

Rectangle sizes.

NSRect

```
typedef struct _NSRect {
    NSPoint origin;
    NSSize size;
} NSRect;
```

Rectangle size and origin.

Rectangle Sides

```
typedef enum _NSRectEdge {
    NSMinXEdge,
    NSMinYEdge,
    NSMaxXEdge,
    NSMaxYEdge
} NSRectEdge;
```

Rectangle sides.

Zero Point

```
const NSPoint NSZeroPoint;
```

A zero point.

Zero-Sized Rectangle

```
const NSRect NSZeroRect;
```

A rectangle with zero size and origin.

Zero Size

```
const NSSize NSZeroSize;
```

A size with zero height and width.

Hash Table

Hash Enumerator

```
typedef struct NSHashEnumerator;
```

Private type for enumerating.

Hash Table

```
typedef struct _NSHashTable NSHashTable;
```

Hash table type.

Hash Table Call Backs

```
typedef struct {
    unsigned (*hash)(NSHashTable *table,
                           const void *anObject);
    BOOL (*isEqual)(NSHashTable *table,
                           const void *anObject,
                           const void *anObject);
    void (*retain)(NSHashTable *table,
                           const void *anObject);
    void (*release)(NSHashTable *table,
                           void *anObject);
```

```
    NSString *(*describe)(NSHashTable *table,
                          const void *anObject);
} NSHashTableCallBacks;
```

Describes callback functions. `hash` is a hashing function. Note that elements with equal values must have equal hash function values. `isEqual` is a comparison function. `retain` is a retaining function call when adding elements to the table. `release` is a releasing function called when a data element is removed from the table. `describe` is a description function.

The following constants describe specific hash table callbacks. See `NSFoundationGlobals.m` for more information.

```
const NSHashTableCallBacks NSIntHashCallBacks;
```

For sets of pointer-sized or smaller quantities.

```
const NSHashTableCallBacks NSNonOwnedPointerHashCallBacks;
```

For sets of pointers hashed by address.

```
const NSHashTableCallBacks NSNonRetainedObjectHashCallBacks;
```

For sets of objects without retaining and releasing.

```
const NSHashTableCallBacks NSObjectHashCallBacks;
```

For sets of objects; similar to `NSSet`.

```
const NSHashTableCallBacks NSOwnedPointerHashCallBacks;
```

For sets of pointers with transfer of ownership upon insertion.

```
const NSHashTableCallBacks NSPointerToStructHashCallBacks;
```

For sets of pointers to `struct`s when the first field of the `struct` is the size of an `int`.

```
const NSHashTableCallBacks NSOwnedObjectIdentityHashCallBacks;
```

For sets that own the objects but use pointer comparison.

Map Table

Map Enumerator

```
typedef struct NSMapEnumerator;
```

Private type for enumerating.

Map Table

```
typedef struct _NSMapTable NSMapTable;
```

Map table type.

Map Table Key Callbacks

```
typedef struct {
    unsigned (*hash)(NSMapTable *table,
                     const void *anObject);
    BOOL (*isEqual)(NSMapTable *table,
                     const void *anObject,
                     const void *anObject);
    void (*retain)(NSMapTable *table,
                     const void *anObject);
    void (*release)(NSMapTable *table,
                     void *anObject);
    NSString *(*describe)(NSMapTable *table,
                     const void *anObject);
    const void *notAKeyMarker;
} NSMapTableKeyCallBacks;
```

Callback functions for a key. `hash` is a hashing function. Note that elements with equal values must have equal hash function values. `isEqual` is a comparison function. `retain` is a retaining function call when adding elements to the table. `release` is a releasing function called when a data element is removed from the table. `describe` is a description function. `notAKeyMarker` is a quantity that is not a key to the hash table.

Map Table Value Callbacks

```
typedef struct {
    void (*retain)(NSMapTable *table,
                        const void *anObject);
    void (*release)(NSMapTable *table,
                        void *anObject);
    NSString *(*describe)(NSMapTable *table,
                        const void *anObject);
 } NSMapTableValueCallBacks;
```

Callback functions for a value. `retain` is a retaining function call when adding elements to the table. `release` is a releasing function called when a data element is removed from the table. `describe` is a description function.

Not An Integer Map Key

```
#define NSNotAnIntMapKey;
```

Quantity that is never a map key.

Not A Pointer Map Key

```
#define NSNotAPointerMapKey;
```

Quantity that is never a map key.

Pointer-Sized Map Key Callbacks

```
const NSMapTableKeyCallBacks NSIntMapKeyCallBacks;
```

For keys that are pointer-sized or smaller quantities.

Pointer-Sized MapValue Callbacks

```
const NSMapTableValueCallBacks NSIntMapValueCallBacks;
```

For values that are pointer-sized quantities.

Non-Owned Pointer Map Key Callbacks

```
const NSMapTableKeyCallBacks NSNonOwnedPointerMapKeyCallBacks;
```

For keys that are pointers not freed.

Non-Owned Pointer Map Value Callbacks

```
const NSMapTableValueCallBacks NSNonOwnedPointerMapValueCallBacks;
```

For values that are owned pointers.

Non-Owned Pointer Or Null Map Key Callbacks

```
const NSMapTableKeyCallBacks
    NSNonOwnedPointerOrNullMapKeyCallBacks;
```

For keys that are pointers not freed, or `NULL`.

Non-Retained Object Map Key Callbacks

```
const NSMapTableKeyCallBacks
   NSNonRetainedObjectMapKeyCallBacks;
```

For sets of objects without retaining and releasing.

Object Map Key Callbacks

```
const NSMapTableKeyCallBacks NSObjectMapKeyCallBacks;
```

For keys that are objects.

Object Map Value Callbacks

```
const NSMapTableValueCallBacks NSObjectMapValueCallBacks;
```

For values that are objects.

Owned Pointer Map Key Callbacks

```
const NSMapTableKeyCallBacks NSOwnedPointerMapKeyCallBacks;
```

For keys that are pointers with transfer of ownership upon insertion.

Owned Pointer Map Value Callbacks

```
const NSMapTableValueCallBacks NSOwnedPointerMapValueCallBacks;
```

For values that are owned pointers.

Non-Retained Object Map Value Callbacks

```
const NSMapTableValueCallBacks
    NSNonRetainedObjectMapValueCallBacks;
```

For values which are objects that should not be retained.

Notification Queue

Posting Style

```
typedef enum
    NSPostWhenIdle,
    NSPostASAP,
    NSPostNow
} NSPostingStyle;
```

`NSPostWhenIdle` means to post the notification when the run loop is idle. `NSPostASAP` means to post the notification as soon as possible. And `NSPostNow` means to post the notification immediately.

Notification Coalescing

```
typedef enum {
    NSNotificationNoCoalescing,
    NSNotificationCoalescingOnName,
    NSNotificationCoalescingOnSender,
} NSNotificationCoalescing;
```

`NSNotificationNoCoalescing` means not to coalesce similar notifications in the queue. `NSNotificationCoalescingOnName` means to coalesce notifications in the queue matching name. And `NSNotificationCoalescingOnSender` means to coalesce notifications in the queue matching sender.

Run Loop

Connection Reply Mode

```
NSString *NSConnectionReplyMode;
```

NSRunLoop mode in which Distributed Object system seeks replies.

Default Run Loop Mode

```
NSString *NSDefaultRunLoopMode;
```

Common NSRunLoop mode.

Searching

Comparison Result

```
typedef enum _NSComparisonResult {
    NSOrderedAscending,
    NSOrderedSame,
    NSOrderedDescending
} NSComparisonResult;
```

Ordered comparison results.

Anchored Search

```
enum {
    NSCaseInsensitiveSearch,
    NSLiteralSearch,
    NSBackwardsSearch,
    NSAnchoredSearch
};
```

Flags passed to various search methods.

Not Found

```
enum {NSNotFound};
```

Indicates an item not found.

String

String Encodings

```
typedef unsigned NSStringEncoding;
```

Known string encodings.

Unicode String Encodings

```
enum {
    NSASCIIStringEncoding,
    NSNEXTSTEPStringEncoding,
    NSJapaneseEUCStringEncoding,
    NSUTF8StringEncoding,
    NSISOLatin1StringEncoding ,
    NSSymbolStringEncoding ,
    NSNonLossyASCIIStringEncoding,
    NSShiftJISStringEncoding,
    NSISOLatin2StringEncoding,
    NSUnicodeStringEncoding
};
```

Known Unicode string encodings.

OpenStep Unicode Base

```
enum _NSOpenStepUnicodeReservedBase {
    NSOpenStepUnicodeReservedBase
};
```

Base for Unicode characters.

Maximum String Length

```
NSMaximumStringLength
```

Maximum string length, defined as `INT_MAX-1`.

8

Threads

Thread Priorities

```
typedef enum {
    NSInteractiveThreadPriority,
    NSBackgroundThreadPriority,
    NSLowThreadPriority
} NSThreadPriority;
```

Notifications

```
NSString *NSBecomingMultiThreaded;
NSString *NSThreadExiting;
```

User Defaults

```
NSString *NSArgumentDomain;
//For defaults parsed from the application’s arguments.
NSString *NSGlobalDomain;
//For defaults seen by all applications.
NSString *NSRegistrationDomain;
//For registered defaults.
NSString *NSUserDefaultsChanged;
//Public notification.
NSString *NSWeekDayNameArray;
//Keys for language-dependent information.
NSString *NSShortWeekDayNameArray;
NSString *NSMonthNameArray;
NSString *NSShortMonthNameArray;
NSString *NSTimeFormatString;
NSString *NSDateFormatString;
NSString *NSTimeDateFormatString;
NSString *NSShortTimeDateFormatString;
NSString *NSCurrencySymbol;
NSString *NSDecimalSeparator;
NSString *NSThousandsSeparator;
NSString *NSInternationalCurrencyString;
NSString *NSCurrencyString;
NSString *NSDecimalDigits;
NSString *NSAMPMDesignation;
```

Miscellaneous

NSArgumentInfo

```
typedef struct {
    int         offset;
    int         size;
    char        *type;
} NSArgumentInfo;
```

Specifies the layout of arguments used in invocations. See the `NSCoder` class description for a list of argument types.

NSRange

```
typedef struct _NSRange {
    unsigned int location;
    unsigned int length;
} NSRange;
```

Specifies a range of items in arrays, strings, and so on.

NSTimeInterval

```
typedef double NSTimeInterval;
```

Time interval difference between two dates.

NSZone

```
typedef struct _NSZone NSZone;
```

Large region allocation. See also `NSCreateZone()`(Foundation Kit Functions chapter).

Part 3 — Display PostScript

Classes 9

The single class listed here and the protocol in the following section constitute OpenStep's object-oriented interface to the Display PostScript System. Many of the argument and return types that appear below (specifically, those having a "DPS" prefix) are not described in this document. Rather, they are detailed in the specification for the Display PostScript System itself, as found in the *Display PostScript System, Client Library Reference Manual*, by Adobe Systems Incorporated.

9

NSDPSContext

Inherits From:	NSObject
Conforms To:	NSObject (NSObject)
Declared In:	DPSClient / NSDPSContext.h

Class Description

The `NSDPSContext` class is the programmatic interface to objects that represent Display PostScript System *contexts*. A context can be thought of as a *destination* to which PostScript code is sent for execution. Each Display PostScript context contains its own complete PostScript environment including its own local VM (PostScript Virtual Memory). Every context has its own set of stacks, including an operand stack, graphics state stack, dictionary stack, and execution stack. Every context also contains a `FontDirectory` which is local to that context, plus a `SharedFontDirectory` that is shared across all contexts. There are three built-in dictionaries in the dictionary stack. From top to bottom, they are `userdict`, `globaldict`, and `systemdict`. `userdict` is private to the context, while `globaldict` and `systemdict` are shared by all contexts. `globaldict` is a modifiable dictionary containing information common to all contexts. `systemdict` is a read-only dictionary containing all the PostScript operators.

At any time there is the notion of the *current context*. The current context for the current thread may be set using `setCurrentContext:`.

`NSDPSContext` objects by default write their output to a specified *data* destination. This is used for printing, FAXing, and for generation of saved EPS (Encapsulated PostScript) code. The means to create contexts that interact with displays are platform-specific. The `NSApplication` object creates a context by default.

NSDPSContext Objects and Display PostScript System Context Records

When an `NSDPSContext` object is created, it creates and manages a `DPSContext` record. Programmers familiar with the client side C function interface to the Display PostScript System can access the `DPSContext` record by sending a `context` message to an `NSDPSContext` object. You can then operate on this context record using any of the functions or single operator

functions defined in the Display PostScript System client library. Conversely, you can create an `NSDPSContext` object from a `DPSContext` record with the `DPSContextObject()` function, as defined in "Client Library Functions". You can then work with the created `NSDPSContext` object using any of the methods described here.

General Exception Conditions

A variety of exceptions can be raised from `NSDPSContext`. In most cases, exceptions are raised because of errors returned from the Display PostScript Server. Exceptions are listed under "Types and Constants." Also see the *Display PostScript System, Client Library Reference Manual*, by Adobe Systems Incorporated, for more details on Display PostScript System error names and their possible causes.

Method Types

Activity	ClassMethod
Initializing a Context	– initWithMutableData:forDebugging: languageEncoding:nameEcoding:textProc: errorProc:
Testing the Drawing Destination	– isDrawingToScreen
Accessing Context Data	– mutableData
Setting and Identifying the Current Context	+ currentContext – setCurrentContext: – DPSContext
Controlling the Context	– flush – interruptExecution – notifyObjectWhenFinishedExecuting: – resetCommunication – wait
Managing Returned Text and Errors	+ stringForDPSError: – errorProc – setErrorProc: – setTextProc: – textProc
Sending Raw Data	– printFormat: – printFormat:arguments: – writeData: – writePostScriptWithLanguageEncodingConversion:
Managing Binary Object Sequences	– awaitReturnValues – writeBOSArray:count:ofType: – writeBOSNumString:length:ofType:scale: – writeBOSString:length: – writeBinaryObjectSequence:length: – updateNameMap
Managing Chained Sequences	– chainChildContext: – childContext – parentContext – unchainContext
Debugging Aids	+ areAllContextsOutputTraced + areAllContextsSynchronized + setAllContextsOutputTraced: + setAllContextsSynchronized: – isOutputTraced – isSynchronized – setOutputTraced: – setSynchronized:

Class Methods

areAllContextsOutputTraced

```
+ (BOOL)areAllContextsOutputTraced
```

Returns YES if the data flowing between the application's contexts and their destinations is copied to diagnostic output.

areAllContextsSynchronized

```
+ (BOOL)areAllContextsSynchronized
```

Returns YES if all NSPDSContext objects invoke the wait method after sending each batch of output.

currentContext

```
+ (NSDPSContext *)currentContext
```

Returns the current context of the current thread. See also setCurrentContext:.

setAllContextsOutputTraced:

```
+ (void)setAllContextsOutputTraced:(BOOL)flag
```

When flag is YES, causes the data (PostScript code, return values, etc.) flowing between the all the application's contexts and their destinations to be copied to diagnostic output. See also areAllContextsOutputTraced, isOutputTraced.

setAllContextsSynchronized:

```
+ (void)setAllContextsSynchronized:(BOOL)flag
```

When flag is YES, causes the wait method to be invoked each time an NSDPSContext object sends a batch of output to its destination. See also areAllContextsSynchronized, setSynchronized:, isSynchronized.

setCurrentContext:

```
+ (void)setCurrentContext:(NSDPSContext *)context
```

Installs `context` as the current context of the current thread. See also `currentContext`.

stringForDPSError:

```
+ (NSString *)stringForDPSError:(const DPSBinObjSeqRec *)error
```

Returns a string representation of `error`.

Instance Methods

awaitReturnValues

```
- (void)awaitReturnValues
```

Waits for all return values from the result table.

chainChildContext:

```
- (void)chainChildContext:(NSDPSContext *)child
```

Links `child` (and all of it's children) to the receiver as its chained context, a context that receives a copy of all PostScript code sent to the receiver.

childContext

```
- (NSDPSContext *)childContext
```

Returns the receiver's child context, or `nil` if none exists. See also `parentContext`.

DPSContext

```
- (DPSContext)DPSContext
```

Returns the corresponding `DPScontext`.

errorProc

```
- (DPSErrorProc)errorProc
```

Returns the context's error callback function. See also `setErrorProc:`.

flush

```
- (void)flush
```

Forces any buffered data to be sent to its destination.

initWithMutableData:forDebugging: languageEncoding:nameEcoding:textProc: errorProc:

```
- initWithMutableData:(NSMutableData *)data
    forDebugging:(BOOL)debug
    languageEncoding:(DPSProgramEncoding)langEnc
    nameEcoding:(DPSNameEncoding)nameEnc
    textProc:(DPSTextProc)tProc errorProc:(DPSErrorProc)errorProc
```

Initializes a newly allocated `NSDPSContext` that writes its output to `data` using the language and name encodings specified by `langEnc` and `nameEnc`. The callback functions `tProc` and `errorProc` handle text and errors generated by the context. If `debug` is `YES`, the output is given in human-readable form in which large structures (such as images) may be represented by comments.

interruptExecution

```
- (void)interruptExecution
```

Interrupts execution in the receiver's context.

isDrawingToScreen

```
- (BOOL)isDrawingToScreen
```

Returns `YES` if the drawing destination is the screen.

isOutputTraced

```
- (BOOL)isOutputTraced
```

Returns YES if the data flowing between the application's single context and its destination is copied to diagnostic output. See also `setOutputTraced:`.

isSynchronized

```
- (BOOL)isSynchronized
```

Returns whether the `wait` method is invoked each time the receiver sends a batch of output to the server.

mutableData

```
- (NSMutableData *)mutableData
```

Returns the receiver's data object.

notifyObjectWhenFinishedExecuting:

```
- (void)notifyObjectWhenFinishedExecuting:
    (id <NSDPSContextNotification>)object
```

Registers `object` to receive a `contextFinishedExecuting:` message when the `NSDPSContext`'s destination is ready to receive more input.

parentContext

```
- (NSDPSContext *)parentContext
```

Returns the receiver's parent context, or `nil` if none exists. See also `childContext`.

printFormat:

```
- (void)printFormat:(NSString *)format,...
```

Constructs a string from `format` and following string objects (in the manner of `printf()`) and sends it to the context's destination. See also `printFormat:arguments:`.

printFormat:arguments:

```
- (void)printFormat:(NSString *)format arguments:(va_list)argList
```

Constructs a string from `format` and `argList` (in the manner of `vprintf()`) and sends it to the context's destination. See also `printFormat:`.

resetCommunication

```
- (void)resetCommunication
```

Discards any data that hasn't already been sent to its destination.

setErrorProc:

```
- (void)setErrorProc:(DPSErrorProc)proc
```

Sets the context's error callback function to `proc`. See also `errorProc`.

setOutputTraced:

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

When `flag` is `YES`, causes the data (PostScript code, return values, etc.) flowing between the application's single context and the Display PostScript server to be copied to diagnostic output. See also `isOutputTraced`.

setSynchronized:

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

Sets whether the `wait` method is invoked each time the receiver sends a batch of output to its destination.

setTextProc:

```
- (void)setTextProc:(DPSTextProc)proc
```

Sets the context's text callback function to `proc`.

textProc

```
- (DPSTextProc)textProc
```

Returns the context's text callback function.

unchainContext

```
- (void)unchainContext
```

Unlinks the child context (and all of it's children) from the receiver's list of chained contexts.

updateNameMap

```
- (void)updateNameMap
```

Updates the context's name map from the client library's name map.

wait

```
- (void)wait
```

Waits until the `NSDPSContext`'s destination is ready to receive more input.

writeBOSArray:count:ofType:

```
- (void)writeBOSArray:(const void *)data count:(unsigned int)items
    ofType:(DPSDefinedType)type
```

Write an array to the context's destination as part of a a binary object sequence. The array is taken from `data` and consists of `items` items of type `type`.

writeBOSNumString:length:ofType:scale:

```
- (void)writeBOSNumString:(const void *)data
    length:(unsigned int)count
    ofType:(DPSDefinedType)type scale:(int)scale
```

Write a number string to the context's destination as part of a binary object sequence. The string is taken from `data` as described by `count`, `type`, and `scale`.

writeBOSString:length:

```
- (void)writeBOSString:(const void *)data
    length:(unsigned int)bytes
```

Write a string to the context's destination as part of a binary object sequence. The string is taken from `bytes` (a count) of `data`.

writeBinaryObjectSequence:length:

```
- (void)writeBinaryObjectSequence:(const void *)data
    length:(unsigned int)bytes
```

Write a binary object sequence to the context's destination. The sequence consists of `bytes` (a count) of `data`.

writeData:

```
- (void)writeData:(NSData *)buf
```

Sends the PostScript data in `buf` to the context's destination.

writePostScriptWithLanguageEncodingConversion:

```
- (void)writePostScriptWithLanguageEncodingConversion:(NSData *)buf
```

Writes the PostScript data in `buf` to the context's destination. The data, formatted as plain text, encoded tokens, or a binary object sequence, is converted as necessary depending on the language encoding of the receiving context.

9

Protocols 10

NSDPSContextNotification

Adopted by:	No OpenStep classes
Declared In:	DPSClient / NSDPSContext.h

Protocol Description

The `NSDPSContextNotification` protocol supplies information about the execution status of a sequence of PostScript commands previously sent to the Display PostScript server.

Instance Methods

`contextFinishedExecuting:`

```
- (void)contextFinishedExecuting:(NSDPSContext *)context
```

Notifies the receiver that the context has finished executing a batch of PostScript commands. See `notifyObjectWhenFinishedExecuting:` (`NSDPSContext`).

10

Operators 11

This chapter describes the operators found in OpenStep, but not in the standard PostScript language. The *PostScript Language Reference Manual* (the Red Book), Second Edition, by Adobe Systems Incorporated, provides the specifications for standard PostScript and Display PostScript operators.

The following operator descriptions are in the format used by the *PostScript Language Reference Manual*, and the Adobe publication *Programming the Display PostScript System with X* (the Orange Book). For example:

operand$_1$ ... *operand*$_n$ **operator** *result*$_1$ ... *result*$_m$

Compositing Operators

composite

src$_x$ *src*$_y$ *width height srcgstate dest*$_x$ *dest*$_y$ *op* **composite** –

Performs the compositing operation specified by *op* between pairs of pixels in two images, a source and a destination. The source pixels are in the Drawable referred to by the srcgstate graphics state, and the destination pixels are in the Drawable specified by the current graphics state. If srcgstate is null, the current graphics state is assumed.

The rectangle specified by src_x, src_y, width, and height defines the source image. The outline of the rectangle may cross pixel boundaries due to fractional coordinates, scaling, or rotated axes. The pixels included in the source are all those that the outline of the rectangle encloses or enters.

The destination image has the same size, shape, and orientation as the source; $dest_x$ and $dest_y$ give destination's location image compared to the source. Even if the two graphic states have different orientations, the images will not have different orientations; composite will not rotate images.

Both images are clipped to the frame rectangles of the respective Drawables. The destination image is further clipped to the clipping path of the current graphics state. The result of a composite operation replaces the destination image.

op specifies the compositing operation. The color of each destination image pixel (alpha value) after the operation, dst' (dstA') is given by

*dst' = src * Fs(srcA, dstA, op) + dst * Fd(srcA, dstA, op)*

*dstA' = srcA * Fs(srcA, dstA, op) + dstA * Fd(srcA, dstA, op)*

where *src* and *srcA* are the source color and alpha values, *dst* and *dstA* are the destination color and alpha values, and *Fs* and *Fd* are the functions given in the following table. The choices for *op* are also given in the following table.

Table 11-1 Composite Operation and Compositing Equation Factors

Op	Fs	Fd
Clear	0	0
Copy	1	0
Sover	1	1 - srcA
Sin	dstA	0
Sout	1 - dstA	0
Satop	dstA	1 - srcA
Dover	1 - dstA	1
Din	0	srcA
Dout	0	1 - srcA
Datop	1 - dstA	srcA

Table 11-1 Composite Operation and Compositing Equation Factors *(Continued)*

Op	Fs	Fd
Xor	1 - dstA	1 - srcA
PlusD[1]	N / A	N / A
PlusL[2]	1	1
dissolve[3]	delta	1 - delta
Highlight[4]	N / A	N / A

1. **PlusD** does not follow the general equation. The equation is dst' = (1-dst) + (1-src); If the result is less than 0 (black) then the result is 0.
2. For **PlusL** the addition saturates. That is if (src + dst > white) the result is white.
3. For **dissolve**, Fa and Fs have another parameter: the delta operand to the dissolve operator.
4. **Highlight** doesn't follow the general equation. It turns white pixels light gray (2/3) and light gray pixels white. Pixels of other colors are unaffected. Alpha values are unaffected. Highlight is a valid op only for the **compositerect** operator.

compositerect

$dest_x$ $dest_y$ width height op **compositerect** –

Composites rectangle of current color and coverage with image in current graphics state.

In general, this operator is the same as the composite operator except that there is no real source image. The destination is in the current graphics state; *$dest_x$*, *$dest_y$*, width, and height describe the destination image in that graphics state's current coordinate system. The effect on the destination is as if there were a source image filled with the color and coverage specified by the graphics state's current color and coverage parameters. *op* has the same meaning as the *op* operand of the composite operator; however, one additional operation, Highlight, is allowed.

Highlight turns every white pixel in the destination rectangle to light gray and every light gray pixel to white, regardless of the pixel's coverage value. Light gray is defined as 2/3. Repeating the same operation reverses the effect. (On monochrome displays, Highlight inverts each pixel, white becomes black, black becomes white.)

Note that the Highlight operation doesn't change the value of a pixel's coverage component. To ensure that the pixel's color and coverage combination remains valid, Highlight operations should be temporary and should be reversed before any further compositing.

For this operation, the pixels included in the destination are those that the outline of the specified rectangle encloses or enters. The destination image is clipped to the frame rectangle and clipping path of the window in the current graphics state.

dissolve

src_x src_y width height srcgstate $dest_x$ $dest_y$ delta **dissolve** –

Dissolves between area of window referred to by srcgstate and equal area of window referred to by the current graphics state. The effect of this operation is a blending of a source and a destination image. The first seven arguments choose source and destination pixels as they do for composite. The exact fraction of the blend is specified by delta, which is a floating-point number between 0.0 and 1.0; the resulting image is:

delta * source + (1- delta) * destination

If srcgstate is null, the current graphics state is assumed. The values of the composite ops are available for applications in the PostScript systemdict. The definitions are as follows:

- /Clear 0 def
- /Copy 1 def
- /Sover 2 def
- /Sin 3 def
- /Sout 4 def
- /Satop 5 def
- /Dover 6 def
- /Din 7 def
- /Dout 8 def
- /Datop 9 def
- /Xor 10 def
- /PlusD 11 def
- /Highlight 12 def
- /PlusL 13 def

Graphics State Operators

setalpha

coverage **setalpha** –

Sets the coverage parameter in the current graphics state to *coverage*. *coverage* should be a number between 0 and 1, with 0 corresponding to transparent, 1 corresponding to opaque, and intermediate values corresponding to partial coverage. The default value is 1. This sets how much background shows through when compositing images. If the coverage value given is less than 0, the coverage parameter is set to 0. If the value is greater than 1, the coverage parameter is set to 1.

The coverage value affects the color painted by PostScript marking operations. The current color is pre-multiplied by the alpha value before rendering. This multiplication occurs after the current color has been transformed to the RGB color space.

currentalpha

– **currentalpha** *coverage*

Returns the coverage parameter of the current graphics state.

11

Client Library Functions 12

The Display PostScript Client Library is composed of system-dependent and a system-independent parts. The *Display PostScript System, Client Library Reference Manual*, by Adobe Systems, Incorporated, provides the specification for the system-independent portion of this library.

Functions that are part of OpenStep's system-dependent part of the Display PostScript Client Library are listed here.

PostScript Execution Context Functions

`DPSContextObject()`

```
NSDPSContext *DPSContextObject(DPSContext ctxt)
```

Converts a `DPSContext` to an `NSDPSContext` object.

12

Communication with the Display PostScript Server

Send a PostScript User Path to the Display PostScript Server

These functions are used to send a user path, plus one other `action`, to the Display PostScript Server. In the …`WithMatrix` forms of these functions, the `matrix` argument is the optional matrix argument used by the `ustroke`, `inustroke`, and `ustrokepath` operators. The `matrix` argument may be `NULL`, in which case it is ignored.

PSDoUserPath()

```
void PSDoUserPath(const void *coords, int numCoords,
    DPSNumberFormat numType, const DPSUserPathOp *ops, int numOps,
    const void *bbox, DPSUserPathAction action)
```

PSDoUserPathWithMatrix()

```
void PSDoUserPathWithMatrix(void *coords, int numCoords,
    DPSNumberFormat numType, unsigned char *ops, int numOps,
    void *bbox, DPSUserPathAction action, float matrix[6])
```

PSDoUserPathWithMatrix()

```
void PSDoUserPathWithMatrix(void *coords, int numCoords,
    DPSNumberFormat numType, unsigned char *ops, int numOps,
    void *bbox, DPSUserPathAction action, float matrix[6])
```

DPSDoUserPath()

```
void DPSDoUserPath(DPSContext context, const void *coords,
    int numCoords, DPSNumberFormat numType,
    const DPSUserPathOp *ops,int numOps, const void *bbox,
    DPSUserPathAction action)
```

DPSDoUserPathWithMatrix()

```
void DPSDoUserPathWithMatrix(DPSContext context, void *coords,
    int numCoords, DPSNumberFormat numType, unsigned char *ops,
    int numOps, void *bbox, DPSUserPathAction action,
    float matrix[6])
```

Send PostScript Code to the Display PostScript Server

PSFlush()

```
void PSFlush(void)
```

PSWait()

```
void PSWait(void)
```

12

Single-Operator Functions 13

Single-operator functions provide a C language interface to the individual operators of the PostScript language. The specification for a single-operator function is identical to that of the PostScript operator it represents. The *PostScript Language Reference Manual, Second Edition*, by Adobe Systems Incorporated, provides the specifications of all standard PostScript operators. Also refer to the *Display PostScript System, Client Library Reference Manual*, by Adobe Systems Incorporated. Listed below are single-operator functions that correspond to operators found in OpenStep but not in the standard implementation of the PostScript language.

These functions have either a “PS” or a “DPS” prefix. For every single-operator function with a “PS” prefix, there’s a corresponding single-operator function with a “DPS” prefix. The PS and DPS functions are identical except that DPS functions take an additional (first) argument that represents the PostScript execution context.

Besides using standard C language types, some single-operator functions use `userobject`—an `int` that refers to the value returned by `DPSDefineUserObject()`.

In the function descriptions below, `x` and `y` refer to the origin of *source* rectangles, and `w` and `h` refer to the width and height of the source rectangles. `gstateNum` refers to the graphics state (gstate) of the source rectangle. `dx` and `dy` refer to the origin of the *destination* for the compositing or dissolving operation. `op` refers to the specific compositing operation. `a` or `alpha` refers to the coverage component used for compositing operations.

13

"PS" Prefix Functions

PScomposite()

```
void PScomposite(float x, float y, float w, float h,
    int gstateNum, float dx, float dy, int op)
```

PScompositerect()

```
void PScompositerect(float x, float y, float w, float h, int op)
```

PScurrentalpha()

```
void PScurrentalpha(float *alpha)
```

PSdissolve()

```
void PSdissolve(float x, float y, float w, float h,
    int gstateNum, float dx, float dy, float delta)
```

PSsetalpha()

```
void PSsetalpha(float a)
```

"DPS" Prefix Functions

DPScomposite()

```
void DPScomposite(DPSContext ctxt, float x, float y, float w,
    float h, int gstateNum, float dx, float dy, int op)
```

DPScompositerect()

```
void DPScompositerect(DPSContext ctxt, float dx, float dy,
    float w, float h, int op)
```

DPScurrentalpha()

```
void DPScurrentalpha(DPSContext ctxt, float *pcoverage)
```

DPSdissolve()

```
void DPSdissolve(DPSContext ctxt, float x, float y, float w, float h,
    int gstateNum, float dx, float dy, float delta)
```

DPSsetalpha()

```
void DPSsetalpha(DPSContext ctxt, float a)
```

13

Types and Constants 14

The Display PostScript Client Library is composed of system-dependent and a system-independent parts. The *Display PostScript System, Client Library Reference Manual*, by Adobe Systems, Incorporated, provides the specification for the system-independent portion of this library.

The defined types, enumeration constants, and global variables that are part of OpenStep's system-dependent part of the Display PostScript Client Library are listed here.

Defined Types

Number Format

```
typedef enum _DPSNumberFormat {
#ifdef __BIG_ENDIAN__
    dps_float = 48,
    dps_long = 0,
    dps_short = 32
#else
    dps_float = 48+128,
    dps_long = 0+128,
    dps_short = 32+128
#endif
} DPSNumberFormat;
```

Other permitted values are:

- For 32-bit fixed-point numbers, use `dps_long` plus the number of bits in the fractional part.
- For 16-bit fixed-point numbers, use `dps_short` plus the number of bits in the fractional part.

NSBackingStoreType

```
typedef enum _NSBackingStoreType {
    NSBackingStoreRetained,
    NSBackingStoreNonretained,
    NSBackingStoreBuffered
} NSBackingStoreType;
```

Backing store types.

Compositing Operations

```
typedef enum _NSCompositingOperation {
    NSCompositeClear,
    NSCompositeCopy,
    NSCompositeSourceOver,
    NSCompositeSourceIn,
    NSCompositeSourceOut,
    NSCompositeSourceAtop,
    NSCompositeDestinationOver,
    NSCompositeDestinationIn,
    NSCompositeDestinationOut,
    NSCompositeDestinationAtop,
    NSCompositeXOR,
    NSCompositePlusDarker,
    NSCompositeHighlight,
    NSCompositePlusLighter
} NSCompositingOperation;
```

NSWindowOrderingMode

```
typedef enum _NSWindowOrderingMode {
    NSWindowAbove,
    NSWindowBelow,
    NSWindowOut
} NSWindowOrderingMode;
```

User Path Operators

```
typedef unsigned char DPSUserPathOp;
enum {
    dps_setbbox,
    dps_moveto,
    dps_rmoveto,
    dps_lineto,
    dps_rlineto,
    dps_curveto,
    dps_rcurveto,
    dps_arc,
    dps_arcn,
    dps_arct,
    dps_closepath,
    dps_ucache
};
```

User path operators. These constants define the operator numbers used to construct the operator array parameter of `DPSDoUserPath`.

User Path Actions

```
typedef enum _DPSUserPathAction {
    dps_uappend,
    dps_ufill,
    dps_ueofill,
    ps_ustroke,
    dps_ustrokepath,
    dps_inufill,
    dps_inueofill,
    dps_inustroke,
    dps_def,
    dps_put
} DPSUserPathAction;
```

User path actions. These constants define the action of a `DPSDoUserPath`. In addition to the actions defined here, any other system name index may be used. See the PostScript Language Reference Manual, Second Edition, by Adobe Systems Incorporated, for a detailed list of system name indexes.

14

Enumerations

Alpha Values

```
enum {
    NSAlphaEqualToData,
    NSAlphaAlwaysOne
};
```

Null Object

```
enum {
    DPSNullObject
};
```

User object representing the PostScript null object.

Symbolic Constants

```
DPS_OPENSTEP_ERROR_BASE
```

Error code base.

Global Variables

DPS Exceptions

```
NSString *DPSPostscriptErrorException;
NSString *DPSNameTooLongException;
NSString *DPSResultTagCheckException;
NSString *DPSResultTypeCheckException;
NSString *DPSInvalidContextException;
NSString *DPSSelectException;
NSString *DPSConnectionClosedException;
NSString *DPSReadException;
NSString *DPSWriteException;
NSString *DPSInvalidFDException;
NSString *DPSInvalidTEException;
NSString *DPSInvalidPortException;
NSString *DPSOutOfMemoryException;
NSString *DPSCantConnectException;
```

14

Part 4 — SoundKit

Sound Classes 15

The Sound, SoundMeter, and SoundView classes comprise OpenStep's sound support. These classes are not part of the OpenStep specification.

Sound

Inherits From:	NSObject
Declared In:	soundkit/Sound.h

Class Description

Sound objects represent and manage sounds. A Sound object's sound can be recorded from a microphone, read from a soundfile or NSBundle resource, retrieved from the pasteboard, or created algorithmically. The Sound class also provides an application-wide name table that lets you identify and locate sounds by name.

Playback and recording are performed by background threads, allowing your application to proceed in parallel. You should only use a Sound object to play and record sounds in applications that have a running NSApp object present.

You can also edit a Sound object by adding and removing samples. To minimize data movement (and thus save time), an edited Sound may become fragmented; in other words, its sound data might become discontiguous in memory. While playback of a fragmented Sound object is transparent, it does incur some overhead. If you perform a number of edits you may want to return the Sound

to a contiguous state by sending it a `compactSamples` message before you play it. However, a large `Sound` may take a long time to compact, so a judicious and well-timed use of `compactSamples` is advised. Fragmented `Sound`s are automatically compacted before they're copied to a pasteboard (through the `writeToPasteboard:` method). Also, when you write a `Sound` to a soundfile, the data in the file will be compact regardless of the state of the object.

A `Sound` object contains a structure, named `SNDSoundStruct`, that contains and describes sound data. Here is what it looks like:

```
typedef struct {
    int magic;          /* Must be equal to SND_MAGIC */
    int dataLocation;   /* Offset or pointer to the raw data */
    int dataSize;       /* Number of raw data bytes */
    int dataFormat;     /* Data format code */
    int samplingRate;   /* Sampling rate */
    int channelCount;   /* Number of channels */
    char info[4];       /* Textual sound information */
} SNDSoundStruct;
```

This sound data format is also used as the `Sound` object's file format and pasteboard type.

- `SNDSoundStruct` consists of a header and two variable length quantities: textual information (`info`) and raw data (`dataLocation`). Here's a description of the contents:
- `magic` specifies the magic number used to determine the byte order of the data.
- `dataLocation` specifies an offset from the beginning of the `SNDSoundStruct` to `info`'s end. This raw data always starts where textual `info` (described below) ends.
- `dataSize` is the length of the raw data in bytes.
- `dataFormat` specifies what the data actually means (for example, sample data, dsp core structure), and is one of the following constants:
  - `SND_FORMAT_UNSPECIFIED`
  - `SND_FORMAT_MULAW_8`
  - `SND_FORMAT_LINEAR_8`
  - `SND_FORMAT_LINEAR_16`
  - `SND_FORMAT_LINEAR_24`
  - `SND_FORMAT_LINEAR_32`
  - `SND_FORMAT_FLOAT`

- SND_FORMAT_DOUBLE
- SND_FORMAT_INDIRECT
- SND_FORMAT_NESTED
- SND_FORMAT_DSP_CORE
- SND_FORMAT_DSP_DATA_8
- SND_FORMAT_DSP_DATA_16
- SND_FORMAT_DSP_DATA_24
- SND_FORMAT_DSP_DATA_32
- SND_FORMAT_DISPLAY
- SND_FORMAT_MULAW_SQUELCH
- SND_FORMAT_EMPHASIZED
- SND_FORMAT_COMPRESSED
- SND_FORMAT_COMPRESSED_EMPHASIZED
- SND_FORMAT_DSP_COMMANDS
- SND_FORMAT_DSP_COMMANDS_SAMPLES
- SND_FORMAT_ADPCM_G721
- SND_FORMAT_ADPCM_G722
- SND_FORMAT_ADPCM_G723_3
- SND_FORMAT_ADPCM_G723_5
- SND_FORMAT_ALAW_8
- SND_FORMAT_AES
- SND_FORMAT_DELTA_MULAW_8

- `samplingRate`, and `channelCount` further describe the data.
- `info` is any null-terminated data that the application may need (for example, copyright information, textual description). The four bytes allocated are a minimum, and may be extended to whatever length is required.

Most of the methods defined in the Sound class are implemented so that you needn't be aware of this structure.

Error Codes

The following list contains the error codes returned by Sound object methods.

- SND_ERR_NONE
- SND_ERR_NOT_SOUND
- SND_ERR_BAD_FORMAT
- SND_ERR_BAD_RATE
- SND_ERR_BAD_CHANNEL

- SND_ERR_BAD_SIZE
- SND_ERR_BAD_FILENAME
- SND_ERR_CANNOT_OPEN
- SND_ERR_CANNOT_WRITE
- SND_ERR_CANNOT_READ
- SND_ERR_CANNOT_ALLOC
- SND_ERR_CANNOT_FREE
- SND_ERR_CANNOT_COPY
- SND_ERR_CANNOT_RESERVE
- SND_ERR_NOT_RESERVED
- SND_ERR_CANNOT_RECORD
- SND_ERR_ALREADY_RECORDING
- SND_ERR_NOT_RECORDING
- SND_ERR_CANNOT_PLAY
- SND_ERR_ALREADY_PLAYING
- SND_ERR_NOT_PLAYING
- SND_ERR_NOT_IMPLEMENTED
- SND_ERR_CANNOT_FIND
- SND_ERR_CANNOT_EDIT
- SND_ERR_BAD_SPACE
- SND_ERR_KERNEL
- SND_ERR_BAD_CONFIGURATION
- SND_ERR_CANNOT_CONFIGURE
- SND_ERR_UNDERRUN
- SND_ERR_ABORTED
- SND_ERR_BAD_TAG
- SND_ERR_CANNOT_ACCESS
- SND_ERR_TIMEOUT
- SND_ERR_BUSY
- SND_ERR_CANNOT_ABORT
- SND_ERR_INFO_TOO_BIG
- SND_ERR_UNKNOWN

See also the SoundMeter and SoundView classes.

Activity	Class Method
Creating and freeing a Sound object	+ addName:fromBundle: + addName:fromSoundfile: – dealloc – initFromPasteboard: – initFromSoundfile:

Activity	Class Method
Accessing the Sound name table	+ addName:sound: + findSoundFor: + removeSoundForName:
Reading and writing sound data	– dataForSound – readSoundfile: – writeSoundfile: – writeToPasteboard:
Modifying sound data	– convertToFormat:samplingRate:channelCount: – convertToFormat: – setDataSize:dataFormat:samplingRate: channelCount:infoSize: – setSoundStruct:soundStructSize: – setName: – name
Querying the object	– soundStruct – soundStructSize – data – dataFormat – dataSize – channelCount – samplingRate – sampleCount – duration – info – infoSize – isEmpty – compatibleWith: – processingError

Activity	Class Method
Recording and playing	– pause – pause: – isPlayable – play – play: – record – record: – resume – resume: – stop – stop: – samplesProcessed – status – soundBeingProcessed – soundStructBeingProcessed – waitUntilStopped
Editing sound data	– isEditable – copySamples:at:count: – copySound: – deleteSamples – deleteSamplesAt:count: – insertSamples:at: – needsCompacting – compactSamples
Accessing the delegate	– setDelegate: – delegate – tellDelegate:
Accessing the sound hardware	+ getVolume:: + setVolume:: + isMuted + setMute:
Methods implemented by the delegate	– didPlay: – didRecord: – hadError: – willPlay: – willRecord:

Class Methods

addName:fromBundle:

```
+ addName:(NSString *)name fromBundle:(NSBundle *)aBundle
```

Creates a Sound object from the sound resource named name in the NSBundle aBundle, assigns the name name to the object, and places the name on the sound name table. If name is already in use, or if the resource isn't found or can't be read, the Sound isn't created and nil is returned. Otherwise, the new Sound is returned.

addName:fromSoundfile:

```
+ addName:(NSString *)name fromSoundfile:(NSString *)filename
```

Creates a Sound object from the soundfile filename, assigns the name name to the object, and adds it to the named Sound table. If name is already in use, or if filename isn't found or can't be read, the Sound isn't created and nil is returned. Otherwise, the new Sound is returned.

addName:sound:

```
+ addName:(NSString *)name sound:(Sound *)aSound
```

Assigns the name name to the Sound aSound and adds it to the named Sound table. Returns aSound, or nil if name is already in use.

findSoundFor:

```
+ findSoundFor:(NSString *)aName
```

Finds and returns the named Sound object. First the named Sound table is searched; if the sound isn't found, then the method looks for "aName.snd" in the sound segment of the application's executable file. Finally, *the file* is searched for in the following directories (in order):

- ~/openstep/Library/Sounds
- /usr/local/openstep/Library/Sounds
- /usr/openstep/Library/Sounds

where ~ represents the user's home directory. If the Sound eludes the search, nil is returned.

getVolume::

```
+ getVolume:(float *)left :(float *)right
```

Returns, by reference, the stereo output levels as floating-point numbers between 0.0 and 1.0.

isMuted

```
+ (BOOL)isMuted
```

Returns YES if the sound output level is currently muted.

removeSoundForName:

```
+ removeSoundForName:(NSString *)name
```

Removes the named Sound from the named Sound table. If the Sound isn't found, returns nil; otherwise returns the Sound.

setMute:

```
+ setMute:(BOOL)aFlag
```

Mutes and unmutes the sound output level as aFlag is YES or NO, respectively. If successful, returns self; otherwise returns nil.

setVolume::

```
+ setVolume:(float)left :(float)right
```

Sets the stereo output levels. These affect the volume of the stereo signals sent to the built-in speaker and headphone jacks. left and right must be floating-point numbers between 0.0 (minimum) and 1.0 (maximum). If successful, returns self; otherwise returns nil.

Instance Methods

channelCount

```
- (int)channelCount
```

Returns the number of channels in the Sound.

compactSamples

```
- (int)compactSamples
```

The Sound's sampled data is compacted into a contiguous block, undoing the fragmentation that can occur during editing. If the Sound's data isn't fragmented (its format isn't SND_FORMAT_INDIRECT), then this method does nothing. Compacting a large sound can take a long time; keep in mind that when you copy a Sound to a pasteboard, the object is automatically compacted before it's copied. Also, the soundfile representation of a Sound contains contiguous data so there's no need to compact a Sound before writing it to a soundfile simply to ensure that the file representation will be compact. See the class description for a list of error codes returned by this method.

compatibleWith:

```
- (BOOL)compatibleWith:aSound
```

Returns YES if the format, sampling rate, and channel count of aSound's sound data is the same as that of the Sound receiving this message. If one (or both) of the Sounds doesn't contain a sound (soundStruct returns nil) then the objects are declared compatible and YES is returned.

convertToFormat:

```
- (int)convertToFormat:(int)newFormat
```

This is the same as convertToFormat:samplingRate:channelCount:, except that only the format is changed. See the class description for a list of error codes returned by this method.

convertToFormat:samplingRate:channelCount:

```
- (int)convertToFormat:(int)newFormat
    samplingRate:(double)newRate
    channelCount:(int)newChannelCount
```

Convert the Sound's data to the given format, sampling rate, and number of channels. The following conversions are possible:

- Arbitrary sampling rate conversion.
- Compression and decompression.
- Floating-point formats (including double-precision) to and from linear formats.
- Mono to stereo.
- Mu-law to and from linear formats.

See the class description for a list of error codes returned by this method.

copySamples:at:count:

```
- (int)copySamples:aSound at:(int)startSample
    count:(int)sampleCount
```

Replaces the Sound's sampled data with a copy of a portion of aSound's data. The copied portion starts at aSound's `startSample`'th sample (zero-based) and extends over `sampleCount` samples. The Sound receiving this message must be editable and the two Sounds must be compatible. If the specified portion of aSound is fragmented, the Sound receiving this message will also be fragmented. See the class description for a list of error codes returned by this method.

copySound:

```
- (int)copySound:aSound
```

Replaces the Sound's data with a copy of aSound's data. The Sound receiving this message needn't be editable, nor must the two Sounds be compatible. See the class description for a list of error codes returned by this method.

data

```
– (unsigned char *)data
```

Returns a pointer to the Sound's sampled data. You can use the pointer to examine, create, and modify the sound data. To intelligently manipulate the data, you need to be aware of its size, format, sampling rate, and the number of channels that it contains (a query method for each of these attributes is provided by the Sound class). The size of the data, in particular, must be respected; it's set when the Sound is created or given a new sound (through `readSoundfile:`, for example) and can't be changed directly. To resize the data, you should invoke one of the editing methods such as `insertSamples:at:` or `deleteSamplesAt:count:`. To start with a new, unfragmented sound with a determinate length, invoke the `setDataSize:dataFormat:samplingRate:channelCount:infoSize:` method. Keep in mind that the sound data in a fragmented sound is a pointer to a NULL-terminated list of pointers to SNDSoundStructs, one for each fragment. To examine or manipulate the samples in a fragmented sound, you must understand the SNDSoundStruct structure.

dataFormat

```
– (int)dataFormat
```

Returns the format of the Sound's data. If the data is fragmented, the format of the samples is returned (in other words, SND_FORMAT_INDIRECT is never returned by this method).

dataForSound

```
- (NSData *) dataForSound
```

Returns the Sound object's data enclosed in an NSData object.

dataSize

```
– (int)dataSize
```

Return the size (in bytes) of the Sound's data. If you modify the data (through the pointer returned by the data method) you must be careful not to exceed its length. If the sound is fragmented, the value returned by this method is the size of the Sound's SNDSoundStruct and doesn't include the actual data itself.

dealloc

```
- dealloc
```

Frees the Sound and deallocates its sound data. The Sound is removed from the named Sound table and its name made eligible for reuse.

delegate

```
- delegate
```

Returns the Sound's delegate.

deleteSamples

```
- (int)deleteSamples
```

Deletes all the samples in the Sound's data. The Sound must be editable. An error code is returned.

deleteSamplesAt:count:

```
- (int)deleteSamplesAt:(int)startSample count:(int)sampleCount
```

Deletes a range of samples from the Sound. `sampleCount` samples are deleted starting with the `startSample`'th sample (zero-based). The Sound must be editable and may become fragmented. See the class description for a list of error codes returned by this method.

duration

```
- (double)duration
```

Returns the Sound's length in seconds.

info

```
- (char *)info
```

Returns textual information about the sound. See the class description for more information.

infoSize

```
- (int)infoSize
```

Returns the size (in bytes) of the Sound's info string.

initFromPasteboard:

```
- initFromPasteboard:(NSPasteboard *)thePboard
```

Initializes the Sound instance, which must be newly allocated, by copying the sound data from the Pasteboard object `thePboard`. A pasteboard can have only one sound entry at a time. Returns `self` (an unnamed Sound) if `thePboard` currently contains a sound entry; otherwise, frees the newly allocated Sound and returns `nil`.

initFromSoundfile:

```
- initFromSoundfile:(NSString *)filename
```

Initializes the Sound instance, which must be newly allocated, from the soundfile `filename`. Returns `self` (an unnamed Sound) if the file was successfully read; otherwise, frees the newly allocated Sound and returns `nil`.

insertSamples:at:

```
- (int)insertSamples:aSound at:(int)startSample
```

Pastes the sound data in aSound into the Sound receiving this message, starting at the receiving Sound's `startSample`'th sample (zero-based). The receiving Sound doesn't lose any of its original sound data—the samples greater than or equal to `startSample` are moved to accommodate the inserted sound data. The receiving Sound must be editable and the two Sounds must be compatible (as determined by `isCompatible:`). If the method is successful, the receiving Sound is fragmented. See the class description for a list of error codes returned by this method.

isEditable

```
- (BOOL)isEditable
```

Returns YES if the Sound's format indicates that it can be edited, otherwise returns NO.

isEmpty

```
- (BOOL)isEmpty
```

Returns YES if the Sound doesn't contain any sound data, otherwise returns NO. This always returns NO if the Sound isn't editable, as determined by sending it the `isEditable` message.

isPlayable

```
- (BOOL)isPlayable
```

Returns YES if the Sound can be played, otherwise returns NO. Some unplayable Sounds just need to be converted to another format, sampling rate, or number of channels; others are inherently unplayable, such as those whose format is SND_FORMAT_DISPLAY.

name

```
- (NSString *)name
```

Returns the Sound's name.

needsCompacting

```
- (BOOL)needsCompacting
```

Returns YES if the Sound's data is fragmented. Otherwise returns NO.

pause

```
- (int)pause
```

Pauses the Sound during recording or playback. See the class description for a list of error codes returned by this method.

pause:

```
- pause:sender
```

Action method that pauses the Sound. Other than the argument and the return type, this is the same as the pause method.

play

```
- (int)play
```

Initiates playback of the Sound. The method returns immediately while the playback continues asynchronously in the background. The playback ends when the Sound receives the stop message, or when its data is exhausted.

When playback starts, willPlay: is sent to the Sound's delegate; when it stops, didPlay: is sent. See the class description for a list of error codes returned by this method. For this method to work properly, the main event loop must not be blocked.

play:

```
- play:sender
```

Action method that plays the Sound. Other than the argument and the return type, this is the same as the play method.

processingError

```
- (int)processingError
```

Returns a constant that represents the last error that was generated. See the class description for a list of error codes returned by this method

readSoundfile:

```
- (int)readSoundfile:(NSString *)filename
```

Replaces the Sound's contents with those of the soundfile filename. The Sound loses its current name, if any. See the class description for a list of error codes returned by this method.

record

```
- (int)record
```

Initiate recording into the Sound. The method returns immediately while the recording continues asynchronously in the background. The recording stops when the Sound receives the `stop` message or when the recording has gone on for the duration of the original sound data. The default recording lasts precisely ten minutes if not stopped. To record for a longer time, first increase the size of the sound data with `setSoundStruct:soundStructSize:` or `setDataSize:dataFormat:samplingRate:channelCount:infoSize:`.

When the recording begins, `willRecord:` is sent to the Sound's delegate; when the recording stops, `didRecord:` is sent. For this method to work properly, the main event loop must not be blocked. See the class description for a list of error codes returned by this method.

record:

```
- record:sender
```

Action method that initiates a recording. Other than the argument and return type, this is the same as the `record` method.

resume

```
- (int)resume
```

Resumes the paused Sound's activity. See the class description for a list of error codes returned by this method.

resume:

```
- resume:sender
```

Action method that resumes the paused Sound.

sampleCount

```
- (int)sampleCount
```

Returns the number of sample frames, or channel count-independent samples, in the Sound.

samplesProcessed

```
- (int)samplesProcessed
```

If the Sound is currently playing or recording, this returns the number of sample frames that have been played or recorded so far. Otherwise, the number of sample frames in the Sound is returned. If the sample frame count can't be determined, –1 is returned.

samplingRate

```
- (double)samplingRate
```

Returns the Sound's sampling rate.

setDataSize:dataFormat:samplingRate: channelCount:infoSize:

```
- (int)setDataSize:(int)newDataSize
    dataFormat:(int)newDataFormat
    samplingRate:(double)newSamplingRate
    channelCount:(int)newChannelCount
    infoSize:(int)newInfoSize
```

Allocates new, unfragmented sound data for the Sound, as described by the arguments. The Sound's previous data is freed. This method is useful for setting a determinate data length prior to a recording or for creating a scratch pad for algorithmic sound creation. See the class description for a list of error codes returned by this method.

setDelegate:

```
- setDelegate:anObject
```

Sets the Sound's delegate to `anObject`. The delegate may implement the following methods:

- `willPlay:`
- `didPlay:`
- `willRecord:`
- `didRecord:`
- `hadError:`

setName:

```
- setName:(NSString *)aName
```

Sets the Sound's name to aName. If aName is already being used, then the Sound's name isn't set and nil is returned; otherwise returns self.

setSoundStruct:soundStructSize:

```
- setSoundStruct:(SNDSoundStruct *)aStruct
    soundStructSize:(int)size
```

Sets the Sound's sound structure to aStruct. The size in bytes of the new structure, including its sound data storage, must be specified by size. This method can be used to set up a large buffer before recording into an existing Sound, by passing the existing SNDSoundStruct in the first argument while making size larger than the current size. (The default buffer holds ten minutes of mulaw sound.) The method is also useful in cases where aStruct already has sound data but isn't encapsulated in a Sound object yet. The Sound's status must be NX_SoundInitialized or NX_SoundStopped for this method to do anything. See the status method's description for a sound status list.

soundBeingProcessed

```
- soundBeingProcessed
```

Returns the Sound object that's being performed. The default implementation always returns self.

soundStruct

```
- (SNDSoundStruct *)soundStruct
```

Returns a pointer to the Sound's SNDSoundStruct structure that holds the object's sound data.

soundStructBeingProcessed

```
- (SNDSoundStruct *)soundStructBeingProcessed
```

Returns a pointer to the SNDSoundStruct structure that's being performed. This may not be the same structure as returned by the soundStruct method—Sound object's contain a private sound structure that may be used for playing recordings. If the Sound isn't currently playing or recording, then this will return the public structure.

soundStructSize

```
- (int)soundStructSize
```

Returns the size, in bytes, of the Sound's sound structure (returned by the soundStruct method). Use of this value requires a knowledge of the SNDSoundStruct architecture.

status

```
- (int)status
```

Return the Sound's current status, one of the following integer constants:

- NX_SoundStopped
- NX_SoundRecording
- NX_SoundPlaying
- NX_SoundInitialized
- NX_SoundRecordingPaused
- NX_SoundPlayingPaused
- NX_SoundRecordingPending
- NX_SoundPlayingPending
- NX_SoundFreed

stop

```
- (int)stop
```

Terminates the Sound's playback or recording. If the Sound was recording, the didRecord: message is sent to the delegate; if playing, didPlay: is sent. See the class description for a list of error codes returned by this method.

stop:

```
- stop:sender
```

Action method that stops the Sound's playback or recording. Other than the argument and the return type, this is the same as the stop method.

tellDelegate:

```
– tellDelegate:(SEL)theMessage
```

Sends theMessage to the Sound's delegate (only sent if the delegate implements theMessage). You never invoke this method directly; it's invoked automatically as the result of activities such as recording and playing. However, you can use it in designing a subclass of Sound.

waitUntilStopped

```
- (int)waitUntilStopped
```

Causes the system to wait until the Sound playback has stopped. Returns SND_ERR_NONE if no error occurred.

writeSoundfile:

```
– (int)writeSoundfile:(NSString *)filename
```

Writes the Sound's contents (its SNDSoundStruct and sound data) to the soundfile filename. See the class description for a list of error codes returned by this method.

writeToPasteboard:

```
– (int)writeToPasteboard:(Pasteboard *)thePboard
```

Puts a copy of the Sound's contents (its SNDSoundStruct and sound data) on the pasteboard maintained by the Pasteboard object thePboard. If the Sound is fragmented, it's compacted before the copy is created. See the class description for a list of error codes returned by this method.

Methods Implemented by the Delegate

didPlay:

– didPlay:sender

Sent to the delegate when the Sound stops playing.

didRecord:

– didRecord:sender

Sent to the delegate when the Sound stops recording.

hadError:

– hadError:sender

Sent to the delegate if an error occurs during recording or playback.

willPlay:

– willPlay:sender

Sent to the delegate when the Sound begins to play.

willRecord:

– willRecord:sender

Sent to the delegate when the Sound begins to record.

15

SoundMeter

Inherits From:	NSView: NSResponder: NSObject
Declared In:	soundkit/SoundMeter.h

Class Description

A `SoundMeter` is a view that displays the amplitude level of a sound as it's being recorded or played back. There are two working parts to the meter: A continuously-updated "running bar" that lengthens ands shrinks to depict the current amplitude level, and a "peak bubble" that displays and holds the greatest amplitude that was detected within the last few samples. An optional bezeled border is drawn around the object's frame.

To use a `SoundMeter`, you must first associate it with a `Sound` object, through the `setSound:` method, and then send the `SoundMeter` a `run:` message. To stop the meter's display, you send the object a `stop:` message. Neither `run:` nor `stop:` affect the performance of the meter's sound.

You can retrieve a `SoundMeter`'s running and peak values through the `floatValue` and `peakValue` methods. The values that these methods return are valid only while the `SoundMeter` is running. A `SoundMeter` also keeps track of the minimum and maximum amplitude over the duration of a run; these can be retrieved through `minValue` and `maxValue`. All `SoundMeter` amplitude levels are normalized to fit between 0.0 (inaudible) and 1.0 (maximum amplitude).

Method Types

Activity	Class Method
Initializing a SoundMeter instance	– initWithFrame:
Graphic attributes	– setBezeled: – isBezeled – setBackgroundGray: – backgroundGray – setForegroundGray: – foregroundGray – setPeakGray: – peakGray
Metering attributes	– setSound: – sound – setFloatValue: – setHoldTime: – holdTime
Retrieving meter values	– floatValue – maxValue – minValue – peakValue
Operating the object	– run: – isRunning – stop:
Drawing the object	– drawCurrentValue – drawRect:

Instance Methods

backgroundGray

```
- (float)backgroundGray
```

Returns the `SoundMeter`'s background color. The default is dark gray (`NSDarkGray`).

drawCurrentValue

```
- drawCurrentValue
```

Draws the SoundMeter's running bar and peak bubble. You never invoke this method directly; it's invoked automatically while the SoundMeter is running. You can override this method to change the look of the running bar and peak bubble.

drawRect:

```
- drawRect:(NSRect)rect
```

Draws all the components of the SoundMeter (frame, running bar, and peak bubble). You never invoke this method directly; however, you can override it in a subclass to change the way the components are displayed.

encodeWithCoder:

```
- (void)encodeWithCoder:(NSCoder *) aCoder
```

Encodes the receiving SoundMeter using aCoder. See the Application Kit's NSCoding protocol and NSCoder class documention for more information. See also initWithCoder:.

floatValue

```
- (float)floatValue
```

Returns the current running amplitude value as a floating-point number between 0.0 and 1.0. This is the amplitude level that's displayed by the running bar.

foregroundGray

```
- (float)foregroundGray
```

Returns the color of the running bar. The default is light gray (NSLightGray).

holdTime

```
- (float)holdTime
```

Returns the SoundMeter's hold time—the amount of time during which a peak amplitude is detected and displayed by the peak bubble—in seconds. The default is 0.7 seconds.

initWithCoder:

```
- initWithCoder: (NSCoder *) aCoder
```

Initializes and returns a new SoundMeter instance from data in aCoder. See the Application Kit's NSCoding protocol and NSCoder class documention for more information. See also encodeWithCoder:.

initWithFrame:

```
– initWithFrame:(const NSRect *)frameRect
```

Initializes the SoundMeter, fitting its graphic components within frameRect. The object's attributes are initialized as follows:

Attribute	Value
Peak hold time	0.7 seconds
Background gray	NSDarkGray
Running bar gray	NSLightGray
Peak bubble gray	NSWhite
Border	bezeled

isBezeled

```
– (BOOL)isBezeled
```

Returns YES (the default) if the SoundMeter has a border; otherwise, returns NO. Note that the SoundMeter class doesn't provide a method to change the type of border—it can display a bezeled border or none at all.

isRunning

```
– (BOOL)isRunning
```

Returns YES if the SoundMeter is currently running; otherwise, returns NO. The SoundMeter's status doesn't depend on the activity of its Sound object.

maxValue

```
- (float)maxValue
```

Returns the maximum running value so far. You can invoke this method after you stop this SoundMeter to retrieve the overall maximum value for the previous performance. The maximum value is cleared when you restart the SoundMeter.

minValue

```
- (float)minValue
```

Returns the minimum running value so far. You can invoke this method after you stop this SoundMeter to retrieve the overall minimum value for the previous performance. The minimum value is cleared when you restart the SoundMeter.

peakGray

```
- (float)peakGray
```

Returns the SoundMeter's peak bubble gray. The default is white (NSWhite).

peakValue

```
- (float)peakValue
```

Returns the most recently detected peak value as a floating-point number between 0.0 and 1.0. This is the amplitude level that's displayed by the peak bubble.

run:

```
- run:sender
```

Starts the SoundMeter running. The object SoundMeter must have a Sound object associated with it for this method to have an effect. Note that this method only affects the state of the SoundMeter—it doesn't trigger any activity in the Sound.

setBackgroundGray:

```
- setBackgroundGray:(float)aValue
```

Sets the SoundMeter's background color. The default is dark gray (NSDarkGray).

setBezeled:

```
- setBezeled:(BOOL)aFlag
```

If aFlag is YES, a bezeled border is drawn around the SoundMeter. If aFlag is NO and the SoundMeter has a frame, the frame is removed.

setFloatValue:

```
- setFloatValue:(float)aValue
```

Sets the current running value to aValue. You never invoke this method directly; it's invoked automatically when the SoundMeter is running. However, you can reimplement this method in a subclass of SoundMeter.

setForegroundGray:

```
- setForegroundGray:(float)aValue
```

Sets the SoundMeter's running bar color. The default is light gray (NSLightGray).

setHoldTime:

```
- setHoldTime:(float)seconds
```

Sets the SoundMeter's peak value hold time in seconds. This is the amount of time during which peak amplitudes are detected and held by the peak bubble.

setPeakGray:

– setPeakGray:(float)aValue

Sets the SoundMeter's peak bubble color. The default is white (NSWhite).

setSound:

– setSound:aSound

Sets the SoundMeter's Sound object.

sound

– sound

Returns the Sound object that the SoundMeter is metering.

stop:

– stop:sender

Stops the SoundMeter's metering activity. Note that this method only affects the state of the SoundMeter—it doesn't trigger any activity in the Sound.

SoundView

Inherits From:	NSView: NSResponder: NSObject
Declared In:	soundkit/SoundView.h

Class Description

A SoundView object provides a graphical representation of sound data. This data is taken from an associated Sound object. In addition to displaying a Sound object's data, a SoundView provides methods that let you play and record into the Sound object, and peform simple cut, copy, and paste editing of its data. A cursor into the display is provided, allowing the user to set the insertion point and to create a selection over the sound data.

Sound Display

Sounds are displayed on a two-dimensional graph. The amplitudes of individual samples are measured vertically and plotted against time, which proceeds left to right along the horizontal axis. A `SoundView`'s coordinate system is scaled and translated (vertically) so full amplitude fits within the bounds rectangle with 0.0 amplitude running through the center of the view.

For many sounds, the length of the sound data in samples is greater than the horizontal measure of the bounds rectangle. A `SoundView` employs a reduction factor to determine the ratio of samples to display units and plots the minimum and maximum amplitude values of the samples within that ratio. For example, a reduction factor of 10.0 means that the minimum and maximum values among the first ten samples are plotted in the first display unit, the minimum and maximum values of the next ten samples are displayed in the second display unit and so on.

Lines are drawn between the chosen values to yield a continuous shape. Two drawing modes are provided:

- In `NX_SOUNDVIEW_WAVE` mode, the drawing is rendered in an oscilloscopic fashion.
- In `NX_SOUNDVIEW_MINMAX` mode, two lines are drawn, one to connect the maximum values, and one to connect the minimum values.

As you zoom in (as the reduction factor decreases), the two drawing modes become indistinguishable.

Autoscaling the Display

When a `SoundView`'s sound data changes (due to editing or recording), the manner in which the `SoundView` is redisplayed depends on its **autoscale** flag. With autoscaling disabled, the `SoundView`'s frame grows or shrinks (horizontally) to fit the new sound data and the reduction factor is unchanged. If autoscaling is enabled, the reduction factor is automatically recomputed to maintain a constant frame size. By default, autoscaling is disabled; this is to accommodate the use of a `SoundView` object as the document of an `NSScrollView`.

Method Types

Activity	Class Method
Initializing a SoundView object	– initWithFrame:
Freeing a SoundView instance	– dealloc
Modifying the object	– scaleToFit – setBackgroundGray: – setBezeled: – setContinuous: – setDelegate: – setDisplayMode: – setEnabled: – setForegroundGray: – setOptimizedForSpeed: – setSound: – sizeToFit
Querying the object	– backgroundGray – delegate – displayMode – foregroundGray – getSelection:size: – isAutoScale – isBezeled – isContinuous – isEnabled – isOptimizedForSpeed – reductionFactor – sound
Selecting and editing the sound data	– copy: – cut: – delete: – mouseDown: – paste: – selectAll: – setSelection:size: – isEditable – setEditable:

Activity	Class Method
Pasteboard and Services support	– pasteboard:provideDataForType: – readSelectionFromPasteboard: – validRequestorForSendType:andReturnType: – writeSelectionToPasteboard:types:
Modifying the display coordinates	– setAutoscale: – setReductionFactor:
Drawing the object	– drawRect: – drawSamplesFrom:to: – hideCursor – showCursor
Responding to events	– acceptsFirstResponder – becomeFirstResponder – resignFirstResponder
Performing the sound data	– pause: – isPlayable – play: – record: – resume: – soundBeingProcessed – stop:
Communicating with the delegate	– tellDelegate:
Methods Implemented by the delegate	– didPlay: – didRecord: – hadError: – selectionDidChange: – soundDidChange: – willPlay: – willRecord:

Instance Methods

acceptsFirstResponder

```
- (BOOL)acceptsFirstResponder
```

If the SoundView is enabled, this returns YES, allowing the SoundView to become the first responder. Otherwise, it returns NO. This method is automatically invoked by objects defined by the Application Kit. You should never need to invoke it directly.

backgroundGray

```
- (float)backgroundGray
```

Returns the SoundView's background gray value (NSWhite by default).

becomeFirstResponder

```
- becomeFirstResponder
```

Promotes the SoundView to first responder. You never invoke this method directly.

copy:

```
- copy:sender
```

Copies the current selection to the pasteboard.

cut:

```
- cut:sender
```

Deletes the current selection from the SoundView, copies it to the pasteboard, and sends a soundDidChange: message to the delegate. The insertion point is positioned to where the selection used to start.

dealloc

```
- dealloc
```

Frees the SoundView but not its Sound object nor its delegate. The willFree: message is sent to the delegate.

delegate

```
- delegate
```

Returns the SoundView's delegate object.

delete:

```
- delete:sender
```

Deletes the current selection from the SoundView's Sound and sends the soundDidChange: message to the delegate. The deletion isn't placed on the pasteboard.

didPlay:

```
- didPlay:sender
```

Used to redirect delegate messages from the SoundView's Sound object; you never invoke this method directly.

didRecord:

```
- didRecord:sender
```

Used to redirect delegate messages from the SoundView's Sound object; you never invoke this method directly.

displayMode

```
- (int)displayMode
```

Returns the SoundView's display mode, one of NX_SOUNDVIEW_WAVE (oscilloscopic display) or NX_SOUNDVIEW_MINMAX (minimum / maximum display; this is the default).

drawSamplesFrom:to:

```
- drawSamplesFrom:(int)first to:(int)last
```

Redisplays the given range of samples.

drawRect:

```
- drawRect(NSRect)rect
```

Displays the SoundView's sound data. The selection is highlighted and the cursor is drawn (if it isn't currently hidden). Do not send this message directly to a SoundView object. To cause a SoundView to draw itself, send it one of the display messages defined by the NSView class.

foregroundGray

```
- (float)foregroundGray
```

Returns the SoundView's foreground gray value (NSBlack by default).

getSelection:size:

```
- getSelection:(int *)firstSample size:(int *)sampleCount
```

Returns the selection by reference. The index of the selection's first sample (counting from 0) is returned in firstSample. The size of the selection in samples is returned in sampleCount. The method itself returns self.

hadError:

```
- hadError:sender
```

Used to redirect delegate messages from the SoundView's Sound object; you never invoke this method directly.

hideCursor

```
- hideCursor
```

Hides the SoundView's cursor. This is usually handled automatically.

initWithFrame:

```
- initWithFrame:(NSRect)frameRect
```

Initializes the SoundView, fitting the object within the rectangle pointing to by frameRect. The initialized SoundView doesn't contain any sound data.

isAutoScale

```
- (BOOL)isAutoScale
```

Returns YES if the SoundView is in autoscaling mode, otherwise returns NO.

isBezeled

```
- (BOOL)isBezeled
```

Returns YES if the SoundView has a bezeled border, otherwise returns NO (the default).

isContinuous

```
- (BOOL)isContinuous
```

Returns YES if the SoundView responds to mouse-dragged events (as set through setContinuous:). The default is NO.

isEditable

```
- (BOOL)isEditable
```

Returns YES if the SoundView's sound data can be edited.

isEnabled

```
- (BOOL)isEnabled
```

Returns YES if the SoundView is enabled, otherwise returns NO. The mouse has no effect in a disabled SoundView. By default, a SoundView is enabled.

isOptimizedForSpeed

```
- (BOOL)isOptimizedForSpeed
```

Returns YES if the SoundView is optimized for speedy display. SoundViews are optimized by default.

isPlayable

```
- (BOOL)isPlayable
```

Returns YES if the SoundView's sound data can be played without first being converted.

mouseDown:

```
- mouseDown:(NSEvent *)theEvent
```

Allows a selection to be defined by clicking and dragging the mouse. This method takes control until a mouse-up occurs. While dragging, the selected region is highlighted. On mouse up, the delegate is sent the `selectionDidChange:` message. If `isContinuous` is `YES`, `selectionDidChange:` messages are also sent while the mouse is being dragged. You never invoke this method; it's invoked automatically in response to the user's actions.

paste:

```
- paste:sender
```

Replaces the current selection with a copy of the sound data currently on the pasteboard. If there is no selection the pasteboard data is inserted at the cursor position. The pasteboard data must be compatible with the `SoundView`'s data, as determined by the `Sound` method `compatibleWith:`. If the paste is successful, the `soundDidChange:` message is sent to the delegate.

pasteboard:provideDataForType:

```
- pasteboard:(NSPasteboard *) pboard
    provideDataForType:(NSString *)pboardType
```

Places the `SoundView`'s entire sound on the given pasteboard. Currently, the `pboardType` argument must be "NSSoundPboardType", the pasteboard type that represents sound data.

pause:

```
- pause:sender
```

Pauses the current playback or recording session by invoking `Sound`'s `pause:` method. If no sound is being processed, returns `nil`; otherwise, returns `self`.

play:

```
- play:sender
```

Play the current selection by invoking Sound's `play:` method. If there is no selection, the SoundView's entire Sound is played. The `willPlay:` message is sent to the delegate before the selection is played; `didPlay:` is sent when the selection is done playing.

readSelectionFromPasteboard:

```
- readSelectionFromPasteboard:(NSPasteboard *)pboard
```

Replaces the SoundView's current selection with the sound data on the given pasteboard. The pasteboard data is converted to the format of the data in the SoundView (if possible). If the SoundView has no selection, the pasteboard data is inserted at the cursor position. Sets the current error code for the SoundView's Sound object (which you can retrieve by sending `processingError` to the Sound) and returns `self`.

record:

```
- record:sender
```

Replaces the SoundView's current selection with newly recorded material. If there is no selection, the recording is inserted at the cursor. The `willRecord:` message is sent to the delegate before the recording is started; `didRecord:` is sent after the recording has completed. Recorded data is always taken from the CODEC microphone input.

reductionFactor

```
- (float)reductionFactor
```

Returns the SoundView's reduction factor, computed as

reductionFactor = sampleCount / displayUnits

resignFirstResponder

```
- resignFirstResponder
```

Resigns the position of first responder.

resume:

– resume:sender

Resumes the current playback or recording session by invoking Sound's resume: method. If no sound is being processed, returns nil; otherwise, returns self.

scaleToFit

– scaleToFit

Recomputes the SoundView's reduction factor to fit the sound data (horizontally) within the current frame. Invoked automatically when the SoundView's data changes and the SoundView is in autoscale mode. If the SoundView isn't in autoscale mode, sizeToFit is invoked when the data changes. You never invoke this method directly; a subclass can reimplement this method to provide specialized behavior.

selectAll:

– selectAll:sender

Creates a selection over the SoundView's entire Sound.

setAutoscale:

– setAutoscale:(BOOL)aFlag

Sets the SoundView's automatic scaling mode, used to determine how the SoundView is redisplayed when its data changes. With autoscaling enabled (aFlag is YES), the SoundView's reduction factor is recomputed so the sound data fits within the view frame. If it's disabled (aFlag is NO), the frame is resized and the reduction factor is unchanged. If the SoundView is in a ScrollingView, autoScaling should be disabled (autoscaling is disabled by default).

setBackgroundGray:

– setBackgroundGray:(float)aGray

Sets the SoundView's background gray value to aGray; the default is NSWhite.

setBezeled:

- setBezeled:(BOOL)aFlag

If aFlag is YES, the display is given a bezeled border. By default, the border of a SoundView display isn't bezeled. If autodisplaying is enabled, the Sound is automatically redisplayed.

setContinuous:

- setContinuous:(BOOL)aFlag

Sets the state of continuous action messages. If aFlag is YES, selectionDidChange: messages are sent to the delegate as the mouse is being dragged. If NO, the message is sent only on mouse up. The default is NO.

setDelegate:

- setDelegate:anObject

Sets the SoundView's delegate to anObject. The delegate is sent messages when the user changes or acts on the selection.

setDisplayMode:

- setDisplayMode:(int)aMode

Sets the SoundView's display mode, either NX_SOUNDVIEW_WAVE or NX_SOUNDVIEW_MINMAX (the default). If autodisplaying is enabled, the Sound is automatically redisplayed.

setEditable:

- setEditable:(BOOL)aFlag

Enables or disables editing in the SoundView as aFlag is YES or NO. By default, a SoundView is editable.

setEnabled:

- setEnabled:(BOOL)aFlag

Enables or disables the SoundView as aFlag is YES or NO. The mouse has no effect in a disabled SoundView. By default, a SoundView is enabled.

setForegroundGray:

```
- setForegroundGray:(float)aGray
```

Sets the SoundView's foreground gray value to aGray. The default is NSWhite.

setOptimizedForSpeed:

```
- setOptimizedForSpeed:(BOOL)flag
```

Sets the SoundView to optimize its display mechanism. Optimization greatly increases the speed with which data can be drawn, particularly for large sounds. It does so at the loss of some precision in representing the sound data; however, these inaccuracies are corrected as you zoom in on the data. All SoundView's are optimized by default.

setReductionFactor:

```
- setReductionFactor:(float)reductionFactor
```

Recomputes the size of the SoundView's frame, if autoscaling is disabled. The frame's size (in display units) is set according to the formula

```
displayUnits = sampleCount / reductionFactor
```

Increasing the reduction factor zooms out, decreasing zooms in on the data. If autodisplaying is enabled, the Sound is automatically redisplayed.

If the SoundView is in autoscaling mode, or reductionFactor is less than 1.0, the method avoids computing the frame size and returns nil. (In autoscaling mode, the reduction factor is automatically recomputed when the sound data changes—see scaleToFit:.) Otherwise, the method returns self. If reductionFactor is the same as the current reduction factor, the method returns immediately without recomputing the frame size.

setSelection:size:

```
- setSelection:(int)firstSample size:(int)sampleCount
```

Sets the selection to be `sampleCount` samples wide, starting with sample `firstSample` (samples are counted from 0).

setSound:

– `setSound:aSound`

Sets the `SoundView`'s `Sound` object to `aSound`. If autoscaling is enabled, the drawing coordinate system is adjusted so `aSound`'s data fits within the current frame. Otherwise, the frame is resized to accommodate the length of the data. If autodisplaying is enabled, the `SoundView` is automatically redisplayed.

showCursor

– `showCursor`

Displays the `SoundView`'s cursor. This is usually handled automatically.

sizeToFit

– `sizeToFit`

Resizes the `SoundView`'s frame (horizontally) to maintain a constant reduction factor. This method is invoked automatically when the `SoundView`'s data changes and the `SoundView` isn't in autoscale mode. If the `SoundView` is in autoscale mode, `scaleToFit` is invoked when the data changes. You never invoke this method directly; a subclass can reimplement this method to provide specialized behavior.

sound

– `sound`

Returns a pointer to the `SoundView`'s `Sound` object.

soundBeingProcessed

– `soundBeingProcessed`

Returns the `Sound` object that's currently being played or recorded into. Note that the actual `Sound` object that's being performed isn't necessarily the `SoundView`'s `sound` (the object returned by the `sound` method); for efficiency,

SoundView creates a private performance Sound object. While this is generally an implementation detail, this method is supplied in case the SoundView's delegate needs to know exactly which object will be (or was) performed.

stop:

```
- stop:sender
```

Stops the SoundView's current recording or playback.

tellDelegate:

```
- tellDelegate:(SEL)theMessage
```

Sends theMessage to the SoundView's delegate with the SoundView as the argument. If the delegate doesn't respond to the message, then it isn't sent. You normally never invoke this method; it's invoked automatically when an action, such as playing or editing, is performed. However, you can invoke it in the design of a SoundView subclass.

validRequestorForSendType:andReturnType:

```
- validRequestorForSendType:(NSString)sendType
    andReturnType:(NSString)returnType
```

You never invoke this method; it's implemented to support services that act on sound data.

willPlay:

```
- willPlay:sender
```

Used to redirect delegate messages from the SoundView's Sound object; you never invoke this method directly.

willRecord:

```
- willRecord:sender
```

Used to redirect delegate messages from the SoundView's Sound object; you never invoke this method directly.

writeSelectionToPasteboard:types:

```
- writeSelectionToPasteboard:(NSPasteboard *)pboard
    types:(NSArray *)types;
```

Places a copy of the SoundView's current selection on the given pasteboard. The types argument is currently ignored.

Methods Implemented by the Delegate

didPlay:

```
- didPlay:sender
```

Sent to the delegate just after the SoundView's sound is played.

didRecord:

```
- didRecord:sender
```

Sent to the delegate just after the SoundView's sound is recorded into.

hadError:

```
- hadError:sender
```

Sent to the delegate if an error is encountered during recording or playback of the SoundView's sound.

selectionDidChange:

```
- selectionDidChange:sender
```

Sent to the delegate when the SoundView's selection changes.

soundDidChange:

```
- soundDidChange:sender
```

Sent to the delegate when the SoundView's sound data is edited.

willFree:

– willFree:sender

Sent to the delegate when the SoundView is freed.

willPlay:

– willPlay:sender

Sent to the delegate just before the SoundView's sound is played.

willRecord:

– willRecord:sender

Sent to the delegate just before the SoundView's sound is recorded into.

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z