Application Kit Reference for Java

(Legacy)

Cocoa > Java

2007-02-01

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

Every effort has been made to ensure that the information in this document is accurate. Apple is not responsible for typographical errors.

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

Adobe, Acrobat, and PostScript are trademarks or registered trademarks of Adobe Systems Incorporated in the U.S. and/or other countries.

Simultaneously published in the United States and Canada.

Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

Figures

The Application Kit

Package: com.apple.cocoa.application

Introduction

Important: The Java API for the Application Kit is deprecated in Mac OS X version 10.4 and later. You should use the Objective-C API, documented in *Application Kit Framework Reference*, to develop Cocoa applications.

The Application Kit is a framework containing all the objects you need to implement your graphical, event-driven user interface: windows, panels, buttons, menus, scrollers, and text fields. The Application Kit handles all the details for you as it efficiently draws on the screen, communicates with hardware devices and screen buffers, clears areas of the screen before drawing, and clips views. The number of classes in the Application Kit may seem daunting at first. However, most Application Kit classes are support classes that you use indirectly. You also have the choice at which level you use the Application Kit:

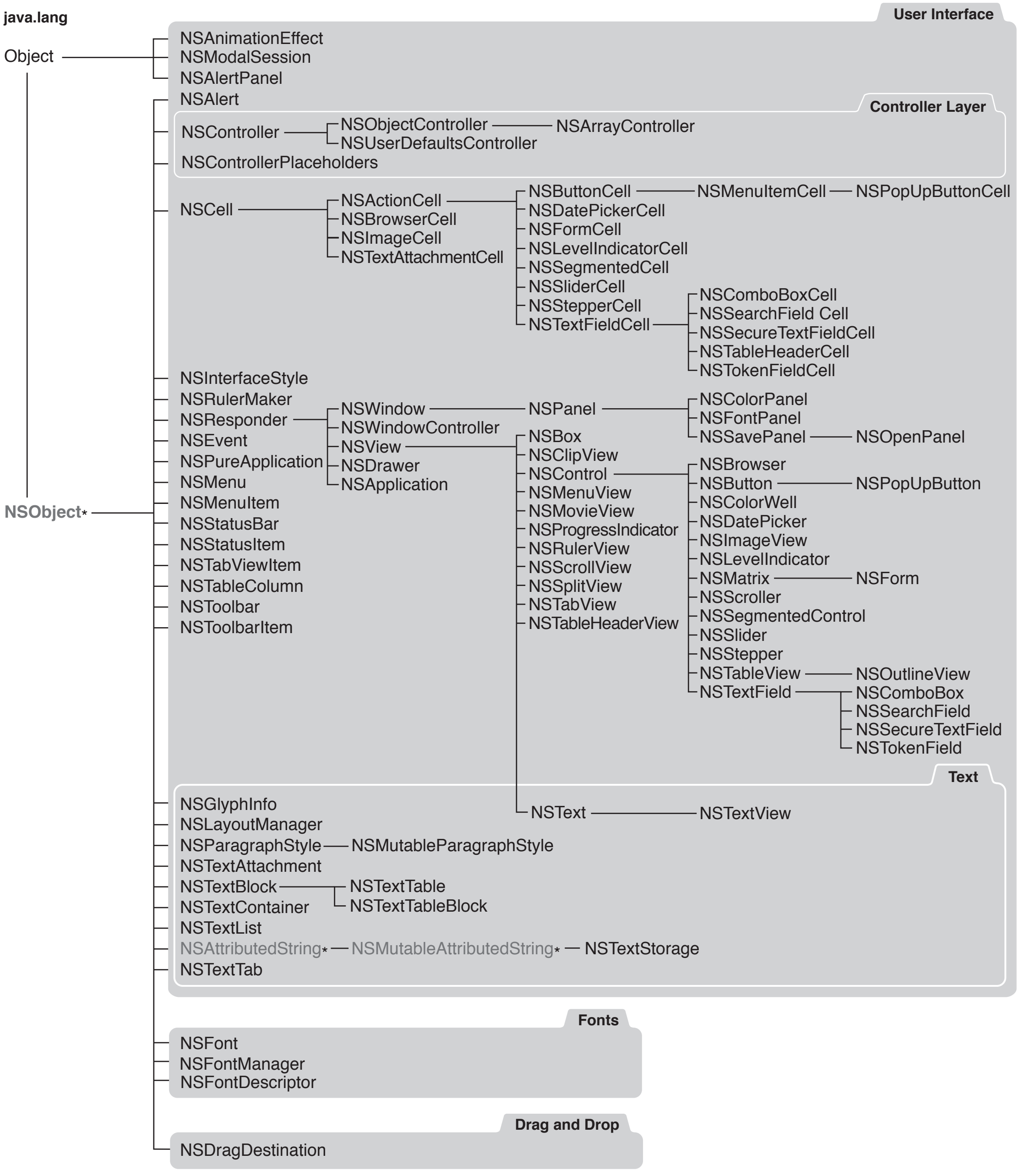
- Use Interface Builder to create connections from user interface objects to your application objects. In this case, all you need to do is implement your application classes—implement those action and delegate methods. For example, implement the method that is invoked when the user selects a menu item.
- Control the user interface programmatically, which requires more familiarity with Application Kit classes and interfaces. For example, allowing the user to drag an icon from one window to another requires some programming and familiarity with the NSDragging... interfaces.
- Implement your own objects by subclassing NSView or other classes. When subclassing NSView you write your own drawing methods using graphics functions. Subclassing requires a deeper understanding of how the Application Kit works.

To learn more about the Application Kit, review the NSApplication (page 93), NSWindow (page 1795), and NSView (page 1725) class specifications, paying close attention to delegate methods. For a deeper understanding of how the Application Kit works, see the specifications for NSResponder (page 1185) and NSRunLoop (NSRunLoop is in the Foundation framework).

Application Kit Classes and Interfaces

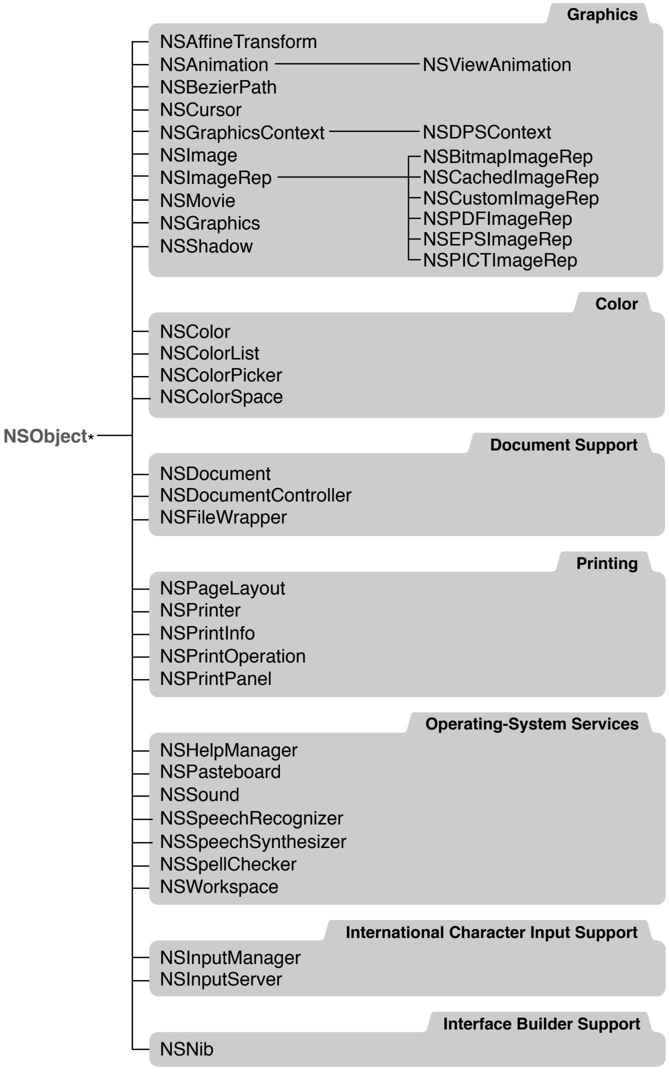
The Application Kit is large; it comprises more than 125 classes and interfaces. The classes all descend from the Foundation framework's NSObject class (see Figure I-1 (page 36)). The following sections briefly describe some of the topics that the Application Kit addresses through its classes and interfaces.

Figure I-1 The Application Kit class inheritance



*Class defined in the Foundation framework

Java Application Kit Continued



*Class defined in the Foundation framework

Encapsulating an Application

Every application uses a single instance of NSApplication to control the main event loop, keep track of the application's windows and menus, distribute events to the appropriate objects (that is, itself or one of its windows), and receive notification of application-level events. An NSApplication object has a delegate (an object that you assign) that is notified when the application starts or terminates, is hidden or activated, should open a file selected by the user, and so forth. By setting the NSApplication object's delegate and implementing the delegate methods, you customize the behavior of your application without having to subclass NSApplication.

General Event Handling and Drawing

The NSResponder class defines the responder chain, an ordered list of objects that respond to user events. When the user clicks the mouse button or presses a key, an event is generated and passed up the responder chain in search of an object that can "respond" to it. Any object that handles events must inherit from the NSResponder class. The core Application Kit classes, NSApplication, NSWindow, and NSView, inherit from NSResponder.

An NSApplication object maintains a list of NSWindow objects—one for each window belonging to the application—and each NSWindow object maintains a hierarchy of NSView objects. The view hierarchy is used for drawing and handling events within a window. An NSWindow object handles window-level events, distributes other events to its views, and provides a drawing area for its views. An NSWindow object also has a delegate allowing you to customize its behavior.

NSView is an abstract class for all objects displayed in a window. All subclasses implement a drawing method using graphics functions; `drawRect` (page 1753) is the primary method you override when creating a new NSView.

Panels

The NSPanel class is a subclass of NSWindow that you use to display transient, global, or pressing information. For example, you would use an instance of NSPanel, rather than an instance of NSWindow, to display error messages or to query the user for a response to remarkable or unusual circumstances. The Application Kit implements some common panels for you such as the Save, Open and Print panels, used to save, open, and print documents. Using these panels gives the user a consistent "look and feel" across applications for common operations.

Menus and Cursors

The NSMenu, NSMenuItem, 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, NSScrollView, and NSSplitView classes provide graphic "accessories" to other view objects or collections of views in windows. With the NSBox class, you can group elements in windows and draw a border around the entire group. The NSSplitView class lets you "stack" views vertically or horizontally, apportioning to each view some amount of a common territory; a sliding control bar lets the user redistribute the territory among views. The NSScrollView class and its helper class, NSClipView, provide a scrolling mechanism as well as the graphic objects that let the user initiate and control a scroll. The NSRulerView class allows you to add a ruler and markers to a scroll view.

Controlling an Application

The NSControl and NSCell classes, and their subclasses, define a common set of user interface objects such as 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 an action message to a target object. You typically use Interface Builder to set these targets and actions by Control-dragging from the control object to your application or other object. You can also set targets and actions programmatically.

An NSControl object is associated with one or more NSCell objects that implement the details of drawing and handling events. For example, a button comprises both an NSButton object and an NSButtonCell object. The reason for this separation of functionality is primarily to allow NSCell classes to be reused by NSControl classes. For example, NSMatrix and NSTableView can contain multiple NSCell objects of different types.

Tables

The NSTableView class displays data in row and column form. NSTableView is ideal for, but not limited to, displaying database records, where rows correspond to each record and columns contain record attributes. The user can edit individual cells and rearrange the columns. You control the behavior and content of an NSTableView object by setting its delegate and data source objects.

Text and Fonts

The NSTextField class implements a simple editable text field, and the NSTextView class provides more comprehensive editing features for larger text bodies.

NSTextView, a subclass of the abstract NSText class, defines the interface to Cocoa's extended text system. NSTextView supports rich text, attachments (graphics, file, and other), input management and key binding, and marked text attributes. NSTextView works with the font panel and menu, rulers and paragraph styles, the Services facility (for example, the spell-checking service), and the pasteboard. NSTextView also allows customizing through delegation and notifications—you rarely need to subclass NSTextView. You rarely create instances of NSTextView programmatically either, since objects on Interface Builder's palettes, such as NSTextField, NSForm, and NSScrollView, already contain NSTextView objects.

It is also possible to do more powerful and more creative text manipulation (such as displaying text in a circle) using NSTextStorage, NSLayoutManager, NSTextContainer, and related classes.

The NSFont and NSFontManager classes encapsulate and manage font families, sizes, and variations. The NSFont class defines a single object for each distinct font; for efficiency, these objects, which can be rather 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 classes NSImage and NSImageRep encapsulate graphics data, allowing you to easily and efficiently access images stored in files on the disk and displayed on the screen. NSImageRep subclasses each know how to draw an image from a particular kind of source data. 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.

Color is supported by the classes NSColor, NSColorPanel, NSColorList, NSColorPicker, and NSColorWell. NSColor supports a rich set of color formats and representations, including custom ones. The other classes are mostly interface classes: They define and present panels and views that allow the user to select and apply colors. For example, the user can drag colors from the color panel to any color well. The NSColorPicking interface lets you extend the standard color panel.

Dragging

With very little programming on your part, custom view objects can be dragged and dropped anywhere. Objects become part of this dragging mechanism by conforming to NSDragging... interfaces: draggable objects conform to the NSDraggingSource interface, and destination objects (receivers of a drop) conform to the NSDraggingDestination interface. The Application Kit hides all the details of tracking the cursor and displaying the dragged image.

Printing

The NSPrinter, NSPrintPanel, NSPageLayout, and NSPrintInfo classes work together to provide the means for printing the information that your application displays in its windows and views. You can also create an EPS representation of an NSView.

Accessing the File System

Use the NSFileWrapper class to create objects that correspond to files or directories on disk. NSFileWrapper will hold the contents of the file in memory so that it can be displayed, changed, or transmitted to another application. It also provides an icon for dragging the file or representing it as an attachment. Or use the NSFileManager class in the Foundation framework to access and enumerate file and directory contents. The NSOpenPanel and NSSavePanel classes also provide a convenient and familiar user interface to the file system.

Sharing Data With Other Applications

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

Checking Spelling

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

Localization

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. For example, an application may need to have separate Japanese, English, French, and German versions of character strings, icons, nib files, or context help. Resource files specific to a particular language are grouped together in a subdirectory of the bundle directory (the directories with the "`.lproj`" extension). Usually you set up localization resource files using Interface Builder. See the specification for NSBundle class for more information on localization (NSBundle is in the Foundation framework).

Classes

NSActionCell

Inherits from	NSCell : NSObject
Implements	NSCoding (NSCell)
Package:	com.apple.cocoa.application
Companion guide	Action Messages

Overview

Important: The information in this document is obsolete and should not be used for new development.

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; 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.

Tasks

Constructors

`NSActionCell` (page 47)
Creates an empty NSActionCell.

Configuring an NSActionCell

`setAlignment` (page 49)
Sets the alignment of text in the receiver.

`setBezeled` (page 49)
Sets whether the receiver draws itself with a bezeled border and marks it as needing redisplay.

`setBordered` (page 49)
Sets whether the receiver draws itself outlined with a plain border and marks it as needing redisplay.

`setEnabled` (page 49)
Sets whether the receiver is enabled or disabled.

`setFloatingPointFormat` (page 50)
: Sets the receiver's floating-point format as described in the NSCell class specification for the `setFloatingPointFormat` (page 325) method.

`setFont` (page 50)
: Sets the font to be used when the receiver displays text.

`setImage` (page 50)
: Sets the image to be displayed in the receiver.

Obtaining and Setting Cell Values

`doubleValue` (page 48)
: Returns the receiver's value as a `double` after validating any editing of cell content.

`floatValue` (page 48)
: Returns the receiver's value as a `float` after validating any editing of cell content.

`intValue` (page 48)
: Returns the receiver's value as an `int` after validating any editing of cell content.

`stringValue` (page 51)
: Returns the receiver's value as a string object as converted by the cell's formatter, if one exists.

`setObjectValue` (page 50)
: Discards any editing of the receiver's text and sets its object value to *`object`*.

Getting and Setting the Cell's View

`controlView` (page 47)
: Returns the view (normally an NSControl) in which the receiver was last drawn or `null` if the receiver has no control view (usually because it hasn't yet been placed in the view hierarchy).

`setControlView` (page 49)
: Sets the view (normally an NSControl) of the receiver to *`view`*.

Assigning Target and Action

`setAction` (page 48)
: Sets the selector used for the action message to *`aSelector`*.

`action` (page 47)
: Returns the receiver's action message selector.

`setTarget` (page 51)
: Sets the receiver's target object to *`anObject`*.

`target` (page 51)
: Returns the receiver's target object.

Assigning a Tag

`setTag` (page 51)
Sets the receiver's tag to *`anInt`*.

`tag` (page 51)
Returns the receiver's tag.

Constructors

NSActionCell

Creates an empty NSActionCell.

```
public NSActionCell()
```

Creates an NSActionCell initialized with *`aString`* and set to have the cell's default menu.

```
public NSActionCell(String aString)
```

Discussion
If no field editor (a shared NSText object) has been created for all NSActionCells, one is created.

Creates an NSActionCell initialized with *`anImage`* and set to have the cell's default menu.

```
public NSActionCell(NSImage anImage)
```

Discussion
If *`anImage`* is `null`, no image is set.

Instance Methods

action

Returns the receiver's action message selector.

```
public NSSelector action()
```

See Also
`setAction` (page 48)
`setTarget` (page 51)
`target` (page 51)

controlView

Returns the view (normally an NSControl) in which the receiver was last drawn or `null` if the receiver has no control view (usually because it hasn't yet been placed in the view hierarchy).

```
public NSView controlView()
```

doubleValue

Returns the receiver's value as a `double` after validating any editing of cell content.

```
public double doubleValue()
```

Discussion

If the receiver is not a text-type cell or the cell value is not scannable, the method returns 0.

See Also

`validateEditing` (page 463) (NSControl)

floatValue

Returns the receiver's value as a `float` after validating any editing of cell content.

```
public float floatValue()
```

Discussion

If the receiver is not a text-type cell or the cell value is not scannable, the method returns 0.

See Also

`validateEditing` (page 463) (NSControl)

intValue

Returns the receiver's value as an `int` after validating any editing of cell content.

```
public int intValue()
```

Discussion

If the receiver is not a text-type cell or the cell value is not scannable, the method returns 0.

See Also

`validateEditing` (page 463) (NSControl)

setAction

Sets the selector used for the action message to *aSelector*.

```
public void setAction(NSSelector aSelector)
```

See Also

`action` (page 47)

`setTarget` (page 51)

`target` (page 51)

setAlignment

Sets the alignment of text in the receiver.

```
public void setAlignment(int mode)
```

Discussion

mode is one of five constants: NSText.LeftTextAlignment, NSText.RightTextAlignment, NSText.CenterTextAlignment, NSText.JustifiedTextAlignment, and NSText.NaturalTextAlignment (the default alignment for the text). The method marks the receiver as needing redisplay after discarding any editing changes that were being made to cell text.

setBezeled

Sets whether the receiver draws itself with a bezeled border and marks it as needing redisplay.

```
public void setBezeled(boolean flag)
```

Discussion

The setBezeled and setBordered (page 49) methods are mutually exclusive—that is, a border can be only plain or bezeled.

setBordered

Sets whether the receiver draws itself outlined with a plain border and marks it as needing redisplay.

```
public void setBordered(boolean flag)
```

Discussion

The setBezeled (page 49) and setBordered methods are mutually exclusive—that is, a border can be only plain or bezeled.

setControlView

Sets the view (normally an NSControl) of the receiver to *view*.

```
public void setControlView(NSView view)
```

Discussion

The control view is typically set in the receiver's implementation of drawWithFrameInView (page 310) (NSCell). Set to null if the receiver has no control view (usually because it hasn't yet been placed in the view hierarchy).

Availability

Available in Mac OS X v10.4 and later.

setEnabled

Sets whether the receiver is enabled or disabled.

```
public void setEnabled(boolean flag)
```

Discussion
The text of disabled cells is changed to gray. If a cell is disabled, it cannot be highlighted, does not support mouse tracking (and thus cannot participate in target/action functionality), and cannot be edited. The method marks the receiver as needing redisplay after discarding any editing changes that were being made to cell text.

setFloatingPointFormat

Sets the receiver's floating-point format as described in the NSCell class specification for the `setFloatingPointFormat` (page 325) method.

```
public void setFloatingPointFormat(boolean autoRange, int leftDigits, int
    rightDigits)
```

Discussion
NSActionCell's implementation of the method supplements NSCell's by marking the receiver as needing redisplay after discarding any editing changes that were being made to cell text.

setFont

Sets the font to be used when the receiver displays text.

```
public void setFont(NSFont fontObj)
```

Discussion
If the receiver is not a text-type cell, the method converts it to that type. If *fontObj* is `null` and the receiver is a text-type cell, the font currently held by the receiver is autoreleased. NSActionCell supplements NSCell's implementation of this method by marking the updated cell as needing redisplay. If the receiver was converted to a text-type cell and is selected, it also updates the field editor with *fontObj*.

setImage

Sets the image to be displayed in the receiver.

```
public void setImage(NSImage image)
```

Discussion
If *image* is `null`, the image currently displayed by the receiver is removed.

setObjectValue

Discards any editing of the receiver's text and sets its object value to *object*.

```
public void setObjectValue(Object object)
```

Discussion
If the object value is afterward different from what it was before the method was invoked, the method marks the receiver as needing redisplay.

setTag

Sets the receiver's tag to *anInt*.

```
public void setTag(int anInt)
```

See Also

`tag` (page 51)

setTarget

Sets the receiver's target object to *anObject*.

```
public void setTarget(Object anObject)
```

See Also

`action` (page 47)

`setAction` (page 48)

`target` (page 51)

stringValue

Returns the receiver's value as a string object as converted by the cell's formatter, if one exists.

```
public String stringValue()
```

Discussion

If no formatter exists and the value is a String, returns the value as a plain, attributed, or localized formatted string. If the value is not a String or can't be converted to one, returns an empty string. The method supplements NSCell's implementation by validating and retaining any editing changes being made to cell text.

See Also

`validateEditing` (page 463) (NSControl)

tag

Returns the receiver's tag.

```
public int tag()
```

See Also

`setTag` (page 51)

target

Returns the receiver's target object.

```
public Object target()
```

See Also

`action` (page 47)

`setAction` (page 48)

`setTarget` (page 51)

NSAffineTransform Additions

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.application
Companion guide	Cocoa Drawing Guide

Overview

The Application Kit extends Foundation's NSAffineTransform class by adding:

- Methods for applying affine transformations to the current graphics context.
- A method for applying an affine transformation to an NSBezierPath.

Note: In Mac OS X v10.3 and earlier the NSAffineTransform class was declared and implemented entirely in the Application Kit framework. As of Mac OS X v10.4 the NSAffineTransform class has been split across the Foundation Kit and Application Kit frameworks.

Tasks

Setting the Current Transform in the Current Graphics State

`set` (page 54)
Sets the current transformation matrix to the receiver's transformation matrix.

`concat` (page 54)
Appends the receiver's matrix to the current transformation matrix stored in the current graphics context, replacing the current transformation matrix with the result.

Transforming Data and Objects

`transformBezierPath` (page 54)
Creates and returns a new NSBezierPath object with each point in *`aPath`* transformed by the receiver.

Instance Methods

concat

Appends the receiver's matrix to the current transformation matrix stored in the current graphics context, replacing the current transformation matrix with the result.

```
public void concat()
```

Discussion
Concatenation is performed by matrix multiplication—see "Manipulating Transform Values".

If this method is invoked from within an NSView's `drawRect` (page 1753) method, then the current transformation matrix is an accumulation of the screen, window, and any superview's transformation matrices. Invoking this method defines a new user coordinate system whose coordinates are mapped into the former coordinate system according to the receiver's transformation matrix. To undo the concatenation, you must invert the receiver's matrix and invoke this method again.

See Also
`set` (page 54)
`invert`

set

Sets the current transformation matrix to the receiver's transformation matrix.

```
public void set()
```

Discussion
The current transformation is stored in the current graphics context and is applied to subsequent drawing operations. You should use this method sparingly because it removes the existing transformation matrix, which is an accumulation of transformation matrices for the screen, window, and any superviews. Instead use the `concat` (page 54) method to add this transformation matrix to the current transformation matrix.

transformBezierPath

Creates and returns a new NSBezierPath object with each point in *`aPath`* transformed by the receiver.

```
public NSBezierPath transformBezierPath(NSBezierPath aPath)
```

Discussion
The original NSBezierPath object is not modified.

See Also
`transformPoint` `transformSize`

NSAlert

Inherits from	NSObject
Package:	com.apple.cocoa.application
Availability	Available in Mac OS X v10.3 and later.
Companion guides	Dialogs and Special Panels Sheet Programming Topics for Cocoa

Overview

You use an NSAlert object to display an alert, either as an application-modal dialog or as a sheet attached to a document window. The methods of the NSAlert class allow you to specify alert level, icon, button titles, and alert text. The class also lets your alerts display help buttons and provides ways for applications to offer help specific to an alert. To display an alert as a sheet, invoke the `beginSheet` (page 59) method; to display one as an application-modal dialog, use the `runModal` (page 61) method.

By design, an NSAlert object is intended for a single alert—that is, an alert with a unique combination of title, buttons, and so on—that is displayed upon a particular condition. It is recommended that you create a unique NSAlert object for different alert dialogs.

Subclassing Notes

The NSAlert class is not designed for subclassing.

Tasks

Constructors

`NSAlert` (page 57)
Creates an empty NSAlert.

Creating an Alert

`alertWithError` (page 58)
Returns an NSAlert object initialized from information in error object *`anError`*.

Managing Alert Text

`setInformativeText` (page 62)
Set's the receiver's informative text to *`informativeText`*.

`informativeText` (page 60)
Returns the receiver's informative text.

`setMessageText` (page 63)
Set's the receiver's message text, or title, to *`messageText`*.

`messageText` (page 61)
Returns the receiver's message text, or title.

Managing Alert Icon

`setIcon` (page 62)
Sets the icon to be displayed in the alert to *`icon`*.

`icon` (page 60)
Returns the icon displayed in the alert.

Managing Alert Buttons

`addButtonWithTitle` (page 58)
Adds a button with title *`aTitle`* to the alert.

`buttons` (page 59)
Returns the receiver's buttons.

Managing Help Text

`setShowsHelp` (page 63)
Sets whether the receiver has a help button.

`showsHelp` (page 63)
Returns whether the receiver has a help button.

`setHelpAnchor` (page 62)
Sets the help anchor associated with the alert to *`anchor`*.

`helpAnchor` (page 60)
Returns the receiver's HTML help anchor or `null` if there is none.

Managing Alert Style

`setAlertStyle` (page 61)
Sets the alert style of the receiver.

`alertStyle` (page 58)
Returns the constant identifying the receiver's alert style.

Managing the Delegate

`setDelegate` (page 62)
Sets the receiver's delegate, the object responsible for displaying help for the alert.

`delegate` (page 60)
Returns the receiver's delegate (not to be confused with the alert sheet's modal delegate).

Displaying the Alert

`runModal` (page 61)
Runs the receiver as an application-modal dialog and returns the constant positionally identifying the button clicked.

`beginSheet` (page 59)
Runs the receiver modally as a alert sheet attached to *window*.

Obtaining the Alert's Window

`window` (page 64)
Returns the application-modal dialog (an NSPanel object) or the document-modal sheet (an NSWindow object) associated with the receiver.

Showing help

`alertShowHelp` (page 65) *delegate method*
The delegate causes help to be displayed for *alert*, directly or indirectly.

Constructors

NSAlert

Creates an empty NSAlert.

```
public NSAlert()
```

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

Static Methods

alertWithError

Returns an NSAlert object initialized from information in error object `anError`.

```
public static NSAlert alertWithError(NSError anError)
```

Discussion
NSAlert extracts the localized error description, recovery suggestion, and recovery options from `anError` and uses them as the alert's message text, informative text, and button titles, respectively.

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

Instance Methods

addButtonWithTitle

Adds a button with title `aTitle` to the alert.

```
public NSButton addButtonWithTitle(String aTitle)
```

Discussion
Buttons are placed starting near the right side of the alert and going toward the left side (for left to right languages). The first three buttons are identified positionally as `FirstButtonReturn`, `SecondButtonReturn`, `ThirdButtonReturn` in the return-code parameter evaluated by the modal delegate. Subsequent buttons are identified as `ThirdButtonReturn` +*n*, where *n* is an integer. Do not make `aTitle` a `null` or empty string, as doing so claims a spot in the button array.

By default, the first button has a key equivalent of Return, any button with a title of "Cancel" has a key equivalent of Escape, and any button with the title "Don't Save" has a key equivalent of Command-d (but only if it is not the first button). You can also assign a different key equivalents for a button using NSButton's `setKeyEquivalent` (page 262) method. You can also use NSButton's `setTag` (page 460) method to set the return value.

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

See Also
`buttons` (page 59)

alertStyle

Returns the constant identifying the receiver's alert style.

```
public int alertStyle()
```

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

See Also
`setAlertStyle` (page 61)

beginSheet

Runs the receiver modally as a alert sheet attached to *window*.

```
public void beginSheet(NSWindow window, Object delegate, NSSelector didEndSelector,
     Object contextInfo)
```

Discussion
After it gets the user's response but before it dismisses the sheet, the receiver invokes the *didEndSelector* method in the modal *delegate*, passing in any contextual data specified in *contextInfo*.

The *didEndSelector* method should have the following signature:

```
void alertDidEnd(NSAlert alert, int returnCode, Object contextInfo);
```

where *alert* is the NSAlert object, *returnCode* specifies which button the user pressed, and *contextInfo* is the same *contextInfo* passed in the original message. The *returnCode* identifies which button the user clicked; it is one of the `...ButtonReturn` constants described in the Constants section. The modal delegate determines which button was clicked ("OK", "Cancel", and so on) and proceeds accordingly.

If you want to dismiss the sheet from within the *didEndSelector* method before the modal delegate carries out an action in response to the return value, send `orderOut` (page 1845) to the window object obtained by sending `window` (page 64) to the *alert* argument. This allows you to chain sheets, for example, by dismissing one sheet before showing the next from within the *didEndSelector* method. Note that you should be careful not to call `orderOut` on the sheet from elsewhere in your program before the *didEndSelector* method is invoked.

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

See Also
`runModal` (page 61)

buttons

Returns the receiver's buttons.

```
public NSArray buttons()
```

Discussion
The rightmost button is at index 0.

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

See Also
`addButtonWithTitle` (page 58)

delegate

Returns the receiver's delegate (not to be confused with the alert sheet's modal delegate).

```
public Object delegate()
```

Discussion

This delegate shows help related to the alert.

Availability

Available in Mac OS X v10.3 and later.

See Also

`setDelegate` (page 62)

`alertShowHelp` (page 65)

helpAnchor

Returns the receiver's HTML help anchor or `null` if there is none.

```
public String helpAnchor()
```

Availability

Available in Mac OS X v10.3 and later.

See Also

`setHelpAnchor` (page 62)

icon

Returns the icon displayed in the alert.

```
public NSImage icon()
```

Discussion

The default image is the application icon (property `NSApplicationIcon`).

Availability

Available in Mac OS X v10.3 and later.

See Also

`setIcon` (page 62)

informativeText

Returns the receiver's informative text.

```
public String informativeText()
```

Availability

Available in Mac OS X v10.3 and later.

See Also

`setInformativeText` (page 62)

`messageText` (page 61)

messageText

Returns the receiver's message text, or title.

```
public String messageText()
```

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

See Also

`setMessageText` (page 63)

`informativeText` (page 60)

runModal

Runs the receiver as an application-modal dialog and returns the constant positionally identifying the button clicked.

```
public int runModal()
```

Discussion
See "Constants" (page 64) for the list of applicable constants.

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

See Also

`beginSheet` (page 59)

setAlertStyle

Sets the alert style of the receiver.

```
public void setAlertStyle(int style)
```

Discussion
The alert style indicates the severity level of the alert. See "Constants" (page 64) for the list of alert style constants.

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

See Also

`alertStyle` (page 58)

setDelegate

Sets the receiver's delegate, the object responsible for displaying help for the alert.

```
public void setDelegate(Object delegate)
```

Discussion
To remove a delegate, invoke this method with `null` as the argument.

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

See Also
`delegate` (page 60)
`setShowsHelp` (page 63)

setHelpAnchor

Sets the help anchor associated with the alert to *anchor*.

```
public void setHelpAnchor(String anchor)
```

Discussion
To remove a help anchor, invoke this method with `null` as the argument.

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

See Also
`helpAnchor` (page 60),
`setShowsHelp` (page 63)

setIcon

Sets the icon to be displayed in the alert to *icon*.

```
public void setIcon(NSImage icon)
```

Discussion
By default, the image is the application icon, accessed via the application bundle's `NSApplicationIcon` property. To restore the application icon, invoke this method with an argument of `null`.

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

See Also
`icon` (page 60)

setInformativeText

Set's the receiver's informative text to *informativeText*.

```
public native void setInformativeText(String informativeText)
```

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

See Also
`informativeText` (page 60)
`setMessageText` (page 63)

setMessageText

Set's the receiver's message text, or title, to *`messageText`*.

```
public void setMessageText(String messageText)
```

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

See Also
`messageText` (page 61)
`setInformativeText` (page 62)

setShowsHelp

Sets whether the receiver has a help button.

```
public void setShowsHelp(boolean showsHelp)
```

Discussion
When the help button is pressed, the delegate (*`not`* the modal delegate) is first sent a `alertShowHelp` (page 65) message. If there is no delegate, or the delegate does not implement `alertShowHelp` or returns `false`, then the `openHelpAnchor` (page 742) message is sent to the application's NSHelpManager instance with a `null` book and the anchor specified by `setHelpAnchor` (page 62), if any. An exception is thrown if the delegate returns `false` and no help anchor is set.

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

See Also
`setDelegate` (page 62)
`showsHelp` (page 63)

showsHelp

Returns whether the receiver has a help button.

```
public boolean showsHelp()
```

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

See Also
`setShowsHelp` (page 63)

window

Returns the application-modal dialog (an NSPanel object) or the document-modal sheet (an NSWindow object) associated with the receiver.

```
public Object window()
```

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

Constants

NSAlert defines the following alert styles (each of type `int`):

Constant	Description
`WarningStyle`	An alert used to warn the user about a current or impending event. The purpose is more than informational but not critical. This is the default alert style.
`InformationalStyle`	An alert used to inform the user about a current or impending event.
`CriticalStyle`	Reserved this style for critical alerts, such as when there might be severe consequences as a result of a certain user response (for example, a "clean install" will erase all data on a volume). This style causes the icon to be badged with a caution icon.

Currently, there is no visual difference between informational and warning alerts. You should only use the critical (or "caution") alert style if warranted, as specified in "Alerts" in the *Apple Human Interface Guidelines*.

NSAlert return values for buttons are position dependent. The following constants describe the return values for the first three buttons on an alert (assuming a language that reads left to right).

Constant	Description
`FirstButtonReturn`	The user clicked the first (rightmost) button on the dialog or sheet.
`SecondButtonReturn`	The user clicked the second button from the right edge of the dialog or sheet.
`ThirdButtonReturn`	The user clicked the third button from the right edge of the dialog or sheet.

If you have more than three buttons on your alert, the button-position return value is `ThirdButtonReturn` + *n*, where *n* is an integer. For languages that read right to left, the first button's position is closest to the left edge of the dialog or sheet.

Delegate Methods

alertShowHelp

The delegate causes help to be displayed for *alert*, directly or indirectly.

```
public abstract boolean alertShowHelp(NSAlert alert)
```

Discussion
If it directly displays help, it returns whether this action successfully occurred. If the NSAlert object has a help anchor (setHelpAnchor (page 62)), the delegate can return false and the application's NSHelpManager instance will display help using the help anchor. This method is invoked when the user clicks the alert's help button. You do not need to implement this method unless you want to override the help-anchor lookup behavior.

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

See Also
setShowsHelp (page 63)

NSAlertPanel

Inherits from	Object
Package:	com.apple.cocoa.application
Companion guides	Dialogs and Special Panels Sheet Programming Topics for Cocoa

Overview

The objects created by NSAlertPanel are NSPanels (or, in user terminology, dialogs and sheets) displayed in modal sessions; they inform users of an event and offer, through buttons, a set of alternatives.

Tasks

Constructors

`NSAlertPanel` (page 68)

Running an Alert Panel

`runAlert` (page 71)
This method is deprecated. It creates and run an alert sheet on *aWindow*. Use `beginAlertSheet` (page 69) instead.

`runCriticalAlert` (page 72)
This method is deprecated. It creates and run a critical alert sheet on *aWindow*. Use `beginCriticalAlertSheet` (page 70) instead.

`runInformationalAlert` (page 72)
This method is deprecated. It creates and run an informational alert sheet on *aWindow*. Use `beginInformationalAlertSheet` (page 70) instead.

Getting an Alert Panel

`alertPanel` (page 68)

criticalAlertPanel (page 71)

informationalAlertPanel (page 71)

releaseAlert (page 71)

Starting Modal Sheets

beginAlertSheet (page 69)
Creates and runs an alert sheet

beginCriticalAlertSheet (page 70)
Creates and runs a critical alert sheet

beginInformationalAlertSheet (page 70)
Creates and runs an informational alert sheet

Constructors

NSAlertPanel

```
public NSRunAlertPanel()
```

Discussion
Returns an NSAlertPanel.

Static Methods

alertPanel

```
public static NSPanel alertPanel(String title, String message, String defaultButton,
     String alternateButton, String otherButton)
```

Discussion
Creates and returns an NSPanel object with the title of *title*, the text of *message*, and buttons with titles of *defaultButton*, *alternateButton*, and *otherButton*. The buttons are laid out right to left from the lower-right corner of the NSPanel. If *title* is null, a default localized title is used as the dialog title. This method creates a button only if the corresponding button title is non-null. You should deallocate objects returned by this method with the releaseAlert (page 71) method.

See Also
criticalAlertPanel (page 71)
informationalAlertPanel (page 71)

beginAlertSheet

Creates and runs an alert sheet

```
public static void beginAlertSheet(String title, String defaultButton, String
    alternateButton, String otherButton, NSWindow docWindow, Object modalDelegate,
     NSSelector didEndSelector, NSSelector didDismissSelector, Object contextInfo,
     String message)
```

Discussion

on *docWindow*, with the title of *title*, the text of *message*, and buttons with titles of *defaultButton*, *alternateButton*, and *otherButton*.

The buttons are laid out on the lower-right corner of the sheet, with *defaultButton* on the right, *alternateButton* on the left, and *otherButton* in the middle. If *title* is null or an empty string, a default localized title is used ("Alert" in English). If *defaultButton* is null or an empty string, a default localized button title ("OK" in English) is used. For the remaining buttons, this method creates them only if their corresponding button title is non-null.

A Command-D key equivalent for the "Don't Save" button is provided, if one is found. The button titles are searched for the localized value for "Don't Save." If a match is found, that button is assigned a Command-D key equivalent, provided it is not the default button.

If you create a modal panel using runModalForWindow (page 119) or beginSheet (page 108), you can assign the key equivalent yourself, using setKeyEquivalent (page 262) and setKeyEquivalentModifierMask (page 263).

When the modal session is ended, and before the sheet is dismissed, the *didEndSelector* is invoked on the *modalDelegate*. passing *contextInfo*. After the sheet is dismissed, the *didDismissSelector* is invoked on the *modalDelegate*, passing *contextInfo*. Typically, you will want to implement the *didEndSelector*, but you may pass null for the *didDismissSelector*. The two selectors should be defined as follows:

```
sheetDidEnd(NSWindow sheet,  int returnCode,  Object contextInfo)
sheetDidDismiss(NSWindow sheet,  int returnCode,  Object contextInfo)
```

where *sheet* is the alert sheet, *returnCode* specifies which button the user pressed, and *contextInfo* is the same *contextInfo* passed into beginAlertSheet. *returnCode* can be one of the following:

- DefaultReturn means the user pressed the default button.
- AlternateReturn means the user pressed the alternate button.
- OtherReturn means the user pressed the other button.
- ErrorReturn means an error occurred while running the alert panel.

This method is deprecated. Creates and runs an alert sheet. Use the version with the *contextInfo* argument instead.

```
public static void beginAlertSheet(String title, String defaultButton, String
    alternateButton, String otherButton, NSWindow docWindow, Object modalDelegate,
     NSSelector didEndSelector, NSSelector didDismissSelector, String message)
```

beginCriticalAlertSheet

Creates and runs a critical alert sheet

```
public static void beginCriticalAlertSheet(String title, String defaultButton,
    String alternateButton, String otherButton, NSWindow docWindow, Object
    modalDelegate, NSSelector didEndSelector, NSSelector didDismissSelector, Object
     contextInfo, String message)
```

Discussion

on *docWindow*, with the title of *title*, the text of *message*, and buttons with titles of *defaultButton*, *alternateButton*, and *otherButton*.

See the description of `beginAlertSheet` (page 69) for information on layout, default parameters, and the selectors.

The sheet presented to the user is badged with a caution icon. Critical alerts should be used only as specified in the "Alerts" section of *Apple Human Interface Guidelines*.

This method is deprecated. Creates and runs a critical alert sheet. Instead, use the version with the *contextInfo* argument.

```
public static void beginCriticalAlertSheet(String title, String defaultButton,
    String alternateButton, String otherButton, NSWindow docWindow, Object
    modalDelegate, NSSelector didEndSelector, NSSelector didDismissSelector, String
     message)
```

beginInformationalAlertSheet

Creates and runs an informational alert sheet

```
public static void beginInformationalAlertSheet(String title, String defaultButton,
     String alternateButton, String otherButton, NSWindow docWindow, Object
    modalDelegate, NSSelector didEndSelector, NSSelector didDismissSelector, Object
     contextInfo, String message)
```

Discussion

on *docWindow*, with the title of *title*, the text of *message*, and buttons with titles of *defaultButton*, *alternateButton*, and *otherButton*.

See the description of `beginAlertSheet` (page 69) for information on layout, default parameters, and the selectors.

This method is deprecated. Creates and runs an informational alert sheet. Instead, use the version with the *contextInfo* argument.

```
public static void beginInformationalAlertSheet(String title, String defaultButton,
     String alternateButton, String otherButton, NSWindow docWindow, Object
    modalDelegate, NSSelector didEndSelector, NSSelector didDismissSelector, String
     message)
```

criticalAlertPanel

```
public static NSPanel criticalAlertPanel(String title, String message, String
    defaultButton, String alternateButton, String otherButton)
```

Discussion

Creates and returns an NSPanel object with the title of `title`, the text of `message`, and buttons with titles of `defaultButton`, `alternateButton`, and `otherButton`. The NSPanel looks and behaves no differently than a normal alert panel (or dialog). See the description of `alertPanel` (page 68) for information on layout and default parameters. You should deallocate objects returned by this method with the `releaseAlert` (page 71) method.

See Also

`informationalAlertPanel` (page 71)

informationalAlertPanel

```
public static NSPanel informationalAlertPanel(String title, String message, String
     defaultButton, String alternateButton, String otherButton)
```

Discussion

Creates and returns an NSPanel object with the title of `title`, the text of `message`, and buttons with titles of `defaultButton`, `alternateButton`, and `otherButton`. The NSPanel looks and behaves no differently than a normal alert panel (or dialog). See the description of `alertPanel` (page 68) for information on layout and default parameters. You should deallocate objects returned by this method with the `releaseAlert` (page 71) method.

See Also

`criticalAlertPanel` (page 71)

releaseAlert

```
public static void releaseAlert(NSPanel alertPanel)
```

Discussion

Sends autorelease to the delegate of `alertPanel` (page 68) designated by `alertPanel`.

runAlert

```
public static int runAlert(String title, String message, String defaultButton,
    String alternateButton, String otherButton)
```

Discussion

Creates and runs an alert panel (or dialog) with the title of `title`, the text of `message`, and buttons with titles of `defaultButton`, `alternateButton`, and `otherButton`. This method returns a constant indicating which button was pressed (see "Constants" (page 73) for details) or `ErrorReturn` if an error occurred running the modal panel. The buttons are laid out on the lower-right corner of the sheet, with `defaultButton` on the right, `alternateButton` on the left, and `otherButton` in the middle. If `title` is `null` or an empty string, a default localized title is used ("Alert" in English). If `defaultButton` is `null` or an empty string, a default localized button title ("OK" in English) is used. For the remaining buttons, this method creates them only if their corresponding button title is non-`null`.

A Command-D key equivalent for the "Don't Save" button is provided, if one is found. The button titles are searched for the localized value for "Don't Save." If a match is found, that button is assigned a Command-D key equivalent, provided it is not the default button.

If you create a modal panel using `runModalForWindow` (page 119) or `beginSheet` (page 108), you can assign the key equivalent yourself, using `setKeyEquivalent` (page 262) and `setKeyEquivalentModifierMask` (page 263).

See Also

`runCriticalAlert` (page 72)

`runInformationalAlert` (page 72)

This method is deprecated. It creates and run an alert sheet on *aWindow*. Use `beginAlertSheet` (page 69) instead.

```
public static int runAlert(String title, String message, String defaultButton,
    String alternateButton, String otherButton, NSWindow aWindow)
```

runCriticalAlert

```
public static int runCriticalAlert(String title, String message, String
    defaultButton, String alternateButton, String otherButton)
```

Discussion

Creates and runs an alert panel (or dialog) with the title of *title*, the text of *message*, and buttons with titles of *defaultButton*, *alternateButton*, and *otherButton*. The NSPanel looks and behaves no differently than a normal alert panel (or dialog). See the description of `runAlert` (page 71) for information on layout and default parameters.

The panel presented to the user is badged with a caution icon. Critical alerts should be used only as specified in the "Alerts" section of *Apple Human Interface Guidelines*.

See Also

`runInformationalAlert` (page 72)

This method is deprecated. It creates and run a critical alert sheet on *aWindow*. Use `beginCriticalAlertSheet` (page 70) instead.

```
public static int runCriticalAlert(String title, String message, String
    defaultButton, String alternateButton, String otherButton, NSWindow aWindow)
```

runInformationalAlert

```
public static int runInformationalAlert(String title, String message, String
    defaultButton, String alternateButton, String otherButton)
```

Discussion

Creates and runs an alert panel (or dialog) with the title of *title*, the text of *message*, and buttons with titles of *defaultButton*, *alternateButton*, and *otherButton*. The NSPanel looks and behaves no differently than a normal alert panel (or dialog). See the description of `runAlert` (page 71) for information on layout and default parameters.

See Also
`runCriticalAlert` (page 72)

This method is deprecated. It creates and run an informational alert sheet on *aWindow*. Use `beginInformationalAlertSheet` (page 70) instead.

```
public static int runInformationalAlert(String title, String message, String
    defaultButton, String alternateButton, String otherButton, NSWindow aWindow)
```

Constants

Constant	Description
`DefaultReturn`	The value returned when the first (default) button from the right edge of the NSAlertPanel is clicked
`AlternateReturn`	The value returned when the second button from the right edge of the NSAlertPanel is clicked
`OtherReturn`	The value returned when the third button from the right edge of the NSAlertPanel is clicked
`ErrorReturn`	The value returned if running the NSAlertPanel resulted in an error

NSAnimation

Inherits from	NSAnimation : NSObject
Package:	com.apple.cocoa.application
Availability	Available in Mac OS X v10.4 and later.
Companion guide	Drawing and Views Programming Topics for Cocoa

Overview

Objects of the NSAnimation class manage the timing and progress of animations in the user interface. The class also lets you link together multiple animations so that when one animation ends another one starts. It does not provide any drawing support for animation and does not directly deal with views, targets, or actions.

Note: For simple tasks requiring a timing mechanism, consider using NSTask.

NSAnimation objects have several characteristics, including duration, frame rate, and animation curve, which describes the relative speed of the animation over its course. You can set progress marks in an animation, each of which specifies a percentage of the animation completed; when an animation reaches a progress mark, it notifies its delegate and posts a notification to any observers. Animations execute in one of three blocking modes: blocking, non-blocking on the main thread, and non-blocking on a separate thread. The non-blocking modes permit the handling of user events while the animation is running.

Subclassing Notes

For information on subclassing NSAnimation, see “Timing Animations” in *Drawing and Views Programming Topics for Cocoa*.

Tasks

Constructors

`NSAnimation` (page 78)
Creates an NSAnimation object initialized with the specified duration and animation curve.

Configuring an Animation

`animationBlockingMode` (page 78)
: Returns the blocking mode the receiver is next scheduled to run under.

`setAnimationBlockingMode` (page 82)
: Sets the blocking mode of the receiver.

`runLoopModesForAnimating` (page 82)
: Overridden to return the NSRunLoop modes that the receiver uses to run the animation timer in.

`animationCurve` (page 79)
: Returns the animation curve the receiver is running under.

`setAnimationCurve` (page 83)
: Sets the receiver's animation curve.

`delegate` (page 80)
: Returns the delegate of the receiver.

`setDelegate` (page 83)
: Sets the delegate of the receiver.

`duration` (page 81)
: Returns the duration of the animation, in seconds.

`setDuration` (page 83)
: Sets the duration of the animation to a specified number of seconds.

`frameRate` (page 81)
: Returns the frame rate of the animation as the number of updates per second.

`setFrameRate` (page 84)
: Sets the frame rate—the number of updates per second—of the receiver.

Controlling and Monitoring the Animation

`startAnimation` (page 84)
: Starts the animation represented by the receiver.

`stopAnimation` (page 85)
: Stops the animation represented by the receiver.

`isAnimating` (page 81)
: Returns whether the receiver is currently animating.

`currentProgress` (page 80)
: Returns the current progress of the receiver as a `float` value.

`setCurrentProgress` (page 83)
: Sets the current progress of the receiver.

`currentValue` (page 80)
: Returns the current value of the effect based on the current progress.

Managing Progress Marks

`addProgressMark` (page 78)
Adds the progress mark to the receiver.

`removeProgressMark` (page 82)
Removes progress mark from the receiver.

`progressMarks` (page 81)
Returns the receiver's progress marks.

`setProgressMarks` (page 84)
Sets the receiver's progress marks to the values specified in the passed-in array.

Linking Animations Together

`startWhenAnimation` (page 85)
Starts running the animation represented by the receiver when another animation reaches a specific progress mark.

`stopWhenAnimation` (page 86)
Stops running the animation represented by the receiver when another animation reaches a specific progress mark.

`clearStartAnimation` (page 79)
Clears linkage to another animation made with `startWhenAnimation` (page 85).

`clearStopAnimation` (page 79)
Clears linkage to another animation made with `stopWhenAnimation` (page 86)

Methods for the delegate

`animationDidEnd` (page 87) *delegate method*
Sent to the delegate when the specified animation completes its run.

`animationDidStop` (page 87) *delegate method*
Sent to the delegate when the specified animation is stopped before it completes its run.

`animationDidReachProgressMark` (page 87) *delegate method*
Sent to the delegate when an animation reaches a specific progress mark.

`animationShouldStart` (page 88) *delegate method*
Sent to the delegate just after the animation *`animation`* receives a `startAnimation` (page 84) message.

`animationValueForProgress` (page 88) *delegate method*
Sent to the delegate at each frame update, passing in a reference to the running *`animation`* animation and its current progress value *`progress`*.

Constructors

NSAnimation

```
public NSAnimation()
```

Discussion
Creates an NSAnimation object.

Creates an NSAnimation object initialized with the specified duration and animation curve.

```
public NSAnimation(double duration, int animationCurve)
```

Discussion
The *duration* value specifies the number of seconds over which the animation occurs; specifying a negative number throws an exception. You can always later change the duration of an NSAnimation object by sending it a `setDuration` (page 83) message, even while the animation is running. The *animationCurve* parameter is a constant that describes the relative speed of the animation over its course; if it is zero, the default curve (`NSAnimationEaseInOut`) is used. See "Constants" (page 86) for descriptions of these constants.

Instance Methods

addProgressMark

Adds the progress mark to the receiver.

```
public void addProgressMark(float progressMark)
```

Discussion
The *progressMark* argument is a `float` value between 0.0 and 1.0. A progress mark represent a percentage of the animation completed. When the animation reaches a progress mark, a `animationDidReachProgressMark` (page 87) is sent to the delegate and an `AnimationProgressMarkNotification` (page 88) is broadcast to all observers. You might receive multiple notifications of progress advances over multiple marks.

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

See Also
`currentProgress` (page 80)
`removeProgressMark` (page 82)

animationBlockingMode

Returns the blocking mode the receiver is next scheduled to run under.

```
public int animationBlockingMode()
```

Discussion
The animation can run in blocking mode or non-blocking mode; non-blocking mode can be either on the main thread or on a separate thread. See "Constants" (page 86) for valid constants. The default mode is `AnimationBlocking`.

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

See Also
`setAnimationBlockingMode` (page 82)

animationCurve

Returns the animation curve the receiver is running under.

```
public int animationCurve()
```

Discussion
The animation curve describes the relative frame rate over the course of the animation. See "Constants" (page 86) for valid constants.

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

See Also
`setAnimationCurve` (page 83)

clearStartAnimation

Clears linkage to another animation made with `startWhenAnimation` (page 85).

```
public void clearStartAnimation()
```

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

See Also
`startAnimation` (page 84)

clearStopAnimation

Clears linkage to another animation made with `stopWhenAnimation` (page 86)

```
public void clearStopAnimation()
```

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

See Also
`stopAnimation` (page 85)

currentProgress

Returns the current progress of the receiver as a `float` value.

```
public float currentProgress()
```

Discussion

The current progress is a value between 0.0 and 1.0 that represents the percentage of the animation currently completed.

Availability

Available in Mac OS X v10.4 and later.

See Also

`setCurrentProgress` (page 83)

currentValue

Returns the current value of the effect based on the current progress.

```
public float currentValue()
```

Discussion

NSAnimation gets the current value from the delegate in `animationValueForProgress` (page 88) or, if that method is not implemented, computes it from the current progress by factoring in the animation curve. Although this method has no corresponding setter method, you may override it to provide a custom curve. The current value can be less than 0.0 or greater than 1.0. For example, if you make the value greater than 1.0 you can achieve a “rubber effect” where the size of a view is temporarily larger before its final size. Subclasses may override this method to return a custom curve value instead of implementing `animationValueForProgress` (page 88), thereby saving on the overhead of using a delegate.

Availability

Available in Mac OS X v10.4 and later.

See Also

`currentProgress` (page 80)

`setAnimationCurve` (page 83)

delegate

Returns the delegate of the receiver.

```
public Object delegate()
```

Availability

Available in Mac OS X v10.4 and later.

See Also

`setDelegate` (page 83)

duration

Returns the duration of the animation, in seconds.

```
public double duration()
```

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

See Also
`setDuration` (page 83)

frameRate

Returns the frame rate of the animation as the number of updates per second.

```
public float frameRate()
```

Discussion
The frame rate is not guaranteed to be accurate because of differences between systems on the time needed to process a frame.

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

isAnimating

Returns whether the receiver is currently animating.

```
public boolean isAnimating()
```

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

progressMarks

Returns the receiver's progress marks.

```
public NSArray progressMarks()
```

Discussion
Each item in the returned array is an NSNumber object containing a `float` value. If the receiver has no progress marks, an empty array is returned.

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

See Also
`addProgressMark` (page 78)
`setProgressMarks` (page 84)

removeProgressMark

Removes progress mark from the receiver.

```
public void removeProgressMark(float progressMark)
```

Discussion

progressMark is a float value between 0.0 and 1.0 that indicates the portion of the animation completed.

Availability

Available in Mac OS X v10.4 and later.

See Also

addProgressMark (page 78)

runLoopModesForAnimating

Overridden to return the NSRunLoop modes that the receiver uses to run the animation timer in.

```
public NSArray runLoopModesForAnimating()
```

Discussion

By default, the method returns null, which indicates that the animation can be run in default, modal, or event-tracking mode. The value returned from this method is ignored if the animation blocking mode is something other than AnimationNonblocking.

Availability

Available in Mac OS X v10.4 and later.

See Also

setAnimationBlockingMode (page 82)

setAnimationBlockingMode

Sets the blocking mode of the receiver.

```
public void setAnimationBlockingMode(int animationBlockingMode)
```

Discussion

The new blocking mode takes effect the next time the receiver is started and has no effect on an animation underway. The default mode is AnimationBlocking, which means that the animation runs on the main thread in a custom run-loop mode that blocks user events. If *animationBlockingMode* is AnimationNonblocking the animation runs in the main thread in one of the standard run-loop modes or in a mode returned from runLoopModesForAnimating (page 82). If *animationBlockingMode* is AnimationNonblockingThreaded a new thread is spawned to run the animation. See "Constants" (page 86) for valid constants.

Availability

Available in Mac OS X v10.4 and later.

See Also

animationBlockingMode (page 78)

setAnimationCurve

Sets the receiver's animation curve.

```
public void setAnimationCurve(int curve)
```

Discussion
The animation curve describes the relative frame rate over the course of the animation; predefined curves are linear, ease in (slow down near end), ease out (slowly speed up at start), and ease in-ease out (S-curve). Sending this message affects animations already in progress. The setting is ignored if the delegate implements `animationValueForProgress` (page 88). Invalid values throw an exception. See "Constants" (page 86) for valid constants.

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

setCurrentProgress

Sets the current progress of the receiver.

```
public void setCurrentProgress(float progress)
```

Discussion
The *progress* parameter should be a `float` value between 0.0 and 1.0; values that are out of range are pinned to 0.0 or 1.0. You can use this method to adjust the progress of a running animation. The NSAnimation class invokes this method while animation is running to change the progress for the next frame. Subclasses can override this method to get the latest value and perform their action with it, possibly in a secondary thread. Alternatively, you can implement the delegation method `animationValueForProgress` (page 88).

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

See Also
`currentProgress` (page 80)

setDelegate

Sets the delegate of the receiver.

```
public void setDelegate(Object delegate)
```

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

See Also
`delegate` (page 80)

setDuration

Sets the duration of the animation to a specified number of seconds.

```
public void setDuration(double duration)
```

Discussion

Negative values throw an exception. You can change the duration of an animation while it is running. However, setting the duration of a running animation to an interval shorter than the current progress ends the animation.

Availability

Available in Mac OS X v10.4 and later.

See Also

`duration` (page 81)

setFrameRate

Sets the frame rate—the number of updates per second—of the receiver.

```
public void setFrameRate(float framesPerSecond)
```

Discussion

The *framesPerSecond* value must be positive; negative values throw an exception. A frame rate of 0.0 means to go as fast as possible. The frame rate is not guaranteed due to differences among systems for the time needed to process a frame. You can change the frame rate while an animation is running and the new value is used at the next frame. The default frame rate is set to a reasonable value (which is subject to future change).

Availability

Available in Mac OS X v10.4 and later.

See Also

`frameRate` (page 81)

setProgressMarks

Sets the receiver's progress marks to the values specified in the passed-in array.

```
public void setProgressMarks(NSArray progressMarks)
```

Discussion

Each item of the *progressMarks* array should be an NSNumber object containing a `float` value. Passing in `null` clears all progress marks.

Availability

Available in Mac OS X v10.4 and later.

See Also

`progressMarks` (page 81)

startAnimation

Starts the animation represented by the receiver.

```
public void startAnimation()
```

Discussion

The receiver retains itself and is then autoreleased at the end of the animation or when it receives stopAnimation (page 85). If the blocking mode is AnimationBlocking, the method only returns after the animation has completed or the delegate sends it stopAnimation (page 85). If the receiver has a progress of 1.0, it starts again at 0.0.

Availability

Available in Mac OS X v10.4 and later.

See Also

startWhenAnimation (page 85)

stopAnimation (page 85)

startWhenAnimation

Starts running the animation represented by the receiver when another animation reaches a specific progress mark.

```
public void startWhenAnimation(NSAnimation animation, float startProgress)
```

Discussion

The *animation* parameter is the other NSAnimation object and the progress mark is specified in *startProgress*. This method links the running of two animations together. You can set only one NSAnimation object as a start animation and one as a stop animation at any one time. Setting a new start animation removes any animation previously set.

Availability

Available in Mac OS X v10.4 and later.

See Also

clearStartAnimation (page 79)

startAnimation (page 84)

stopWhenAnimation (page 86)

stopAnimation

Stops the animation represented by the receiver.

```
public void stopAnimation()
```

Discussion

The current progress of the receiver is not reset.

Availability

Available in Mac OS X v10.4 and later.

See Also

startAnimation (page 84)

stopWhenAnimation (page 86)

stopWhenAnimation

Stops running the animation represented by the receiver when another animation reaches a specific progress mark.

```
public void stopWhenAnimation(NSAnimation animation, float stopProgress)
```

Discussion
The *animation* parameter is the other NSAnimation object and the progress mark is specified in *stopProgress*. This method links the running of two animations together. You can set only one NSAnimation object as a start animation and one as a stop animation at any one time. Setting a new stop animation removes any animation previously set.

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

See Also
clearStopAnimation (page 79)
startWhenAnimation (page 85)
stopAnimation (page 85)

Constants

These constants describe the curve of an animation—that is, the relative speed of an animation from start to finish. You use them with setAnimationCurve (page 83). The following constants are available in Mac OS X v10.4 and later.

Constant	Description
AnimationEaseInOut	Describes an S-curve in which the animation slowly speeds up and then slows down near the end of the animation. This constant is the default.
AnimationEaseIn	Describes an animation that slows down as it reaches the end.
AnimationEaseOut	Describes an animation that slowly speeds up from the start.
AnimationLinear	Describes an animation in which there is no change in frame rate.

These constants indicate the blocking mode of an NSAnimation object when it is running. You specify one of these constants in setAnimationBlockingMode (page 82). The following constants are available in Mac OS X v10.4 and later.

Constant	Description
AnimationBlocking	Requests the animation to run in the main thread in a custom run-loop mode that blocks user input. This is the default.
AnimationNonblocking	Requests the animation to run in a standard or specified run-loop mode that allows user input.

Constant	Description
`AnimationNonblockingThreaded`	Requests the animation to run in a separate thread that is spawned by the NSAnimation object. (The secondary thread has its own run loop.)

The NSAnimation class also declares the string constant `AnimationProgressMark` for use as a key in accessing the current progress mark in the `userInfo` dictionary of the `AnimationProgressMarkNotification` (page 88) notification.

Delegate Methods

animationDidEnd

Sent to the delegate when the specified animation completes its run.

```
public abstract void animationDidEnd(NSAnimation animation)
```

Discussion
When an NSAnimation object reaches the end of its planned duration, it has a progress value of 1.0.

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

See Also
`animationDidEnd` (page 87)
`currentProgress` (page 80)

animationDidReachProgressMark

Sent to the delegate when an animation reaches a specific progress mark.

```
public abstract void animationDidReachProgressMark(NSAnimation animation, float
    progress)
```

Discussion
The delegate typically implements this method to perform some animation effect for the time slice indicated by *progress*, such as redrawing objects in a view with new coordinates or changing the frame location or size of a window or view. As an alternative to this delegation message, you may choose to observe the `AnimationProgressMarkNotification` (page 88) notification.

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

animationDidStop

Sent to the delegate when the specified animation is stopped before it completes its run.

```
public abstract void animationDidStop(NSAnimation animation)
```

Discussion
An NSAnimation object stops running when it receives a `stopAnimation` (page 85) message.

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

See Also
`animationDidEnd` (page 87)

animationShouldStart

Sent to the delegate just after the animation *animation* receives a `startAnimation` (page 84) message.

```
public abstract boolean animationShouldStart(NSAnimation animation)
```

Discussion
The delegate can use this method to prepare objects and resources for the effect. Return false to cancel the animation.

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

See Also
`animationDidEnd` (page 87)
`animationDidStop` (page 87)

animationValueForProgress

Sent to the delegate at each frame update, passing in a reference to the running *animation* animation and its current progress value *progress*.

```
public abstract float animationValueForProgress(NSAnimation animation, float
    progress)
```

Discussion
The delegate can compute and return a custom curve value for the given progress value. The value of *progress* is always between 0.0 and 1.0. If the delegate does not implement this method, NSAnimation computes the current curve value.

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

See Also
`currentValue` (page 80)

Notifications

AnimationProgressMarkNotification

Posted when the current progress of a running animation reaches one of its progress marks.

The notification object is the running NSAnimation object. The `userInfo` dictionary contains the current progress mark, accessed via the key `AnimationProgressMark`.

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

See Also
`animationDidReachProgressMark` (page 87)

NSAnimationEffect

Inherits from	Object
Package:	com.apple.cocoa.application
Availability	Available in Mac OS X v10.3 and later.

Overview

The NSAnimationEffect class provides system animations for use in your application. An animation effect includes both display and sound.

Tasks

Constructors

`NSAnimationEffect` (page 91)
    Creates an empty NSAnimationEffect.

Showing an Effect

`showEffect` (page 92)
    Runs one of the system animation effects, which includes display and sound.

Constructors

NSAnimationEffect

Creates an empty NSAnimationEffect.

```
public NSAnimationEffect()
```

Discussion
All of the NSAnimationEffect methods are static, so there is no need to create instances of the class.

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

Static Methods

showEffect

Runs one of the system animation effects, which includes display and sound.

```
public static void showEffect(int animationEffect, NSPoint centerLocation, NSSize
     size, Object animationDelegate, NSSelector didEndSelector, Object contextInfo)
```

Discussion

See "Constants" (page 92) for a description of possible values of *animationEffect*. *centerLocation* specifies the center of the animated image in screen coordinates; it indicates where to show the effect. The *size* parameter specifies how big the animated image should be; use NSSize.ZeroSize to get the default size. *animationDelegate* is an object that wants to know when the effect has completed; specify null if there is no animation delegate. If there is an animation delegate, this method invokes the method identified by *didEndSelector* when the animation completes. Whatever value or object is supplied in the *contextInfo* parameter is passed in the *didEndSelector*.

The *didEndSelector* method should have the following signature:

```
void animationEffectDidEnd (Object contextInfo)
```

Availability

Available in Mac OS X v10.3 and later.

Constants

NSAnimationEffect defines the following constants to specify the type of effect:

Constant	Description
DisappearingItemDefault	The default effect used to indicate removal of an item from a collection, such as toolbar (indicates removal, without destroying the underlying data). This is currently Poof, but may change in the future. If you always want to be guaranteed the animation effect will be a poof, use Poof.
Poof	An effect showing a puff of smoke.

NSApplication

Inherits from	NSResponder : NSObject
Implements	NSCoding (NSResponder)
Package:	com.apple.cocoa.application
Companion guides	Application Architecture Overview Notification Programming Topics for Cocoa Sheet Programming Topics for Cocoa System Services

Class at a Glance

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

Principal Attributes

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

Commonly Used Methods

`keyWindow` (page 113)
Returns an NSWindow representing the key window.

`mainWindow` (page 114)
Returns the application's main window.

`registerServicesMenuTypes` (page 117)
Specifies which services are valid for this application.

`runModalForWindow` (page 119)
Runs a modal event loop for the specified NSWindow.

Overview

The NSApplication class provides the central framework for your application's execution. Every application must have exactly one instance of NSApplication (or a subclass of NSApplication). Your program's `main()` function should create this instance by invoking the `sharedApplication` (page 105) class method. After creating the NSApplication object, the `main()` function should load your application's main nib file and then start the event loop by sending the NSApplication object a `run` (page 119) message. If you create an Application project in Xcode, this `main()` function is created for you. The `main()` function Xcode creates begins by calling a function named `NSApplicationMain()`.

The `sharedApplication` (page 105) class method initializes the display environment and connects your program to the window server and the display server. The NSApplication object maintains a list of all the NSWindows the application uses, so it can retrieve any of the application's NSViews. `sharedApplication` (page 105) only performs the initialization once; if you invoke it more than once, it simply returns the NSApplication object it created previously.

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

NSApplication is also responsible for dispatching certain Apple events received by the application. For example, the Mac OS sends Apple events to your application at various times, such as when the application is launched or reopened. NSApplication installs Apple event handlers to handle these events by sending a message to the appropriate object. You can also use the NSAppleEventManager class to register your own Apple event handlers. The `applicationWillFinishLaunching` (page 137) method is generally the best place to do so. For more information on how events are handled and how you can modify the default behavior, including information on working with Apple events in scriptable applications, see "How Cocoa Applications Handle Apple Events." in *Scriptable Application Programming Guide for Cocoa*.

The NSApplication class sets up autorelease pools (instances of the NSAutoreleasePool class) during initialization and inside the event loop—specifically, within its initialization (or `sharedApplication` (page 105)) and `run` (page 119) methods. Similarly, the methods the Application Kit adds to NSBundle employ autorelease pools during the loading of nib files. These autorelease pools aren't accessible outside the scope of the respective NSApplication and NSBundle methods. Typically, an application creates objects either while the event loop is running or by loading objects from nib files, so this lack of access usually isn't a problem. However, if you do need to use Cocoa classes within the `main()` function itself (other than to load nib files or to instantiate NSApplication), you should create an autorelease pool before using the classes and then release the pool when you're done. For more information, see NSAutoreleasePool in the *Foundation Framework Reference*.

The Delegate and Notifications

You can assign a delegate to NSApplication.sharedApplication(). The delegate responds to certain messages on behalf of NSApplication.sharedApplication(). Some of these messages, such as applicationOpenFile (page 133), ask the delegate to perform an action. Another message, applicationShouldTerminate (page 136), lets the delegate determine whether the application should be allowed to quit. The NSApplication class sends these messages directly to its delegate.

The NSApplication.sharedApplication() also posts notifications to the application's default notification center. Any object may register to receive one or more of the notifications posted by NSApplication.sharedApplication() by sending the message addObserver to the default notification center (an instance of the NSNotificationCenter class). The delegate of NSApplication.sharedApplication() is automatically registered to receive these notifications if it implements certain delegate methods. For example, NSApplication.sharedApplication() posts notifications when it is about to be done launching the application and when it is done launching the application (ApplicationWillFinishLaunchingNotification (page 141) and ApplicationDidFinishLaunchingNotification (page 140)). The delegate has an opportunity to respond to these notifications by implementing the methods applicationWillFinishLaunching (page 137) and applicationDidFinishLaunching (page 131). If the delegate wants to be informed of both events, it implements both methods. If it needs to know only when the application is finished launching, it implements only applicationDidFinishLaunching (page 131).

System Services

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

Subclassing Notes

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

To use a custom subclass of NSApplication, simply send sharedApplication (page 105) to your subclass rather than directly to NSApplication. If you create your application in Xcode, you can accomplish this by setting your custom application class to be the principal class. In Xcode, double-click the application target in the Groups and Files list to open the Info window for the target. Then display the Properties pane of the window and replace "NSApplication" in the Principal Class field with the name of your custom class. TheNSApplicationMain function sends sharedApplication (page 105) to the principal class to obtain the global application instance (NSApp)—which in this case will be an instance of your custom subclass of NSApplication.

Methods to Override

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

- Override `run` (page 119) if you want the application to manage the main event loop differently than it does by default. (This a critical and complex task, however, that you should only attempt with good reason.)
- Override `sendEvent` (page 121) if you want to change how events are dispatched or perform some special event processing.
- Override `requestUserAttention` (page 118) if you want to modify how your application attracts the attention of the user (for example, offering an alternative to the bouncing application icon in the Dock).
- Override `targetForAction` (page 125) to substitute another object for the target of an action message.

Special Considerations

The global application object uses autorelease pools in its `run` (page 119) method; if you override this method, you'll need to create your own autorelease pools.

Do not override `sharedApplication` (page 105). The default implementation, which is essential to application behavior, is too complex to duplicate on your own.

Alternatives to Subclassing

NSApplication defines over twenty delegation methods that offer opportunities for modifying specific aspects of application behavior. Instead of making a custom subclass of NSApplication, your application delegate may be able to implement one or more of these methods to accomplish your design goals.

In general, a better design than subclassing NSApplication is to put the code that expresses your application's special behavior into a number of custom objects called controllers. (Controller objects usually inherit directly from NSObject.) Methods defined in your controllers can be invoked from a small dispatcher object without being closely tied to the global application object.

Tasks

Constructors

`NSApplication` (page 104)

Creating and Initializing an NSApplication

`sharedApplication` (page 105)
Returns the NSApplication instance, creating it if it doesn't exist yet.

`finishLaunching` (page 112)
Activates the receiver, opens any files specified by the `NSOpen` user default, and unhighlights the application's icon.

Getting Information About the Framework

`appkitVersionNumber` (page 104)
: Returns the version number for the Application Kit framework.

Changing the Active Application

`activateIgnoringOtherApps` (page 106)
: Makes the receiver the active application.

`hideOtherApplications` (page 112)
: Hides all applications, except the receiver.

`unhideAllApplications` (page 126)
: Unhides all applications, including the receiver.

`isActive` (page 112)
: Returns `true` if this is the active application, `false` otherwise.

`deactivate` (page 110)
: Deactivates the receiver.

Running the Event Loop

`run` (page 119)
: Starts the main event loop.

`isRunning` (page 113)
: Returns `true` if the main event loop is running, `false` otherwise.

`stop` (page 124)
: Stops the main event loop.

`runModalForWindow` (page 119)
: Starts a modal event loop for *aWindow*.

`stopModal` (page 124)
: Stops a modal event loop.

`stopModalWithCode` (page 124)
: Like `stopModal` (page 124), except the argument *returnCode* allows you to specify the value `runModalForWindow` (page 119) will return.

`abortModal` (page 106)
: Aborts the event loop started by `runModalForWindow` (page 119) or `runModalSession` (page 120).

`beginModalSessionForWindow` (page 108)
: Sets up a modal session with the NSWindow *aWindow* and returns an NSModalSession structure representing the session.

`runModalSession` (page 120)
: Runs a modal session represented by *session*, as defined in a previous invocation of `beginModalSessionForWindow` (page 108).

`endModalSession` (page 111)
: Finishes a modal session.

`sendEvent` (page 121)
Dispatches *anEvent* to other objects.

`modalWindow` (page 115)
Returns the modal window that the receiver is displaying.

Getting, Removing, and Posting Events

`currentEvent` (page 110)
Returns the current event, the last event the receiver retrieved from the event queue.

`nextEventMatchingMask` (page 115)
Returns the next event matching *mask*, or `null` if no such event is found before the expiration date specified by *expiration*.

`discardEventsMatchingMask` (page 110)
Removes all events matching *mask* and generated before *lastEvent* from the event queue.

`postEvent` (page 117)
Adds *anEvent* to the receiver's event queue.

Managing Sheets

`beginSheet` (page 108)
Starts a document modal session.

`endSheet` (page 111)
Ends a document modal session by specifying the sheet window, *sheet*.

Managing Windows

`keyWindow` (page 113)
Returns the key window, the NSWindow that receives keyboard events.

`mainWindow` (page 114)
Returns the main window.

`windowWithWindowNumber` (page 128)
Returns the NSWindow object corresponding to *windowNum*.

`windows` (page 128)
Returns an NSArray of the receiver's NSWindows, including offscreen windows.

`makeWindowsPerform` (page 114)
Sends the *aSelector* message to each NSWindow in the application in turn until one returns a value other than `null`.

`setWindowsNeedUpdate` (page 123)
Sets whether the receiver's windows need updating when the receiver has finished processing the current event, depending on the Boolean value passed in place of *flag*.

`updateWindows` (page 127)
Sends an `update` (page 1871) message to each onscreen NSWindow.

`miniaturizeAll` (page 114)
: Miniaturizes all the receiver's windows.

`preventWindowOrdering` (page 117)
: Suppresses the usual window ordering in handling the most recent mouse-down event.

Hiding All Windows

`hide` (page 112)
: Hides all the receiver's windows, and the next application in line is activated.

`isHidden` (page 113)
: Returns `true` if the receiver is hidden, `false` otherwise.

`unhide` (page 126)
: Restores hidden windows to the screen and makes the receiver active.

`unhideWithoutActivation` (page 127)
: Restores hidden windows without activating their owner (the receiver).

Setting the Application's Icon

`setApplicationIconImage` (page 122)
: Sets the receiver's icon to *`anImage`*. This method updates the dock application tile. *`anImage`* will be scaled as necessary to fit the tile.

`applicationIconImage` (page 107)
: Returns the NSImage used for the receiver's icon.

Getting the Main Menu

`setMainMenu` (page 122)
: Makes *`aMenu`* the receiver's main menu.

`mainMenu` (page 113)
: Returns the receiver's main menu.

Managing the Window Menu

`setWindowsMenu` (page 123)
: Makes *`aMenu`* the receiver's Window menu.

`windowsMenu` (page 128)
: Returns the Window menu or `null` if no Window menu has been created.

`arrangeInFront` (page 107)
: Arranges windows listed in the Window menu in front of all other windows.

`addWindowsItem` (page 107)
: Adds an item to the Window menu for *`aWindow`*.

`changeWindowsItem` (page 109)
: Changes the item for *`aWindow`* in the Window menu to *`aString`*.

`removeWindowsItem` (page 118)
Removes the Window menu item for *aWindow*.

`updateWindowsItem` (page 127)
Updates the Window menu item for *aWindow* to reflect the edited status of *aWindow*.

Managing the Services Menu

`setServicesMenu` (page 123)
Makes *aMenu* the receiver's Services menu.

`servicesMenu` (page 121)
Returns the Services menu, or `null` if no Services menu has been created.

`registerServicesMenuTypes` (page 117)
Registers the pasteboard types the receiver can send and receive in response to service requests.

`validRequestorForTypes` (page 128)
Indicates whether the receiver can send and receive the specified pasteboard types.

`setServicesProvider` (page 123)
Registers the object *aProvider* as the service provider.

`servicesProvider` (page 122)
Returns the object that provides the services the receiver advertises in the Services menu of other applications.

Showing Standard Panels

`orderFrontColorPanel` (page 116)
Brings up the color panel, an instance of NSColorPanel.

`orderFrontStandardAboutPanel` (page 116)
Displays a standard About window.

`orderFrontStandardAboutPanelWithOptions` (page 116)
Displays a standard About window with information from *optionsDictionary*.

`orderFrontCharacterPalette` (page 115)
Opens the character palette.

`runPageLayout` (page 120)
Displays the receiver's page layout panel, an instance of NSPageLayout.

`showSystemInfoPanel` (page 105)
Deprecated. Do not use.

Displaying Help

`showHelp` (page 123)
If your project is properly registered, and the necessary keys have been set in the property list, this method launches Help Viewer and displays the first page of your application's help book.

`activateContextHelpMode` (page 106)
Places the receiver in context-sensitive help mode.

Sending Action Messages

sendActionToTargetFromSender (page 121)
Sends the message *anAction* to *aTarget*.

tryToPerform (page 126)
Dispatches action messages.

targetForAction (page 125)
Returns the object that receives the action message *aSelector*.

targetForActionToFrom (page 125)
Finds an object that can receive the message specified by the selector *anAction*.

Getting the Display Context

context (page 109)
Returns the receiver's display context.

Reporting an Exception

reportException (page 118)
Logs *anException*.

Terminating the Application

replyToApplicationShouldTerminate (page 118)
If an application delegate returns TerminateLater to applicationShouldTerminate (page 136), this method must be called with *shouldTerminate* passed as true or false once the application decides if it can terminate.

terminate (page 125)
Terminates the receiver.

Assigning a Delegate

setDelegate (page 122)
Makes *anObject* the receiver's delegate.

delegate (page 110)
Returns the receiver's delegate.

Handling User Attention Requests

cancelUserAttentionRequest (page 109)
Cancels a previous user attention request.

requestUserAttention (page 118)
Starts a user attention request.

`beep` (page 104)
Plays the system beep.

`replyToOpenOrPrint` (page 118)
Handles errors that might occur when the user attempts to open or print files.

Loading Nib Files

`loadNibFromBundle` (page 105)
Unarchives the contents of the nib file named *fileName* in *aBundle*, using *owner* as the nib file's owner (shown as "File's Owner" in Interface Builder).

`loadNibNamed` (page 105)
Unarchives the contents of the nib file named *aNibName*, using *owner* as the nib file's owner (shown as "File's Owner" in Interface Builder).

Opening files

`applicationOpenFiles` (page 133) *delegate method*
Identical to `applicationOpenFile` (page 133) except that the receiver opens multiple files corresponding to the file names in the *filenames* array.

`applicationOpenFile` (page 133) *delegate method*

`applicationOpenFileWithoutUI` (page 133) *delegate method*

`applicationOpenTempFile` (page 134) *delegate method*

`applicationOpenUntitledFile` (page 134) *delegate method*

`applicationShouldOpenUntitledFile` (page 136) *delegate method*

Printing

`applicationPrintFiles` (page 135) *delegate method*
Identical to `applicationPrintFile` (page 134) except that the receiver prints multiple files corresponding to the file names in the *filenames* array.

`applicationPrintFile` (page 134) *delegate method*

`applicationPrintFiles` (page 135) *delegate method*
Prints a group of files.

Supplying a dock menu

`applicationDockMenu` (page 132) *delegate method*
Allows the delegate to supply a dock menu for the application dynamically.

Activating, launching, and updating an application

`applicationDidBecomeActive` (page 130) *delegate method*

`applicationWillBecomeActive` (page 137) *delegate method*

`applicationDidResignActive` (page 131) *delegate method*

`applicationWillResignActive` (page 138) *delegate method*

`applicationDidFinishLaunching` (page 131) *delegate method*

`applicationWillFinishLaunching` (page 137) *delegate method*

`applicationDidChangeScreenParameters` (page 131) *delegate method*

`applicationDidUpdate` (page 132) *delegate method*

`applicationWillUpdate` (page 139) *delegate method*

`applicationShouldHandleReopen` (page 136) *delegate method*
Sent by *theApplication* to the delegate prior to default behavior to reopen (`rapp`) AppleEvents.

Hiding and unhiding an application

`applicationDidHide` (page 131) *delegate method*

`applicationWillHide` (page 138) *delegate method*

`applicationDidUnhide` (page 132) *delegate method*

`applicationWillUnhide` (page 139) *delegate method*

Terminating an application

`applicationShouldTerminate` (page 136) *delegate method*

`applicationShouldTerminateAfterLastWindowClosed` (page 137) *delegate method*
Invoked when the user closes the last window the application has open.

`applicationWillTerminate` (page 139) *delegate method*

Handling errors

`applicationWillPresentError` (page 138) *delegate method*
Sent to the delegate before *`application`* presents an error message, based on the information in *`error`*, to the user.

Constructors

NSApplication

```
public NSApplication()
```

Discussion
The constructor for the NSApplication object. You should not invoke this constructor directly; instead, use the `sharedApplication` (page 105) method.

Static Methods

appkitVersionNumber

Returns the version number for the Application Kit framework.

```
public static double appkitVersionNumber()
```

beep

Plays the system beep.

```
public static void beep()
```

Discussion
A user can select a sound to be played. On a Macintosh computer, for example, the user chooses a sound with the Sound control panel.

The implementation of this method calls the C function `NSBeep`.

loadNibFromBundle

Unarchives the contents of the nib file named `fileName` in `aBundle`, using `owner` as the nib file's owner (shown as "File's Owner" in Interface Builder).

```
public static boolean loadNibFromBundle(NSBundle aBundle, String fileName, Object
     owner)
```

Discussion

The method first looks for the nib file in the language-specific "`.lproj`" directory of `aBundle`. If `fileName` isn't there, this method looks for a non-localized resource in the immediate bundle directory. Returns `true` upon success, or `false` if the specified nib file couldn't be loaded.

loadNibNamed

Unarchives the contents of the nib file named `aNibName`, using `owner` as the nib file's owner (shown as "File's Owner" in Interface Builder).

```
public static boolean loadNibNamed(String aNibName, Object owner)
```

Discussion

The argument `aNibName` need not include the "`.nib`" extension. If there's a bundle for the class of `owner`, this method looks in that bundle for `aNibName`; otherwise, it looks in the main bundle. Returns `true` upon success, or `false` if the specified nib file couldn't be loaded.

sharedApplication

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

```
public static NSApplication sharedApplication()
```

Discussion

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

See Also

`run` (page 119)

`terminate` (page 125)

showSystemInfoPanel

Deprecated. Do not use.

```
public static void showSystemInfoPanel(NSDictionary aDictionary)
```

Instance Methods

abortModal

Aborts the event loop started by `runModalForWindow` (page 119) or `runModalSession` (page 120).

```
public void abortModal()
```

Discussion

When stopped with this method, `runModalForWindow` and `runModalSession` return `RunAbortedResponse`.

`abortModal` must be used instead of `stopModal` (page 124) or `stopModalWithCode` (page 124) when you need to stop a modal event loop from anywhere other than a callout from that event loop. In other words, if you want to stop the loop in response to a user's actions within the modal window, use `stopModal`; otherwise, use `abortModal`. For example, use `abortModal` when running in a different thread from the Application Kit's main thread or when responding to an NSTimer that you have added to the `ModalPanelRunLoopMode` mode of the default NSRunLoop.

See Also

`endModalSession` (page 111)

activateContextHelpMode

Places the receiver in context-sensitive help mode.

```
public void activateContextHelpMode(Object sender)
```

Discussion

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

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

See Also

`showHelp` (page 123)

activateIgnoringOtherApps

Makes the receiver the active application.

```
public void activateIgnoringOtherApps(boolean flag)
```

Discussion

If *`flag`* is `false`, the application is activated only if no other application is currently active. If *`flag`* is `true`, the application activates regardless.

The *flag* is normally set to `false`. When the Finder launches an application, using a value of `false` for *flag* allows the application to become active if the user waits for it to launch, but the application remains unobtrusive if the user activates another application. Regardless of the setting of *flag*, there may be a time lag before the application activates—you should not assume the application will be active immediately after sending this message.

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

You don't need to send this message to make one of the application's NSWindows key. When you send a `makeKeyWindow` (page 1842) message to an NSWindow, you ensure the NSWindow will be the key window when the application is active.

See Also

`deactivate` (page 110)

`isActive` (page 112)

addWindowsItem

Adds an item to the Window menu for *aWindow*.

```
public void addWindowsItem(NSWindow aWindow, String aString, boolean isFilename)
```

Discussion

If *isFilename* is `false`, *aString* appears literally in the menu. If *isFilename* is `true`, *aString* is assumed to be a converted pathname with the name of the file preceding the path (the way NSWindow's `setTitleWithRepresentedFilename` (page 1868) method shows a title). If an item for *aWindow* already exists in the Window menu, this method has no effect. You rarely invoke this method because an item is placed in the Window menu for you whenever an NSWindow's title is set.

See Also

`changeWindowsItem` (page 109)

`setTitle` (page 1868) (NSWindow)

applicationIconImage

Returns the NSImage used for the receiver's icon.

```
public NSImage applicationIconImage()
```

See Also

`setApplicationIconImage` (page 122)

arrangeInFront

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

```
public void arrangeInFront(Object sender)
```

Discussion

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

See Also

`addWindowsItem` (page 107)

`removeWindowsItem` (page 118)

`makeKeyAndOrderFront` (page 1841) (NSWindow)

beginModalSessionForWindow

Sets up a modal session with the NSWindow *aWindow* and returns an NSModalSession structure representing the session.

```
public NSModalSession beginModalSessionForWindow(NSWindow aWindow)
```

Discussion

In a modal session, the application receives mouse events only if they occur in *aWindow*. The NSWindow is made key and ordered to the front.

The `beginModalSessionForWindow` method only sets up the modal session. To actually run the session, use `runModalSession` (page 120). `beginModalSessionForWindow` should be balanced by `endModalSession` (page 111). Make sure these two messages are sent within the same exception-handling scope. That is, if you send `beginModalSessionForWindow` inside an `NS_DURING` construct, you must send `endModalSession` before `NS_ENDHANDLER`.

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

A loop using these methods is similar to a modal event loop run with `runModalForWindow` (page 119), except the application can continue processing between method invocations.

This method has been deprecated. Use `beginSheet` (page 108) instead.

```
public NSModalSession beginModalSessionForWindow(NSWindow theWindow, NSWindow
    docWindow)
```

beginSheet

Starts a document modal session.

```
public void beginSheet(NSWindow sheet, NSWindow docWindow, Object modalDelegate,
    NSSelector didEndSelector, Object contextInfo)
```

Discussion

The *didEndSelector* method is optional. If implemented by the *modalDelegate*, this method is invoked after the modal session has ended and is passed a return code and caller specified in *contextInfo*. *didEndSelector* should have the following signature:

```
public void sheetDidEnd (NSWindow sheet, int returnCode, Object  contextInfo)
```

Use this method in cases where you do not need to do any additional background processing while your sheet runs. This method consumes only enough CPU time to process events and dispatch them to the action methods associated with the sheet. If you want to perform additional background processing, use `runModalSession` (page 120) together with an NSModalSession object instead.

See Also

`endSheet` (page 111)

Starts a document modal session.

```
public void beginSheet(NSWindow sheet, NSWindow docWindow, Object modalDelegate,
    NSSelector didEndSelector)
```

Discussion

The *didEndSelector* method is optional. If implemented by the *modalDelegate*, this method is invoked after the modal session has ended.

cancelUserAttentionRequest

Cancels a previous user attention request.

```
public void cancelUserAttentionRequest(int request)
```

Discussion

request is the return value from a previous call to `requestUserAttention` (page 118). A request is also canceled automatically by user activation of the application.

changeWindowsItem

Changes the item for *aWindow* in the Window menu to *aString*.

```
public void changeWindowsItem(NSWindow aWindow, String aString, boolean isFilename)
```

Discussion

If *aWindow* doesn't have an item in the Window menu, this method adds the item. If *isFilename* is `false`, *aString* appears literally in the menu. If *isFilename* is `true`, *aString* is assumed to be a converted pathname with the file's name preceding the path (the way NSWindow's `setTitleWithRepresentedFilename` (page 1868) places a title).

See Also

`addWindowsItem` (page 107)

`removeWindowsItem` (page 118)

`setTitle` (page 1868) (NSWindow)

context

Returns the receiver's display context.

```
public NSGraphicsContext context()
```

currentEvent

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

```
public NSEvent currentEvent()
```

Discussion
`NSApplication.sharedApplication()` receives events and forwards the current event to the affected NSWindow object, which then distributes it to the objects in its view hierarchy.

See Also
`discardEventsMatchingMask` (page 110)
`postEvent` (page 117)
`sendEvent` (page 121)

deactivate

Deactivates the receiver.

```
public void deactivate()
```

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

See Also
`activateIgnoringOtherApps` (page 106)

delegate

Returns the receiver's delegate.

```
public Object delegate()
```

See Also
`setDelegate` (page 122)

discardEventsMatchingMask

Removes all events matching *mask* and generated before *lastEvent* from the event queue.

```
public void discardEventsMatchingMask(int mask, NSEvent lastEvent)
```

Discussion
Typically, you send this message to an NSWindow rather than to `NSApplication.sharedApplication()`.

mask can contain these constants:

```
NSEvent.LeftMouseDownMask
NSEvent.LeftMouseUpMask
NSEvent.RightMouseDownMask
NSEvent.RightMouseUpMask
NSEvent.MouseMovedMask
NSEvent.LeftMouseDraggedMask
NSEvent.RightMouseDraggedMask
NSEvent.MouseEnteredMask
NSEvent.MouseExitedMask
NSEvent.KeyDownMask
NSEvent.KeyUpMask
NSEvent.FlagsChangedMask
NSEvent.PeriodicMask
NSEvent.CursorUpdateMask
NSEvent.AnyEventMask
```

Use this method to ignore events that occur before a particular kind of event. For example, suppose your application has a tracking loop that you exit when the user releases the mouse button, and you want to discard all events that occurred during that loop. You use `NSEvent.AnyEventMask` as the mask argument and pass the mouse-up event as the *lastEvent* argument. Passing the mouse-up event as *lastEvent* ensures that any events that might have occurred after the mouse-up event (that is, that appear in the queue after the mouse-up event) don't get discarded.

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

See Also
`nextEventMatchingMask` (page 115)

endModalSession

Finishes a modal session.

```
public void endModalSession(NSModalSession session)
```

Discussion
session should be the return value from a previous invocation of `beginModalSessionForWindow` (page 108).

See Also
`runModalSession` (page 120)

endSheet

Ends a document modal session by specifying the sheet window, *sheet*.

```
public void endSheet(NSWindow sheet)
```

Ends a document modal session by specifying the sheet window, *sheet*.

```
public void endSheet(NSWindow sheet, int returnCode)
```

Discussion
Also passes along a *returnCode* to the delegate.

See Also

beginSheet (page 108)

finishLaunching

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

```
public void finishLaunching()
```

Discussion

The run (page 119) method invokes this method before it starts the event loop. When this method begins, it posts an ApplicationWillFinishLaunchingNotification (page 141) to the default notification center. If you override finishLaunching (page 112), the subclass method should invoke the superclass method.

See Also

applicationWillFinishLaunching (page 137)

applicationDidFinishLaunching (page 131)

hide

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

```
public void hide(Object sender)
```

Discussion

This method is usually invoked when the user chooses Hide in the application's main menu. When this method begins, it posts an ApplicationWillHideNotification (page 141) to the default notification center. When it completes successfully, it posts an ApplicationDidHideNotification (page 140).

See Also

miniaturizeAll (page 114)

unhide (page 126)

unhideWithoutActivation (page 127)

applicationDidHide (page 131)

applicationWillHide (page 138)

hideOtherApplications

Hides all applications, except the receiver.

```
public void hideOtherApplications(Object sender)
```

isActive

Returns true if this is the active application, false otherwise.

```
public boolean isActive()
```

See Also

`activateIgnoringOtherApps` (page 106)

`deactivate` (page 110)

isHidden

Returns `true` if the receiver is hidden, `false` otherwise.

```
public boolean isHidden()
```

See Also

`hide` (page 112)

`unhide` (page 126)

`unhideWithoutActivation` (page 127)

isRunning

Returns `true` if the main event loop is running, `false` otherwise.

```
public boolean isRunning()
```

Discussion

`false` means the `stop` (page 124) method was invoked.

See Also

`run` (page 119)

`terminate` (page 125)

keyWindow

Returns the key window, the NSWindow that receives keyboard events.

```
public NSWindow keyWindow()
```

Discussion

This method returns `null` if there is no key window, if the application's nib file hasn't finished loading yet, or if the receiver is not active.

See Also

`mainWindow` (page 114)

`isKeyWindow` (page 1838) (NSWindow)

mainMenu

Returns the receiver's main menu.

```
public NSMenu mainMenu()
```

See Also

`setMainMenu` (page 122)

mainWindow

Returns the main window.

```
public NSWindow mainWindow()
```

Discussion

This method returns `null` if there is no main window, if the application's nib file hasn't finished loading, if the receiver is not active, or if the application is hidden.

See Also

`keyWindow` (page 113)

`isMainWindow` (page 1838) (NSWindow)

makeWindowsPerform

Sends the *aSelector* message to each NSWindow in the application in turn until one returns a value other than `null`.

```
public NSWindow makeWindowsPerform(NSSelector aSelector, boolean flag)
```

Discussion

Returns that NSWindow, or `null` if all NSWindows returned `null` for *aSelector*.

If *flag* is `true`, the *aSelector* message is sent to each of the window server's on-screen windows, going in z-order, until one returns non-nil. A minimized window is not considered to be onscreen for this check. If *flag* is `false`, the message is sent to all windows in `NSApplication.sharedApplication()`'s window list, regardless of whether they are on-screen or not. This order is unspecified.

The method designated by *aSelector* can't take any arguments.

See Also

`sendActionToTargetFromSender` (page 121)

`tryToPerform` (page 126)

`windows` (page 128)

miniaturizeAll

Miniaturizes all the receiver's windows.

```
public void miniaturizeAll(Object sender)
```

See Also

`hide` (page 112)

modalWindow

Returns the modal window that the receiver is displaying.

```
public NSWindow modalWindow()
```

Discussion

If the application isn't displaying a modal window, this method returns `null`.

This method does not return a sheet window. If you need access to a sheet window, use NSWindow's `attachedSheet` (page 1818).

nextEventMatchingMask

Returns the next event matching *mask*, or `null` if no such event is found before the expiration date specified by *expiration*.

```
public NSEvent nextEventMatchingMask(int mask, NSDate expiration, String mode,
    boolean flag)
```

Discussion

A value of `null` for *expiration* is equivalent to `distantPast`. If *flag* is `true`, the event is removed from the queue. See the method description for `discardEventsMatchingMask` (page 110) for a list of the possible values for *mask*.

The *mode* argument names an NSRunLoop mode that determines what other ports are listened to and what timers may fire while `NSApplication.sharedApplication()` is waiting for the event. The possible modes available in the Application Kit are:

```
NSRunLoop.DefaultRunLoopMode
NSApplication.EventTrackingRunLoopMode
NSApplication.ModalPanelRunLoopMode
```

Events that are skipped are left in the queue.

You can use this method to short circuit normal event dispatching and get your own events. For example, you may want to do this in response to a mouse-down event in order to track the mouse while its button is down. In this case, you would set *mask* to accept mouse-dragged or mouse-up events and use the `NSApplication.EventTrackingRunLoopMode`.

See Also

`postEvent` (page 117)

`run` (page 119)

`runModalForWindow` (page 119)

orderFrontCharacterPalette

Opens the character palette.

```
public void orderFrontCharacterPalette(Object sender)
```

Availability

Available in Mac OS X v10.3 and later.

orderFrontColorPanel

Brings up the color panel, an instance of NSColorPanel.

```
public void orderFrontColorPanel(Object sender)
```

Discussion

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

orderFrontStandardAboutPanel

Displays a standard About window.

```
public void orderFrontStandardAboutPanel(Object sender)
```

Discussion

This method calls `orderFrontStandardAboutPanelWithOptions` (page 116) with a `null` argument. See `orderFrontStandardAboutPanelWithOptions` for a description of what's displayed.

orderFrontStandardAboutPanelWithOptions

Displays a standard About window with information from *optionsDictionary*.

```
public void orderFrontStandardAboutPanelWithOptions(NSDictionary optionsDictionary)
```

Discussion

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

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

See Also

`orderFrontStandardAboutPanel` (page 116)

postEvent

Adds *anEvent* to the receiver's event queue.

```
public void postEvent(NSEvent anEvent, boolean flag)
```

Discussion

If *flag* is `true`, the event is added to the front of the queue; otherwise the event is added to the back of the queue.

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

See Also

`currentEvent` (page 110)

`sendEvent` (page 121)

preventWindowOrdering

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

```
public void preventWindowOrdering()
```

Discussion

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

registerServicesMenuTypes

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

```
public void registerServicesMenuTypes(NSArray sendTypes, NSArray returnTypes)
```

Discussion

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

See Also

`validRequestorForTypes` (page 128)

removeWindowsItem

Removes the Window menu item for *aWindow*.

```
public void removeWindowsItem(NSWindow aWindow)
```

Discussion

This method doesn't prevent the item from being automatically added again. Use NSWindow's setExcludedFromWindowsMenu (page 1860) method if you want the item to remain excluded from the Window menu.

See Also

addWindowsItem (page 107)

changeWindowsItem (page 109)

replyToApplicationShouldTerminate

If an application delegate returns TerminateLater to applicationShouldTerminate (page 136), this method must be called with *shouldTerminate* passed as true or false once the application decides if it can terminate.

```
public replyToApplicationShouldTerminate(boolean shouldTerminate)
```

replyToOpenOrPrint

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

```
public void replyToOpenOrPrint(int reply)
```

Discussion

Delegates should invoke this method if an error is encountered in the applicationOpenFiles (page 133) or applicationPrintFiles (page 135) delegate methods. Possible values for *reply* are described in "Constants" (page 128).

Availability

Available in Mac OS X v10.3 and later.

reportException

Logs *anException*.

```
public void reportException(Throwable anException)
```

Discussion

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

requestUserAttention

Starts a user attention request.

```
public int requestUserAttention(int requestType)
```

Discussion

`requestType` is one of the values described in "Constants" (page 128). Activating the application cancels the user attention request. A spoken notification will occur if spoken notifications are enabled. Sending `requestUserAttention` to an application that is already active has no effect.

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

The value returned by this method can be used to manually cancel a request by passing it as the parameter to `cancelUserAttentionRequest` (page 109).

run

Starts the main event loop.

```
public void run()
```

Discussion

The loop continues until a `stop` (page 124) or `terminate` (page 125) message is received. Upon each iteration through the loop, the next available event from the window server is stored and then dispatched by sending it to `NSApplication.sharedApplication()` using `sendEvent` (page 121).

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

See Also

`runModalForWindow` (page 119)

`runModalSession` (page 120)

`applicationDidFinishLaunching` (page 131)

runModalForWindow

Starts a modal event loop for *aWindow*.

```
public int runModalForWindow(NSWindow aWindow)
```

Discussion

Until the loop is broken by a `stopModal` (page 124), `stopModalWithCode` (page 124), or `abortModal` (page 106) message, the application won't respond to any mouse, keyboard, or window-close events unless they're associated with *aWindow*. 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 `RunStoppedResponse`. If `abortModal` is used, it returns the constant `RunAbortedResponse`.

The window *aWindow* is placed on the screen using NSWindow's `center` (page 1821) method, if not already visible, and made key as a result of the `runModalForWindow` message. Do not send NSWindow's `makeKeyAndOrderFront` (page 1841) to *aWindow*.

See Also

`run` (page 119)

runModalSession (page 120)

This method has been deprecated. Use beginSheet (page 108) instead.

```
public int runModalForWindow(NSWindow theWindow, NSWindow docWindow)
```

runModalSession

Runs a modal session represented by *session*, as defined in a previous invocation of beginModalSessionForWindow (page 108).

```
public int runModalSession(NSModalSession session)
```

Discussion

A loop that uses this method is similar in some ways to a modal event loop run with runModalForWindow (page 119), except with this method your code can do some additional work 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 in your loop that the window remains responsive to events. However, you should not invoke this method in a tight loop because it returns immediately if there are no events, and consequently you could end up polling for events rather than blocking.

Typically, you use this method in situations where you want to do some additional background processing while the modal loop runs. For example, while processing a large data set, you might want to use a modal dialog to display progress and give the user a chance to cancel the operation. If you want to display a modal dialog and do not need to do any additional work in parallel, use runModalForWindow (page 119) instead. When there are no pending events, that method waits idly instead of consuming CPU time.

If the modal session was not stopped, this method returns RunContinuesResponse. At this point, your application can do some work before the next invocation of runModalSession (page 120) (as indicated in the example's doSomeWork call). If stopModal (page 124) was invoked as the result of event processing, runModalSession (page 120) returns RunStoppedResponse. If stopModalWithCode (page 124) was invoked, this method returns the value passed to stopModalWithCode. If abortModal (page 106) was invoked, this method returns RunAbortedResponse.

The window is placed on the screen and made key as a result of the runModalSession message. Do not send a separate makeKeyAndOrderFront (page 1841) message.

See Also

endModalSession (page 111)

run (page 119)

runPageLayout

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

```
public void runPageLayout(Object sender)
```

Discussion

If the NSPageLayout instance does not exist, this method creates one. This method is typically invoked when the user selects Page Layout from the application's menu.

sendActionToTargetFromSender

Sends the message *anAction* to *aTarget*.

```
public boolean sendActionToTargetFromSender(NSSelector anAction, Object aTarget,
    Object sender)
```

Discussion

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

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

Returns true if the action is successfully sent, otherwise returns false.

See Also

targetForAction (page 125)

tryToPerform (page 126)

makeWindowsPerform (page 114)

sendEvent

Dispatches *anEvent* to other objects.

```
public void sendEvent(NSEvent anEvent)
```

Discussion

You rarely invoke sendEvent directly, although you might want to override this method to perform some action on every event. sendEvent messages are sent from the main event loop (the run (page 119) method). sendEvent is the method that dispatches events to the appropriate responders—NSApplication.sharedApplication() 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.

See Also

currentEvent (page 110)

postEvent (page 117)

servicesMenu

Returns the Services menu, or null if no Services menu has been created.

```
public NSMenu servicesMenu()
```

See Also
`setServicesMenu` (page 123)

servicesProvider

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

```
public Object servicesProvider()
```

See Also
`setServicesProvider` (page 123)

setApplicationIconImage

Sets the receiver's icon to *anImage*. This method updates the dock application tile. *anImage* will be scaled as necessary to fit the tile.

```
public void setApplicationIconImage(NSImage anImage)
```

Discussion
The following code sample shows how to get the application's icon image and then restore the dock's image:

```
myImage = NSImage.imageNamed("NSApplicationIcon");
NSApplication.sharedApplication().setApplicationIconImage(myImage);
```

See Also
`applicationIconImage` (page 107)

setDelegate

Makes *anObject* the receiver's delegate.

```
public void setDelegate(Object anObject)
```

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

See Also
`delegate` (page 110)

setMainMenu

Makes *aMenu* the receiver's main menu.

```
public void setMainMenu(NSMenu aMenu)
```

See Also
`mainMenu` (page 113)

setServicesMenu

Makes *aMenu* the receiver's Services menu.

```
public void setServicesMenu(NSMenu aMenu)
```

See Also
servicesMenu (page 121)

setServicesProvider

Registers the object *aProvider* as the service provider.

```
public void setServicesProvider(Object aProvider)
```

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

See Also
servicesProvider (page 122)

setWindowsMenu

Makes *aMenu* the receiver's Window menu.

```
public void setWindowsMenu(NSMenu aMenu)
```

See Also
windowsMenu (page 128)

setWindowsNeedUpdate

Sets whether the receiver's windows need updating when the receiver has finished processing the current event, depending on the Boolean value passed in place of *flag*.

```
public void setWindowsNeedUpdate(boolean flag)
```

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

See Also
updateWindows (page 127)

showHelp

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

```
public void showHelp(Object sender)
```

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

See Also
`activateContextHelpMode` (page 106)

stop

Stops the main event loop.

```
public void stop(Object sender)
```

Discussion
This method will break the flow of control out of the `run` (page 119) method, thereby returning to the `main()` function. A subsequent `run` message will restart the loop.

If this method is invoked during a modal event loop, it will break that loop but not the main event loop.

See Also
`runModalForWindow` (page 119)
`runModalSession` (page 120)
`terminate` (page 125)

stopModal

Stops a modal event loop.

```
public void stopModal()
```

Discussion
This method should always be paired with a previous invocation of `runModalForWindow` (page 119) or `beginModalSessionForWindow` (page 108). When `runModalForWindow` (page 119) is stopped with this method, it returns `RunStoppedResponse`. This method stops the loop only if it's executed by code responding to an event. If you need to stop a `runModalForWindow` (page 119) loop outside of one of its event callbacks (for example, a method repeatedly invoked by an NSTimer object or a method running in a different thread), use the `abortModal` (page 106) method.

See Also
`runModalSession` (page 120)
`stopModalWithCode` (page 124)

stopModalWithCode

Like `stopModal` (page 124), except the argument *returnCode* allows you to specify the value `runModalForWindow` (page 119) will return.

```
public void stopModalWithCode(int returnCode)
```

See Also

`abortModal` (page 106)

targetForAction

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

```
public Object targetForAction(NSSelector aSelector)
```

Discussion

If *aSelector* is `NULL`, `null` is returned.

See Also

`sendActionToTargetFromSender` (page 121)

`tryToPerform` (page 126)

`targetForActionToFrom` (page 125)

targetForActionToFrom

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

```
public Object targetForActionToFrom(NSSelector anAction, Object aTarget, Object
    sender)
```

Discussion

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

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

See Also

`sendActionToTargetFromSender` (page 121)

`tryToPerform` (page 126)

`targetForAction` (page 125)

terminate

Terminates the receiver.

```
public void terminate(Object sender)
```

Discussion

This method is typically invoked when the user chooses Quit or Exit from the application's menu. Each use of `terminate` invokes `applicationShouldTerminate` (page 136) to notify the delegate that the application will terminate. If `applicationShouldTerminate` (page 136) returns `false`, control is returned to the main event loop, and the application isn't terminated. Otherwise, the document controller of the application (if one exists) is asked to check its documents and, if there are unsaved changes, ask the user to save those changes. Then, this method posts an `ApplicationWillTerminateNotification` (page 141) to the default notification center. Don't put final cleanup code in your application's `main()` function—it will never be executed. If cleanup is necessary, have the delegate respond to `applicationWillTerminate` (page 139) and perform cleanup in that method.

See Also

`run` (page 119)

`stop` (page 124)

tryToPerform

Dispatches action messages.

```
public boolean tryToPerform(NSSelector aSelector, Object anObject)
```

Discussion

If *aSelector* is `NULL`, `false` is returned. The receiver tries to perform the method *aSelector* using its inherited NSResponder method `tryToPerform` (page 1199). If the receiver doesn't perform *aSelector*, the delegate is given the opportunity to perform it. If either the receiver or its delegate accepts *aSelector*, this method returns `true`. Otherwise, it returns `false`.

unhide

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

```
public void unhide(Object sender)
```

Discussion

Invokes `unhideWithoutActivation` (page 127).

See Also

`activateIgnoringOtherApps` (page 106)

`hide` (page 112)

unhideAllApplications

Unhides all applications, including the receiver.

```
public void unhideAllApplications(Object sender)
```

Discussion

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

unhideWithoutActivation

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

```
public void unhideWithoutActivation()
```

Discussion

When this method begins, it posts an `ApplicationWillUnhideNotification` (page 141) to the default notification center. If it completes successfully, it posts an `ApplicationDidUnhideNotification` (page 140).

See Also

`activateIgnoringOtherApps` (page 106)
`hide` (page 112)
`applicationDidUnhide` (page 132)
`applicationWillUnhide` (page 139)

updateWindows

Sends an `update` (page 1871) message to each onscreen NSWindow.

```
public void updateWindows()
```

Discussion

This method is invoked automatically in the main event loop after each event when running in `NSRunLoop.DefaultRunLoopMode` or `NSApplication.ModalRunLoopMode`. This method is not invoked automatically when running in `NSApplication.EventTrackingRunLoopMode`. If the NSWindow has automatic updating turned on, its `update` (page 1871) method will redraw all the NSWindow's NSViews that need redrawing. If automatic updating is turned off, the `update` (page 1871) message does nothing. (You turn automatic updating on and off by sending `setAutodisplay` (page 1855) to an NSWindow.)

When this method begins, it posts an `ApplicationWillUpdateNotification` (page 141) to the default notification center. When it successfully completes, it posts an `ApplicationDidUpdateNotification` (page 140).

See Also

`setWindowsNeedUpdate` (page 123)
`applicationDidUpdate` (page 132)
`applicationWillUpdate` (page 139)

updateWindowsItem

Updates the Window menu item for *aWindow* to reflect the edited status of *aWindow*.

```
public void updateWindowsItem(NSWindow aWindow)
```

Discussion

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

See Also

`changeWindowsItem` (page 109)

`setDocumentEdited` (page 1859) (NSWindow)

validRequestorForTypes

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

```
public Object validRequestorForTypes(String sendType, String returnType)
```

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

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

See Also
`registerServicesMenuTypes` (page 117)
`validRequestorForTypes` (page 1200) (NSResponder)

windows

Returns an NSArray of the receiver's NSWindows, including offscreen windows.

```
public NSArray windows()
```

windowsMenu

Returns the Window menu or `null` if no Window menu has been created.

```
public NSMenu windowsMenu()
```

See Also
`setWindowsMenu` (page 123)

windowWithWindowNumber

Returns the NSWindow object corresponding to *windowNum*.

```
public NSWindow windowWithWindowNumber(int windowNum)
```

Constants

These are possible return values for `runModalForWindow` (page 119) and `runModalSession` (page 120):

Constant	Description
RunStoppedResponse	Modal session was broken with stopModal (page 124).
RunAbortedResponse	Modal session was broken with abortModal (page 106).
RunContinuesResponse	Modal session is continuing (returned by runModalSession (page 120) only).

The following constants define whether an application should terminate:

Constant	Description
TerminateNow	It is OK to proceed with termination.
TerminateCancel	The application should not be terminated.
TerminateLater	It may be OK to proceed with termination later. The application must call replyToApplicationShouldTerminate (page 118) with true or false once the answer is known. This return value is for delegates that need to provide document modal alerts (sheets) in order to decide whether to quit.

The following constants indicate whether or not a copy or print operation was successful, was cancelled, or failed. These constants are used by the replyToOpenOrPrint (page 118) method:

Constant	Description
DelegateReplySuccess	Indicates the operation succeeded.
DelegateReplyCancel	Indicates the user cancelled the operation.
DelegateReplyFailure	Indicates an error occurred processing the operation.

The following NSApplicationPrintReply constants are returned by applicationPrintFiles (page 135):

Constant	Description
PrintingCancelled	Printing was cancelled. Available in Mac OS X v10.4 and later.
PrintingSuccess	Printing was successful. Available in Mac OS X v10.4 and later.
PrintingFailure	Printing failed. Available in Mac OS X v10.4 and later.
PrintingReplyLater	The result of printing cannot be returned immediately, for example, if printing will cause the presentation of a sheet. If your method returns NSPrintingReplyLater it must always invoke replyToOpenOrPrint (page 118) when the entire print operation has been completed, successfully or not. Available in Mac OS X v10.4 and later.

The following loop mode constants are defined by NSApplication:

Constant	Description
EventTrackingRunLoopMode	A run loop should be set to this mode when tracking events modally, such as a mouse-dragging loop.
ModalPanelRunLoopMode	A run loop should be set to this mode when waiting for input from a modal panel, such as SavePanel or OpenPanel.

The following constants specify the level of severity of a user attention request and are used by cancelUserAttentionRequest (page 109) and requestUserAttention (page 118):

Constant	Description
UserAttentionRequest-Critical	The user attention request is a critical request. The dock icon will bounce until either the application becomes active or the request is canceled.
UserAttentionRequest-Informational	The user attention request is an informational request. The dock icon will bounce for one second (usually a single bounce). The request, though, remains active until either the application becomes active or the request is canceled.

The following constant can be used to determine if you are using a version of the Application Kit framework newer than the version delivered in Mac OS X v10.0:

Constant	Description
AppKitVersionNumber10_0	The Application Kit framework included in Mac OS X v10.0
AppKitVersionNumber10_1	The Application Kit framework included in Mac OS X v10.1
AppKitVersionNumber10_2	The Application Kit framework included in Mac OS X v10.2
AppKitVersionNumber10_2_3	The Application Kit framework included in Mac OS X v10.2.3

Delegate Methods

applicationDidBecomeActive

```
public abstract void applicationDidBecomeActive(NSNotification aNotification)
```

Discussion

Sent by the default notification center immediately after the application becomes active. *aNotification* is always an ApplicationDidBecomeActiveNotification (page 140). You can retrieve the NSApplication object by sending object to *aNotification*.

See Also

applicationDidFinishLaunching (page 131) applicationDidResignActive (page 131)

applicationWillBecomeActive (page 137)

applicationDidChangeScreenParameters

```
public abstract void applicationDidChangeScreenParameters(NSNotification
    aNotification)
```

Discussion

Sent by the default notification center when the configuration of the displays attached to the computer is changed (either programmatically or when the user changes settings in the Displays control panel). *aNotification* is always an ApplicationDidChangeScreenParametersNotification (page 140). You can retrieve the NSApplication object by sending object to *aNotification*.

applicationDidFinishLaunching

```
public abstract void applicationDidFinishLaunching(NSNotification aNotification)
```

Discussion

Sent by the default notification center after the application has been launched and initialized but before it has received its first event. *aNotification* is always an ApplicationDidFinishLaunchingNotification (page 140). You can retrieve the NSApplication object in question by sending object to *aNotification*. The delegate can implement this method to perform further initialization. If the user started up the application by double-clicking a file, the delegate receives the applicationOpenFile (page 133) message before receiving applicationDidFinishLaunching. (applicationWillFinishLaunching (page 137) is sent before applicationOpenFile.)

See Also

finishLaunching (page 112)

applicationDidBecomeActive (page 130)

applicationDidHide

```
public abstract void applicationDidHide(NSNotification aNotification)
```

Discussion

Sent by the default notification center immediately after the application is hidden. *aNotification* is always an ApplicationDidHideNotification (page 140). You can retrieve the NSApplication object in question by sending object to *aNotification*.

See Also

applicationWillHide (page 138)

applicationDidUnhide (page 132)

hide (page 112)

applicationDidResignActive

```
public abstract void applicationDidResignActive(NSNotification aNotification)
```

Discussion

Sent by the default notification center immediately after the application is deactivated. *aNotification* is always an `ApplicationDidResignActiveNotification` (page 140). You can retrieve the NSApplication object in question by sending `object` to *aNotification*.

See Also

`applicationDidBecomeActive` (page 130)

`applicationWillResignActive` (page 138)

applicationDidUnhide

```
public abstract void applicationDidUnhide(NSNotification aNotification)
```

Discussion

Sent by the default notification center immediately after the application is made visible. *aNotification* is always an `ApplicationDidUnhideNotification` (page 140). You can retrieve the NSApplication object in question by sending `object` to *aNotification*.

See Also

`applicationDidHide` (page 131)

`applicationWillUnhide` (page 139)

`unhide` (page 126)

applicationDidUpdate

```
public abstract void applicationDidUpdate(NSNotification aNotification)
```

Discussion

Sent by the default notification center immediately after the NSApplication object updates its NSWindows. *aNotification* is always an `ApplicationDidUpdateNotification` (page 140). You can retrieve the NSApplication object in question by sending `object` to *aNotification*.

See Also

`applicationWillUpdate` (page 139)

`updateWindows` (page 127)

applicationDockMenu

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

```
public abstract NSMenu applicationDockMenu(NSApplication sender)
```

Discussion

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

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

The target and action for each menu item are passed to the dock. On selection of the menu item the dock messages your application, which should invoke `(NSApplication.sharedApplication().sendActionToTargetFromSender (selector, target, null)`.

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

applicationOpenFile

```
public abstract boolean applicationOpenFile(NSApplication theApplication, String
    filename)
```

Discussion

Sent directly by *theApplication* to the delegate. The method should open the file *filename*, returning `true` if the file is successfully opened, and `false` otherwise. If the user started up the application by double-clicking a file, the delegate receives the `applicationOpenFile` message before receiving `applicationDidFinishLaunching` (page 131). (`applicationWillFinishLaunching` (page 137) is sent before `applicationOpenFile`.)

See Also

`applicationOpenFileWithoutUI` (page 133)

`applicationOpenTempFile` (page 134)

`applicationOpenUntitledFile` (page 134)

applicationOpenFiles

Identical to `applicationOpenFile` (page 133) except that the receiver opens multiple files corresponding to the file names in the *filenames* array.

```
public abstract void applicationOpenFiles(NSApplication sender, NSArray filenames)
```

Discussion

Delegates should invoke the `replyToOpenOrPrint` (page 118) method upon success or failure, or when the user cancels the operation.

Availability

Available in Mac OS X v10.3 and later.

applicationOpenFileWithoutUI

```
public abstract boolean applicationOpenFileWithoutUI(Object sender, String filename)
```

Discussion

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

See Also

`applicationOpenFile` (page 133)
`applicationOpenTempFile` (page 134)
`applicationOpenUntitledFile` (page 134)
`applicationPrintFile` (page 134)

applicationOpenTempFile

```
public abstract boolean applicationOpenTempFile(NSApplication theApplication, String
      filename)
```

Discussion

Sent directly by *theApplication* to the delegate. The method should attempt to open the file *filename*, returning `true` if the file is successfully opened, and `false` otherwise.

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

See Also

`applicationOpenFile` (page 133)
`applicationOpenFileWithoutUI` (page 133)
`applicationOpenUntitledFile` (page 134)

applicationOpenUntitledFile

```
public abstract boolean applicationOpenUntitledFile(NSApplication theApplication)
```

Discussion

Sent directly by *theApplication* to the delegate to request that a new, untitled file be opened. Returns `true` if the file was successfully opened, `false` otherwise.

See Also

`applicationOpenFile` (page 133)
`applicationOpenFileWithoutUI` (page 133)
`applicationOpenTempFile` (page 134)

applicationPrintFile

```
public abstract boolean applicationPrintFile(NSApplication theApplication, String
      filename)
```

Discussion

Sent when the user starts up the application on the command line with the `-NSPrint` option. Sent directly by *`theApplication`* to the delegate.

The method should attempt to print the file *`filename`*, returning `true` if the file was successfully printed, and `false` otherwise. The application terminates (using the `terminate` (page 125) method) after this method returns.

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

See Also

`applicationOpenFileWithoutUI` (page 133)

applicationPrintFiles

Identical to `applicationPrintFile` (page 134) except that the receiver prints multiple files corresponding to the file names in the *`filenames`* array.

```
public abstract void applicationPrintFiles(NSApplication sender, NSArray filenames)
```

Discussion

Delegates should invoke the `replyToOpenOrPrint` (page 118) method upon success or failure, or when the user cancels the operation.

This method has been deprecated. Use `applicationPrintFiles` (page 135) instead.

Availability

Deprecated in Mac OS X v10.4.

applicationPrintFiles

Prints a group of files.

```
public abstract int applicationPrintFiles(NSApplication application, NSArray
    fileNames, NSDictionary printSettings, boolean showPrintPanels)
```

Discussion

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

The value returned should be one of the `NSApplicationPrintReply` values defined in "Constants" (page 128).

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

This delegate method replaces `applicationPrintFiles` (page 135), which is now deprecated. If your application delegate only implements the deprecated method, it is still invoked, and NSApplication uses private functionality to arrange for the print settings to take effect.

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

applicationShouldHandleReopen

Sent by *theApplication* to the delegate prior to default behavior to reopen (`rapp`) AppleEvents.

```
public abstract boolean applicationShouldHandleReopen(NSApplication theApplication,
     boolean flag)
```

Discussion
These events are sent whenever the Finder reactivates an already running application because someone double-clicked it again or used the dock to activate it. By default the Application Kit will handle this event by checking whether there are any visible NSWindows (not NSPanels), and, if there are none, it goes through the standard untitled document creation (the same as it does if *theApplication* is launched without any document to open). For most document-based applications, an untitled document will be created. The application delegate will also get a chance to respond to the normal untitled document delegations.If you implement this method in your application delegate, it will be called before any of the default behavior happens. If you return `true`, then NSApplication will go on to do its normal thing. If you return `false`, then NSApplication will do nothing. So, you can either implement this method, do nothing, and return `false` if you do not want anything to happen at all (not recommended), or you can implement this method, handle the event yourself in some custom way, and return `false`.

flag indicates whether NSApplication has found any visible NSWindows in your application. *flag* can be used as an indication of whether NSApplication would do anything if you return `true`.

Note that what happens to minimized windows is not determined yet, but the intent is that *flag* being `false` indicates whether the Application Kit will create a new window to satisfy the reopen event.

applicationShouldOpenUntitledFile

```
public abstract boolean applicationShouldOpenUntitledFile(NSApplication sender)
```

Discussion
Invoked immediately before opening an untitled file. Return `false` to prevent the application from opening an untitled file; return `true` otherwise. Note that `applicationOpenUntitledFile` (page 134) is invoked if this method returns `true`.

applicationShouldTerminate

```
public abstract boolean applicationShouldTerminate(NSApplication sender)
```

Discussion
Invoked from within the `terminate` (page 125) method immediately before the application terminates. *sender* is the NSApplication to be terminated. If this method returns `false`, the application is not terminated, and control returns to the main event loop. Return `true` to allow the application to terminate.

See Also

`terminate` (page 125)

`applicationShouldTerminateAfterLastWindowClosed` (page 137)

`applicationWillTerminate` (page 139)

applicationShouldTerminateAfterLastWindowClosed

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

```
public abstract boolean applicationShouldTerminateAfterLastWindowClosed(NSApplication
     theApplication)
```

Discussion

This method is sent when the last window is closed regardless of whether there are still open panels (a panel in this case is defined as being an instance of NSPanel or one of its subclasses).

If this method returns `false`, the application is not terminated, and control returns to the main event loop. Return `true` to allow the application to terminate. Note that `applicationShouldTerminate` (page 136) is invoked if this method returns `true`.

See Also

`terminate` (page 125)

applicationWillBecomeActive

```
public abstract void applicationWillBecomeActive(NSNotification aNotification)
```

Discussion

Sent by the default notification center immediately before the application becomes active. *aNotification* is always an `ApplicationWillBecomeActiveNotification` (page 140). You can retrieve the NSApplication object in question by sending `object` to *aNotification*.

See Also

`applicationDidBecomeActive` (page 130)

`applicationWillFinishLaunching` (page 137)

`applicationWillResignActive` (page 138)

applicationWillFinishLaunching

```
public abstract void applicationWillFinishLaunching(NSNotification aNotification)
```

Discussion

Sent by the default notification center immediately before the NSApplication object is initialized. *aNotification* is always an `ApplicationWillFinishLaunchingNotification` (page 141). You can retrieve the NSApplication object in question by sending `object` to *aNotification*.

See Also

`applicationDidFinishLaunching` (page 131)

`applicationWillBecomeActive` (page 137)

`finishLaunching` (page 112)

applicationWillHide

```
public abstract void applicationWillHide(NSNotification aNotification)
```

Discussion

Sent by the default notification center immediately after the application is hidden. *aNotification* is always an ApplicationWillHideNotification (page 141). You can retrieve the NSApplication object in question by sending object to *aNotification*.

See Also

applicationDidHide (page 131)

hide (page 112)

applicationWillPresentError

Sent to the delegate before *application* presents an error message, based on the information in *error*, to the user.

```
public abstract NSError applicationWillPresentError(NSApplication application,
    NSError error)
```

Discussion

The delegate can return a new NSError object or the same one passed in *error*.

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

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

Availability

Available in Mac OS X v10.4 and later.

applicationWillResignActive

```
public abstract void applicationWillResignActive(NSNotification aNotification)
```

Discussion

Sent by the default notification center immediately after the application is deactivated. *aNotification* is always an ApplicationWillResignActiveNotification (page 141). You can retrieve the NSApplication object in question by sending object to *aNotification*.

See Also

applicationWillBecomeActive (page 137)

applicationDidResignActive (page 131)

applicationWillTerminate

```
public abstract void applicationWillTerminate(NSNotification aNotification)
```

Discussion

Sent by the default notification center immediately before the application terminates. *aNotification* is always an `ApplicationWillTerminateNotification` (page 141). You can retrieve the NSApplication object in question by sending `object` to *aNotification*. Put any necessary cleanup code in this method.

See Also

`applicationShouldTerminate` (page 136)

`terminate` (page 125)

applicationWillUnhide

```
public abstract void applicationWillUnhide(NSNotification aNotification)
```

Discussion

Sent by the default notification center immediately after the application is unhidden. *aNotification* is always an `ApplicationWillUnhideNotification` (page 141). You can retrieve the NSApplication object in question by sending `object` to *aNotification*.

See Also

`unhide` (page 126)

`applicationDidUnhide` (page 132)

`applicationWillHide` (page 138)

applicationWillUpdate

```
public abstract void applicationWillUpdate(NSNotification aNotification)
```

Discussion

Sent by the default notification center immediately before the NSApplication object updates its NSWindows. *aNotification* is always an `ApplicationWillUpdateNotification` (page 141). You can retrieve the NSApplication object in question by sending `object` to *aNotification*.

See Also

`applicationDidUpdate` (page 132)

`updateWindows` (page 127)

Notifications

These notifications apply to NSApplication. See "Notifications" (page 1913) in NSWorkspace for additional, similar notifications.

ApplicationDidBecomeActiveNotification

Posted immediately after the application becomes active. The notification object is `NSApplication.sharedApplication()`. This notification does not contain a *`userInfo`* dictionary.

ApplicationDidChangeScreenParametersNotification

Posted when the configuration of the displays attached to the computer is changed (either programmatically or when the user changes settings in the Displays control panel). The notification object is `NSApplication.sharedApplication()`. This notification does not contain a *`userInfo`* dictionary.

ApplicationDidFinishLaunchingNotification

Posted at the end of the `finishLaunching` (page 112) method to indicate that the application has completed launching and is ready to run. The notification object is `NSApplication.sharedApplication()`. This notification does not contain a *`userInfo`* dictionary.

ApplicationDidHideNotification

Posted at the end of the `hide` (page 112) method to indicate that the application is now hidden. The notification object is `NSApplication.sharedApplication()`. This notification does not contain a *`userInfo`* dictionary.

ApplicationDidResignActiveNotification

Posted immediately after the application gives up its active status to another application. The notification object is `NSApplication.sharedApplication()`. This notification does not contain a *`userInfo`* dictionary.

ApplicationDidUnhideNotification

Posted at the end of the `unhideWithoutActivation` (page 127) method to indicate that the application is now visible. The notification object is `NSApplication.sharedApplication()`. This notification does not contain a *`userInfo`* dictionary.

ApplicationDidUpdateNotification

Posted at the end of the `updateWindows` (page 127) method to indicate that the application has finished updating its windows. The notification object is `NSApplication.sharedApplication()`. This notification does not contain a *`userInfo`* dictionary.

ApplicationWillBecomeActiveNotification

Posted immediately after the application becomes active. The notification object is `NSApplication.sharedApplication()`. This notification does not contain a *`userInfo`* dictionary.

ApplicationWillFinishLaunchingNotification

Posted at the start of the `finishLaunching` (page 112) method to indicate that the application has completed its initialization process and is about to finish launching. The notification object is `NSApplication.sharedApplication()`. This notification does not contain a *`userInfo`* dictionary.

ApplicationWillHideNotification

Posted at the start of the `hide` (page 112) method to indicate that the application is about to be hidden. The notification object is `NSApplication.sharedApplication()`. This notification does not contain a *`userInfo`* dictionary.

ApplicationWillResignActiveNotification

Posted immediately before the application gives up its active status to another application. The notification object is `NSApplication.sharedApplication()`. This notification does not contain a *`userInfo`* dictionary.

ApplicationWillTerminateNotification

Posted by the `terminate` (page 125) method to indicate that the application will terminate. Posted only if the delegate method `applicationShouldTerminate` (page 136) returns `true`. The notification object is `NSApplication.sharedApplication()`. This notification does not contain a *`userInfo`* dictionary.

ApplicationWillUnhideNotification

Posted at the start of the `unhideWithoutActivation` (page 127) method to indicate that the application is about to be visible. The notification object is `NSApplication.sharedApplication()`. This notification does not contain a *`userInfo`* dictionary.

ApplicationWillUpdateNotification

Posted at the start of the `updateWindows` (page 127) method to indicate that the application is about to update its windows. The notification object is `NSApplication.sharedApplication()`. This notification does not contain a *`userInfo`* dictionary.

NSArrayController

Inherits from	NSObjectController : NSController : NSObject
Implements	NSCoding (NSController)
Package:	com.apple.cocoa.application
Availability	Available in Mac OS X v10.3 and later.
Companion guides	Cocoa Bindings Programming Topics Predicate Programming Guide Core Data Programming Guide

Overview

The NSArrayController is a bindings compatible class that manages an array of objects. It provides selection management and sorting capabilities.

Tasks

Constructors

`NSArrayController` (page 146)
Creates and returns an empty NSArrayController.

Managing Sort Descriptors

`setSortDescriptors` (page 159)
Sets the sort descriptors used by the receiver to arrange objects to *`sortDescriptors`*.

`sortDescriptors` (page 159)
Returns an array of sort descriptors used by the receiver to arrange objects.

Arranging Objects

`arrangeObjects` (page 149)
Returns an array containing the content of the *`objects`* array arranged according to the receiver's `sortDescriptors` (page 159).

arrangedObjects (page 148)
: Returns an array containing the receiver's content objects arranged using arrangeObjects (page 149)

Setting Selection Attributes

setAvoidsEmptySelection (page 156)
: Sets whether the receiver will attempt to avoid an empty selection.

avoidsEmptySelection (page 149)
: Returns true if the receiver requires that the content array attempt to maintain a selection at all times.

setClearsFilterPredicateOnInsertion (page 157)
: Sets whether the receiver automatically clears an existing filter predicate when a new object is inserted or added to the content array.

preservesSelection (page 152)
: Returns whether the receiver will attempt to preserve the current selection then when the content changes.

setPreservesSelection (page 157)
: Sets whether the receiver will attempt to preserve selection when the content changes.

alwaysUsesMultipleValuesMarker (page 148)
: Returns whether the receiver always returns the multiple values marker when multiple objects are selected, even if the selected items have the same value.

setAlwaysUsesMultipleValuesMarker (page 156)
: Sets whether the receiver always returns the multiple values marker when multiple objects are selected, even if they have the same value.

Getting the Current Selection

setSelectionIndex (page 158)
: Sets the receiver's current selection to *index*, returning true if the selection was changed.

selectionIndex (page 154)
: Returns the index of the first object in the receiver's selection, NSArray.NotFound if there is no selection.

Managing Selections

setSelectsInsertedObjects (page 159)
: Sets whether the receiver will automatically select objects as they are inserted.

selectsInsertedObjects (page 156)
: Returns whether the receiver selects inserted objects automatically.

setSelectionIndexes (page 158)
: Sets the receiver's current selection to the objects at *indexe*s, returning true if the selection was changed.

`selectionIndexes` (page 155)
: Returns an index set containing the indexes of the receiver's currently selected objects in the content array.

`addSelectionIndexes` (page 148)
: Adds the objects at the specified indexes in the receiver's content array to the current selection, returning `true` if the selection was changed.

`removeSelectionIndexes` (page 154)
: Removes the object as the specified *`indexes`* from the receiver's current selection, returning `true` if the selection was changed.

`setSelectedObjects` (page 158)
: Sets *`objects`* as the receiver's current selection, returning `true` if the selection was changed.

`selectedObjects` (page 154)
: Returns an array containing the receiver's selected objects.

`addSelectedObjects` (page 147)
: Adds *`objects`* from the receiver's content array to the current selection, returning `true` if the selection was changed.

`removeSelectedObjects` (page 154)
: Removes *`objects`* from the receiver's current selection, returning `true` if the selection was changed.

`selectNext` (page 155)
: Selects the next object, relative to the current selection, in the receiver's arranged content.

`canSelectNext` (page 150)
: Returns `true` if the next object, relative to the current selection, in the receiver's content array can be selected.

`selectPrevious` (page 155)
: Selects the previous object, relative to the current selection, in the receiver's arranged content.

`canSelectPrevious` (page 150)
: Returns `true` if the previous object, relative to the current selection, in the receiver's content array can be selected.

Inserting

`canInsert` (page 149)
: Returns `true` if an object can be inserted into the receiver's content array.

`insert` (page 151)
: Creates a new object of the class specified by `objectClass` (page 1013) and inserts it into the receiver's content array.

Adding and Removing Objects

`addObject` (page 147)
: Adds *`object`* to the receiver's content array and the arranged objects array.

`addObjects` (page 147)
: Adds *`objects`* to the receiver's content array.

`insertObject` (page 151)
: Inserts *object* into the receiver's arranged objects array at the location specified by *index*, and adds it to the receiver's content array.

`insertObjects` (page 152)
: Inserts *objects* into the receiver's arranged objects array at the locations specified in *indexes*, and adds it to the receiver's content array.

`remove` (page 152)
: Removes the receiver's selected objects from the content array.

`removeObject` (page 153)
: Removes *object* from the receiver's content collection.

`removeObjects` (page 153)
: Removes *objects* from the receiver's content array.

Filtering Objects

`clearsFilterPredicateOnInsertion` (page 150)
: Returns whether the receiver automatically clears an existing filter predicate when new items are inserted or added to the content.

`filterPredicate` (page 151)
: Returns the predicate used by the receiver to filter the array controller contents.

`setFilterPredicate` (page 157)
: Sets the predicate used to filter the contents of the receiver to *filterPredicate*, replacing any existing filter predicate.

Constructors

NSArrayController

Creates and returns an empty NSArrayController.

```
public NSArrayController()
```

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

Creates and returns an NSArrayController with the specified *content*.

```
public NSArrayController(Object content)
```

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

Instance Methods

addObject

Adds *object* to the receiver's content array and the arranged objects array.

```
public void addObject(Object object)
```

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

See Also
addObjects (page 147)
insertObject (page 151)
removeObject (page 153)

addObjects

Adds *objects* to the receiver's content array.

```
public void addObjects(NSArray objects)
```

Discussion
If selectsInsertedObjects (page 156) returns true, which is the default, the added objects are selected in the array controller.

It is important to note that inserting many objects with selectsInsertedObjects on can cause a significant performance penalty. In this case it is more efficient to use the setContent (page 1016) method to set the array, or to set selectsInsertedObjects to false before adding the objects with addObjects.

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

See Also
addObject (page 147)
insertObjects (page 152)
removeObjects (page 153)

addSelectedObjects

Adds *objects* from the receiver's content array to the current selection, returning true if the selection was changed.

```
public boolean addSelectedObjects(NSArray objects)
```

Discussion
Attempting to change the selection may cause a commitEditing (page 470) message which fails, thus denying the selection change.

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

See Also
`removeSelectedObjects` (page 154)
`setSelectedObjects` (page 158)

addSelectionIndexes

Adds the objects at the specified indexes in the receiver's content array to the current selection, returning `true` if the selection was changed.

```
public boolean addSelectionIndexes(NSIndexSet indexes)
```

Discussion
Attempting to change the selection may cause a `commitEditing` (page 470) message which fails, thus denying the selection change.

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

See Also
`removeSelectionIndexes` (page 154)

alwaysUsesMultipleValuesMarker

Returns whether the receiver always returns the multiple values marker when multiple objects are selected, even if the selected items have the same value.

```
public boolean alwaysUsesMultipleValuesMarker()
```

Discussion
The default is `false`.

This property is observable using key-value observing.

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

See Also
`setAlwaysUsesMultipleValuesMarker` (page 156)

arrangedObjects

Returns an array containing the receiver's content objects arranged using `arrangeObjects` (page 149)

```
public Object arrangedObjects()
```

Discussion
.

This property is observable using key-value observing.

Availability

Available in Mac OS X v10.3 and later.

See Also

`arrangeObjects` (page 149)

arrangeObjects

Returns an array containing the content of the *`objects`* array arranged according to the receiver's `sortDescriptors` (page 159).

```
public NSArray arrangeObjects(NSArray objects)
```

Discussion

Subclasses should override this method to use a different sort mechanism, provide custom object arrangement, or filter the objects.

Availability

Available in Mac OS X v10.3 and later.

See Also

`arrangedObjects` (page 148)

avoidsEmptySelection

Returns `true` if the receiver requires that the content array attempt to maintain a selection at all times.

```
public boolean avoidsEmptySelection()
```

Discussion

The default is `true`.

This property is observable using key-value observing.

Availability

Available in Mac OS X v10.3 and later.

See Also

`setAvoidsEmptySelection` (page 156)

canInsert

Returns `true` if an object can be inserted into the receiver's content array.

```
public boolean canInsert()
```

Discussion

The result of this method can be used by a binding to enable user interface items.

This property is observable using key-value observing.

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

See Also
`insert` (page 151)

canSelectNext

Returns `true` if the next object, relative to the current selection, in the receiver's content array can be selected.

```
public boolean canSelectNext()
```

Discussion
The result of this method can be used by a binding to enable user interface items.

This property is observable using key-value observing.

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

See Also
`selectNext` (page 155)
`canSelectPrevious` (page 150)

canSelectPrevious

Returns `true` if the previous object, relative to the current selection, in the receiver's content array can be selected.

```
public boolean canSelectPrevious()
```

Discussion
The result of this method can be used by a binding to enable user interface items.

This property is observable using key-value observing.

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

See Also
`canSelectNext` (page 150)
`selectPrevious` (page 155)

clearsFilterPredicateOnInsertion

Returns whether the receiver automatically clears an existing filter predicate when new items are inserted or added to the content.

```
public boolean clearsFilterPredicateOnInsertion()
```

Discussion
The default is `true`.

This property is observable using key-value observing.

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

See Also
`setClearsFilterPredicateOnInsertion` (page 157)

filterPredicate

Returns the predicate used by the receiver to filter the array controller contents.

```
public NSPredicate filterPredicate()
```

Discussion
Returns `null` if no filter predicate is set.

This property is observable using key-value observing.

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

See Also
`setClearsFilterPredicateOnInsertion` (page 157)

insert

Creates a new object of the class specified by `objectClass` (page 1013) and inserts it into the receiver's content array.

```
public void insert(Object sender)
```

Discussion
The *`sender`* is typically the object that invoked this method.

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

See Also
`canInsert` (page 149)

insertObject

Inserts *`object`* into the receiver's arranged objects array at the location specified by *`index`*, and adds it to the receiver's content array.

```
public void insertObject(Object object, int index)
```

Discussion
Subclasses can override this method to provide customized arranged objects support.

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

See Also
`insertObjects` (page 152)
`addObject` (page 147)
`removeObject` (page 153)

insertObjects

Inserts *objects*s into the receiver's arranged objects array at the locations specified in *indexes*, and adds it to the receiver's content array.

```
public void insertObjects(NSArray objects, NSIndexSet indexes)
```

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

See Also
`insertObject` (page 151)
`addObjects` (page 147)
`removeObjects` (page 153)

preservesSelection

Returns whether the receiver will attempt to preserve the current selection then when the content changes.

```
public boolean preservesSelection()
```

Discussion
The default is `true`.

This property is observable using key-value observing.

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

See Also
`setClearsFilterPredicateOnInsertion` (page 157)

remove

Removes the receiver's selected objects from the content array.

```
public void remove(Object object)
```

Discussion
The *sender* is typically the object that invoked this method.

Availability

Available in Mac OS X v10.3 and later.

See Also

`removeObjects` (page 153)

`addObject` (page 147)

removeObject

Removes *object* from the receiver's content collection.

```
public void removeObject(Object object)
```

Discussion

If you are using Core Data, the exact semantics of this method differ depending on the settings for the array controller. If the receiver's content is fetched automatically, removed objects are marked for deletion by the managed object context (and hence removal from the object graph). If, however, the receiver's contentSet is bound to a relationship, removeObject: by default only removes the object from the relationship, not from the object graph. You can, though, set the "Deletes Object on Remove" option for the contentSet binding, in which case objects are marked for deletion as well as being removed from the relationship.

Removes the object at the specified *index* in the receiver's arranged objects from the receiver's content array.

```
public void removeObject(int index)
```

Availability

Available in Mac OS X v10.3 and later.

See Also

`removeObjects` (page 153)

`addObject` (page 147)

removeObjects

Removes *objects* from the receiver's content array.

```
public void removeObjects(NSArray objects)
```

Removes the objects at the specified *indexes* in the receiver's arranged objects from the content array.

```
public void removeObjects(NSIndexSet indexes)
```

Availability

Available in Mac OS X v10.3 and later.

See Also

`removeObject` (page 153)

`addObjects` (page 147)

removeSelectedObjects

Removes *`objects`* from the receiver's current selection, returning `true` if the selection was changed.

```
public boolean removeSelectedObjects(NSArray objects)
```

Discussion
Attempting to change the selection may cause a `commitEditing` (page 470) message which fails, thus denying the selection change.

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

See Also
`addSelectedObjects` (page 147)

removeSelectionIndexes

Removes the object as the specified *`indexes`* from the receiver's current selection, returning `true` if the selection was changed.

```
public boolean removeSelectionIndexes(NSIndexSet indexes)
```

Discussion
Attempting to change the selection may cause a `commitEditing` (page 470) message which fails, thus denying the selection change.

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

See Also
`addSelectionIndexes` (page 148)

selectedObjects

Returns an array containing the receiver's selected objects.

```
public NSArray selectedObjects()
```

Discussion
This property is observable using key-value observing.

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

See Also
`setSelectedObjects` (page 158)

selectionIndex

Returns the index of the first object in the receiver's selection, `NSArray.NotFound` if there is no selection.

```
public int selectionIndex()
```

Discussion
This property is observable using key-value observing.

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

See Also
`setSelectionIndex` (page 158)
`selectionIndexes` (page 155)

selectionIndexes

Returns an index set containing the indexes of the receiver's currently selected objects in the content array.

```
public NSIndexSet selectionIndexes()
```

Discussion
This property is observable using key-value observing.

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

See Also
`setSelectionIndexes` (page 158)
`selectionIndex` (page 154)

selectNext

Selects the next object, relative to the current selection, in the receiver's arranged content.

```
public void selectNext(Object sender)
```

Discussion
The *sender* is typically the object that invoked this method.

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

See Also
`selectPrevious` (page 155)
`canSelectNext` (page 150)

selectPrevious

Selects the previous object, relative to the current selection, in the receiver's arranged content.

```
public void selectPrevious(Object sender)
```

Discussion

The *sender* is typically the object that invoked this method.

Availability

Available in Mac OS X v10.3 and later.

See Also

`selectNext` (page 155)

`canSelectPrevious` (page 150)

selectsInsertedObjects

Returns whether the receiver selects inserted objects automatically.

```
public boolean selectsInsertedObjects()
```

Discussion

The default is `true`.

This property is observable using key-value observing.

Availability

Available in Mac OS X v10.3 and later.

See Also

`setSelectsInsertedObjects` (page 159)

setAlwaysUsesMultipleValuesMarker

Sets whether the receiver always returns the multiple values marker when multiple objects are selected, even if they have the same value.

```
public void setAlwaysUsesMultipleValuesMarker(boolean flag)
```

Discussion

Setting *flag* to `true` can increase performance if your application doesn't allow editing multiple values. The default is `false`.

Availability

Available in Mac OS X v10.4 and later.

See Also

`alwaysUsesMultipleValuesMarker` (page 148)

setAvoidsEmptySelection

Sets whether the receiver will attempt to avoid an empty selection.

```
public void setAvoidsEmptySelection(boolean)
```

Discussion
If *flag* is `true` then the receiver will maintain a selection unless there are no objects in the content array. The default is `true`.

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

See Also
`avoidsEmptySelection` (page 149)

setClearsFilterPredicateOnInsertion

Sets whether the receiver automatically clears an existing filter predicate when a new object is inserted or added to the content array.

```
public void setClearsFilterPredicateOnInsertion(boolean flag)
```

Discussion
The default is `true`.

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

See Also
`clearsFilterPredicateOnInsertion` (page 150)

setFilterPredicate

Sets the predicate used to filter the contents of the receiver to *filterPredicate*, replacing any existing filter predicate.

```
public void setFilterPredicate(NSPredicate filterPredicate)
```

Discussion
If *filterPredicate* is null any existing filter predicate is removed.

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

See Also
`filterPredicate` (page 151)

setPreservesSelection

Sets whether the receiver will attempt to preserve selection when the content changes.

```
public void setPreservesSelection(boolean flag)
```

Discussion
If *flag* is `true` then the selection will be preserved, if possible. The default is `true`.

Availability

Available in Mac OS X v10.3 and later.

See Also

`preservesSelection` (page 152)

setSelectedObjects

Sets *`objects`* as the receiver's current selection, returning `true` if the selection was changed.

```
public boolean setSelectedObjects(NSArray objects)
```

Discussion

Attempting to change the selection may cause a `commitEditing` (page 470) message which fails, thus denying the selection change.

Availability

Available in Mac OS X v10.3 and later.

See Also

`selectedObjects` (page 154)

`addSelectedObjects` (page 147)

setSelectionIndex

Sets the receiver's current selection to *`index`*, returning `true` if the selection was changed.

```
public boolean setSelectionIndex(int index)
```

Discussion

Attempting to change the selection may cause a `commitEditing` (page 470) message which fails, thus denying the selection change.

Availability

Available in Mac OS X v10.3 and later.

See Also

`selectionIndex` (page 154)

`setSelectionIndexes` (page 158)

setSelectionIndexes

Sets the receiver's current selection to the objects at *`index`*es, returning `true` if the selection was changed.

```
public boolean setSelectionIndexes(NSIndexSet indexes)
```

Discussion

Attempting to change the selection may cause a `commitEditing` (page 470) message which fails, thus denying the selection change.

To select all the receiver's objects, indexes should be an NSIndexSet with indexes [0..count -1]. To deselect all indexes, pass an empty NSIndexSet.

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

See Also
`selectionIndexes` (page 155)
`setSelectionIndex` (page 158)

setSelectsInsertedObjects

Sets whether the receiver will automatically select objects as they are inserted.

```
public void setSelectsInsertedObjects(boolean flag)
```

Discussion
If *`flag`* is `true` then items will be selected upon insertion. The default is `true`.

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

See Also
`selectsInsertedObjects` (page 156)

setSortDescriptors

Sets the sort descriptors used by the receiver to arrange objects to *`sortDescriptors`*.

```
public void setSortDescriptors(NSArray sortDescriptors)
```

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

See Also
`sortDescriptors` (page 159)

sortDescriptors

Returns an array of sort descriptors used by the receiver to arrange objects.

```
public NSArray sortDescriptors()
```

Discussion
This property is observable using key-value observing.

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

See Also
`setSortDescriptors` (page 159)

NSBezierPath

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.application
Companion guide	Basic Drawing

Overview

An NSBezierPath object allows you to create paths using PostScript-style commands. Paths consist of straight and curved line segments joined together. Paths can form recognizable shapes such as rectangles, ovals, arcs, and glyphs; they can also form complex polygons using either straight or curved line segments. A single path can be closed by connecting its two endpoints, or it can be left open.

An NSBezierPath object can contain multiple disconnected paths, whether they are closed or open. Each of these paths is referred to as a subpath of the NSBezierPath object. The subpaths of an NSBezierPath object must be manipulated as a group. The only way to manipulate subpaths individually is to create separate NSBezierPath objects for each.

For a given NSBezierPath object, you can stroke the path's outline or fill the region occupied by the path. You can also use the path as a clipping region for views or other regions. Using methods of NSBezierPath, you can also perform hit detection on the filled or stroked path. Hit detection is needed to implement interactive graphics, as in rubberbanding and dragging operations.

The current graphics context is automatically saved and restored for all drawing operations, so your application does not need to worry about the graphics settings changing across invocations.

Tasks

Constructors

`NSBezierPath` (page 166)
Creates a new NSBezierPath object.

Creating an NSBezierPath Object

`bezierPath` (page 166)
Creates and returns a new NSBezierPath object.

`bezierPathWithOvalInRect` (page 166)
Creates and returns a new NSBezierPath object with an oval path, inscribed in the rectangle *aRect*, to the receiver's path.

`bezierPathWithRect` (page 166)
Creates and returns a new NSBezierPath object with a rectangular path specified by *aRect*.

`bezierPathByFlatteningPath` (page 174)
Returns a "flattened" copy of the receiver.

`bezierPathByReversingPath` (page 174)
Returns a new NSBezierPath created with the contents of the receiver's path reversed.

Constructing Paths

`moveToPoint` (page 179)
Moves the receiver's current point to *aPoint*, starting a new subpath, without adding any line segments.

`lineToPoint` (page 178)
Appends a straight line to the receiver's path from the current point to *aPoint*.

`curveToPoint` (page 176)
Adds a Bezier cubic curve to the receiver's path from the current point to *aPoint*, using *controlPoint1* and *controlPoint2* as the Bezier cubic control points.

`closePath` (page 175)
Closes the most recently added subpath by appending a straight line segment from the current point to the subpath's starting point.

`relativeMoveToPoint` (page 180)
Moves the receiver's current point to a new point, specified by the parameter *aPoint* as a relative distance from the current point.

`relativeLineToPoint` (page 180)
Appends a straight line to the receiver's path from the current point to *aPoint*, which is specified as a relative distance from the current point.

`relativeCurveToPoint` (page 179)
Adds a Bezier cubic curve to the receiver's path from the current point to a new location, which is specified as a relative distance from the current point.

Appending Paths and Some Common Shapes

`appendBezierPath` (page 173)
Appends *aPath* to the receiver's path.

`appendBezierPathWithOvalInRect` (page 174)
Appends an oval path, inscribed in the rectangle *aRect*, to the receiver's path.

`appendBezierPathWithArcFromPoint` (page 173)
: Appends an arc of a circle to the receiver's path.

`appendBezierPathWithArcWithCenter` (page 173)
: Appends an arc of a circle to the receiver's path.

`appendBezierPathWithGlyph` (page 173)
: Appends an outline of *aGlyph* in *fontObj* to the receiver's path.

`appendBezierPathWithRect` (page 174)
: Appends a rectangular path, specified by *aRect*, to the receiver's path.

Accessing Attributes

`defaultWindingRule` (page 168)
: Returns the default winding rule.

`setDefaultWindingRule` (page 171)
: Sets the default winding rule to *windingRule*.

`windingRule` (page 184)
: Returns the winding rule used to fill the receiver's path, that is, paint the region enclosed by the path.

`setWindingRule` (page 183)
: Sets the winding rule used to fill the receiver's path, that is, paint the region enclosed by the path.

`defaultLineCapStyle` (page 167)
: Returns the default line cap style.

`setDefaultLineCapStyle` (page 169)
: Sets the default line cap style to *lineCap*.

`lineCapStyle` (page 177)
: Returns the receiver's line cap style.

`setLineCapStyle` (page 181)
: Sets the receiver's line cap style to *lineCapStyle*.

`defaultLineJoinStyle` (page 168)
: Returns the default line join style.

`setDefaultLineJoinStyle` (page 170)
: Sets the default line join style to *lineJoinStyle*.

`lineJoinStyle` (page 178)
: Returns the receiver's line join style.

`setLineJoinStyle` (page 182)
: Sets the receiver's line join style to *lineJoinStyle*.

`defaultLineWidth` (page 168)
: Returns the default line width, in points.

`setDefaultLineWidth` (page 171)
: Sets the default line width to *width* points.

`lineWidth` (page 179)
: Returns the receiver's line width, in points.

`setLineWidth` (page 182)
: Sets the receiver's line width to *lineWidth* points.

`defaultMiterLimit` (page 168)
: Returns the default miter limit.

`setDefaultMiterLimit` (page 171)
: Sets the current default miter limit to *limit*.

`miterLimit` (page 179)
: Returns the receiver's miter limit.

`setMiterLimit` (page 182)
: Sets the receiver's miter limit to *miterLimit*.

`defaultFlatness` (page 167)
: Returns the default flatness attribute.

`setDefaultFlatness` (page 169)
: Sets the default flatness attribute to *flatness*.

`flatness` (page 177)
: Returns the receiver's flatness.

`setFlatness` (page 181)
: Sets the receiver's flatness to *flatness*.

`lineDashPattern` (page 177)
: Returns the line-stroking pattern for the receiver.

`lineDashPhase` (page 178)
: Returns the line-stroking phase for the receiver.

`setLineDash` (page 182)
: Sets the line-stroking pattern for the receiver.

Drawing Paths

`stroke` (page 183)
: Draws a line along the receiver's path using the current color and other drawing attributes (for example, line cap style, line join style, and line width).

`fill` (page 177)
: Renders the receiver's path by painting the region enclosed by the path.

`fillRect` (page 169)
: Fills a rectangular path specified by *aRect* with the current color.

`strokeRect` (page 172)
: Strokes a rectangular path specified by *aRect* using the current drawing style and color.

`strokeLineFromPoint` (page 172)
: Strokes a line from *point1* to *point2* using the current drawing attributes (for example, color, line cap style, and line width).

Clipping Paths

`addClip` (page 172)
: Intersects the current clipping path, stored in the current graphics context, with the receiver's path, and replaces the current clipping path with the resulting path.

`setClip` (page 181)
: Replaces the current clipping path with the area inside this path as determined by the winding rule.

`clipRect` (page 167)
: Intersects the current clipping path, stored in the current graphics context, with the rectangle referred to by *`aRect`*, and replaces the current clipping path with the resulting path.

Hit Detection

`containsPoint` (page 175)
: Returns `true` if the receiver contains *`aPoint`*, `false` otherwise.

Querying Paths

`bounds` (page 175)
: Returns the bounding box of the receiver's path.

`controlPointBounds` (page 175)
: Returns the bounding box of the receiver's path, including any control points.

`currentPoint` (page 176)
: Returns the receiver's current point (the trailing point or ending point in the most recently added segment).

`isEmpty` (page 177)
: Returns whether the receiver is empty.

Applying Transformations

`transformUsingAffineTransform` (page 183)
: Transforms all points in the receiver using *`aTransform`*.

Accessing Elements of a Path

`elementCount` (page 176)
: Returns the number of element types currently stored by the receiver's path.

`removeAllPoints` (page 180)
: Remove all points from the receiver's path.

Caching Paths

`cachesBezierPath` (page 175)
: Returns `true` if this object maintains a cached image of its path; otherwise returns `false`.

`setCachesBezierPath` (page 180)
: Sets whether the receiver should cache its path information.

Constructors

NSBezierPath

Creates a new NSBezierPath object.

```
public NSBezierPath()
```

Discussion
The path is initially empty.

Static Methods

bezierPath

Creates and returns a new NSBezierPath object.

```
public static NSBezierPath bezierPath()
```

Discussion
The path is initially empty.

bezierPathWithOvalInRect

Creates and returns a new NSBezierPath object with an oval path, inscribed in the rectangle *aRect*, to the receiver's path.

```
public static NSBezierPath bezierPathWithOvalInRect(NSRect aRect)
```

Discussion
If *aRect* specifies a square, the inscribed path is a circle. The inscribed path starts at the top center of *aRect*, and arc segments are constructed counterclockwise to complete the oval.

See Also
`bezierPath` (page 166)
`appendBezierPathWithOvalInRect` (page 174)

bezierPathWithRect

Creates and returns a new NSBezierPath object with a rectangular path specified by *aRect*.

```
public static NSBezierPath bezierPathWithRect(NSRect aRect)
```

Discussion
The path starts at the origin of *aRect* and is constructed counterclockwise.

See Also
`bezierPath` (page 166)

`appendBezierPathWithRect` (page 174)
`fillRect` (page 169)
`strokeRect` (page 172)

clipRect

Intersects the current clipping path, stored in the current graphics context, with the rectangle referred to by *aRect*, and replaces the current clipping path with the resulting path.

```
public static void clipRect(NSRect aRect)
```

See Also

`addClip` (page 172)
`setClip` (page 181)

defaultFlatness

Returns the default flatness attribute.

```
public static float defaultFlatness()
```

Discussion

The flatness attribute is the accuracy (or smoothness) with which curves are rendered. It is also the maximum error tolerance, measured in pixels, where smaller numbers give smoother curves at the expense of more computation. The exact interpretation may vary slightly on different rendering devices.

The default flatness value is 0.6.

See Also

`setDefaultFlatness` (page 169)
`flatness` (page 177)

defaultLineCapStyle

Returns the default line cap style.

```
public static int defaultLineCapStyle()
```

Discussion

The default line cap style is `LineCapStyleButt`.

See Also

`setDefaultLineCapStyle` (page 169)
`defaultLineJoinStyle` (page 168)
`defaultLineWidth` (page 168)
`lineCapStyle` (page 177)

defaultLineJoinStyle

Returns the default line join style.

```
public static int defaultLineJoinStyle()
```

Discussion

The default line join style is `LineJoinStyleMiter`.

See Also

`setDefaultLineJoinStyle` (page 170)
`defaultLineCapStyle` (page 167)
`defaultLineWidth` (page 168)
`lineJoinStyle` (page 178)

defaultLineWidth

Returns the default line width, in points.

```
public static float defaultLineWidth()
```

Discussion

The default line width is 1.0.

See Also

`setDefaultLineWidth` (page 171)
`defaultLineCapStyle` (page 167)
`defaultLineJoinStyle` (page 168)
`lineWidth` (page 179)

defaultMiterLimit

Returns the default miter limit.

```
public static float defaultMiterLimit()
```

Discussion

The default miter limit is 10.0.

See Also

`setDefaultMiterLimit` (page 171)
`miterLimit` (page 179)

defaultWindingRule

Returns the default winding rule.

```
public static int defaultWindingRule()
```

Discussion

The default winding rule is `WindingRuleNonZero`.

See Also

`setDefaultWindingRule` (page 171)

`windingRule` (page 184)

fillRect

Fills a rectangular path specified by *aRect* with the current color.

```
public static void fillRect(NSRect aRect)
```

See Also

`appendBezierPathWithRect` (page 174)

`bezierPathWithRect` (page 166)

`strokeRect` (page 172)

`set` (page 376) (NSColor)

setDefaultFlatness

Sets the default flatness attribute to *flatness*.

```
public static void setDefaultFlatness(float flatness)
```

Discussion

The flatness attribute is the accuracy (or smoothness) with which curves are rendered. *flatness* is the maximum error tolerance, measured in pixels, where smaller numbers give smoother curves at the expense of more computation. The exact interpretation may vary slightly on different rendering devices.

The default flatness value is 0.6.

See Also

`defaultFlatness` (page 167)

`setFlatness` (page 181)

setDefaultLineCapStyle

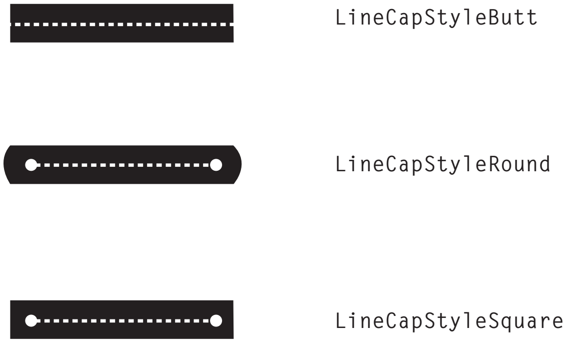
Sets the default line cap style to *lineCap*.

```
public static void setDefaultLineCapStyle(int lineCap)
```

Discussion

The line cap style specifies the shape of the endpoints of an open path when stroked. Figure 9-1 (page 170) shows the appearance of the available line cap styles.

Figure 9-1 Line cap styles



See Also

`defaultLineCapStyle` (page 167)

`setDefaultLineJoinStyle` (page 170)

`setDefaultLineWidth` (page 171)

`setLineCapStyle` (page 181)

setDefaultLineJoinStyle

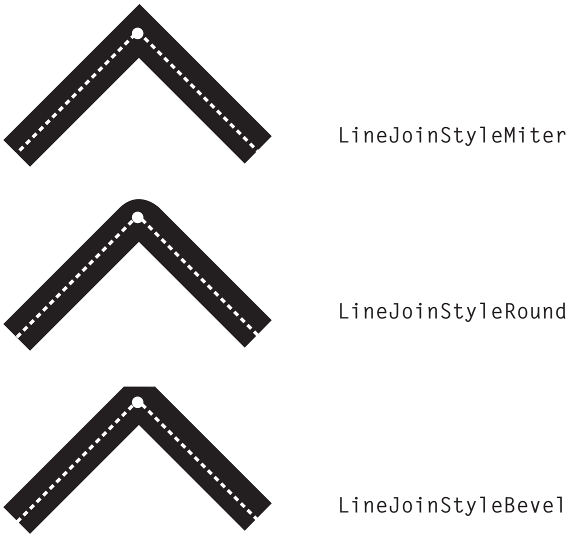
Sets the default line join style to *`lineJoinStyle`*.

```
public static void setDefaultLineJoinStyle(int lineJoinStyle)
```

Discussion

The line join style specifies the shape of the joints between connected segments of a stroked path. Figure 9-2 (page 170) shows the appearance of the available line join styles.

Figure 9-2 Line join styles



See Also

`defaultLineJoinStyle` (page 168)

`setDefaultLineCapStyle` (page 169)

`setDefaultLineWidth` (page 171)

`setDefaultMiterLimit` (page 171)

`setLineJoinStyle` (page 182)

setDefaultLineWidth

Sets the default line width to *width* points.

```
public static void setDefaultLineWidth(float width)
```

Discussion

The line width is the thickness of stroked paths. A width of 0 is interpreted as the thinnest line that can be rendered on a particular device. The actual rendered line width may vary from *width* by as much as 2 device pixels, depending on the position of the line with respect to the pixel grid. The width of the line may also be affected by scaling factors specified in the current transformation matrix of the active graphics context.

See Also

`defaultLineWidth` (page 168)

`setDefaultLineCapStyle` (page 169)

`setDefaultLineJoinStyle` (page 170)

`setLineWidth` (page 182)

setDefaultMiterLimit

Sets the current default miter limit to *limit*.

```
public static void setDefaultMiterLimit(float limit)
```

Discussion

Setting the miter limit avoids spikes produced by line segments that join at sharp angles. If the ratio of the miter length—the diagonal length of the miter—to the line thickness exceeds the miter limit, the corner is treated as a bevel join instead of a miter join. The default miter limit value is 10, which cuts off miters at angles less than 11 degrees.

See Also

`defaultMiterLimit` (page 168)

`setDefaultLineJoinStyle` (page 170)

`setMiterLimit` (page 182)

setDefaultWindingRule

Sets the default winding rule to *windingRule*.

```
public static void setDefaultWindingRule(int windingRule)
```

Discussion

The winding rule is used to fill the receiver's path, that is, paint the region enclosed by the path. Possible values for the *windingRule* parameter are `WindingRuleNonZero` and `WindingRuleEvenOdd`. See "Winding Rules and Filling Paths" for more information on how winding rules affect the appearance of filled paths.

See Also

`defaultWindingRule` (page 168)

`setWindingRule` (page 183)

strokeLineFromPoint

Strokes a line from *point1* to *point2* using the current drawing attributes (for example, color, line cap style, and line width).

```
public static void strokeLineFromPoint(NSPoint point1, NSPoint point2)
```

See Also

`lineToPoint` (page 178)

`moveToPoint` (page 179)

`setDefaultLineCapStyle` (page 169)

`setDefaultLineWidth` (page 171)

`stroke` (page 183)

strokeRect

Strokes a rectangular path specified by *aRect* using the current drawing style and color.

```
public static void strokeRect(NSRect aRect)
```

Discussion

The path is stroked beginning at the rectangle's origin and proceeding in a counterclockwise direction.

See Also

`appendBezierPathWithRect` (page 174)

`bezierPathWithRect` (page 166)

`fillRect` (page 169)

`setDefaultLineJoinStyle` (page 170)

`setDefaultLineWidth` (page 171)

`set` (page 376) (NSColor)

Instance Methods

addClip

Intersects the current clipping path, stored in the current graphics context, with the receiver's path, and replaces the current clipping path with the resulting path.

```
public void addClip()
```

Discussion
The current winding rule is applied to determine the clipping area of the receiver. This method does not affect the receiver's path.

See Also
`clipRect` (page 167)
`setClip` (page 181)

appendBezierPath

Appends *aPath* to the receiver's path.

```
public void appendBezierPath(NSBezierPath aPath)
```

Discussion
This method adds the operations used to create *aPath* to the end of the receiver's path. This method does not explicitly try to connect the two paths, although the operations in *aPath* may still cause this effect.

appendBezierPathWithArcFromPoint

Appends an arc of a circle to the receiver's path.

```
public void appendBezierPathWithArcFromPoint(NSPoint fromPoint, NSPoint toPoint,
    float radius)
```

Discussion
The arc lies on the perimeter of a circle with radius *radius* inscribed in an angle defined by the current point, *fromPoint*, and *toPoint*. The arc is drawn between the tangent points of the circle with the two legs of the angle. The arc usually does not contain the points *fromPoint* and *toPoint*. A line is drawn from the current point to the starting point of the arc, which is located at the tangent point of the first leg of the angle.

appendBezierPathWithArcWithCenter

Appends an arc of a circle to the receiver's path.

```
public void appendBezierPathWithArcWithCenter(NSPoint center, float radius, float
     startAngle, float endAngle, boolean clockwise)
```

Discussion
The circle is centered at *center* with radius *radius*. The arc lies on the perimeter of the circle, between *startAngle* and *endAngle*, measured in degrees counterclockwise from the x axis. If *clockwise* is `true`, the arc is drawn clockwise around the circle from *startAngle* to *endAngle*. If *clockwise* is `false`, then the arc is drawn counterclockwise around the circle.

appendBezierPathWithGlyph

Appends an outline of *aGlyph* in *fontObj* to the receiver's path.

```
public void appendBezierPathWithGlyph(int aGlyph, NSFont fontObj)
```

Discussion

If *aGlyph* is not encoded in *fontObj*—that is, the font does not have an entry for the specified glyph—then no path is appended to the receiver.

The receiver must not be empty—that is, it must already contain an initial point, or an exception is thrown. An initial point may be specified using `moveToPoint` (page 179).

appendBezierPathWithOvalInRect

Appends an oval path, inscribed in the rectangle *aRect*, to the receiver's path.

```
public void appendBezierPathWithOvalInRect(NSRect aRect)
```

Discussion

If *aRect* specifies a square, the inscribed path is a circle. The inscribed path starts at the top center of *aRect*, and arc segments are constructed counterclockwise to complete the oval.

appendBezierPathWithRect

Appends a rectangular path, specified by *aRect*, to the receiver's path.

```
public void appendBezierPathWithRect(NSRect aRect)
```

Discussion

The path starts at the origin of *aRect*, and line segments are added proceeding counterclockwise from the origin, ending with a `closePath` (page 175) message to complete the path.

See Also

`bezierPathWithRect` (page 166)

`fillRect` (page 169)

`strokeRect` (page 172)

bezierPathByFlatteningPath

Returns a "flattened" copy of the receiver.

```
public NSBezierPath bezierPathByFlatteningPath()
```

Discussion

Flattening a path converts all curved line segments into straight line approximations. For example, a curved line can be approximated with a series of straight lines. The granularity of approximation is called the flatness. This method uses the value set by `setDefaultFlatness` (page 169).

bezierPathByReversingPath

Returns a new NSBezierPath created with the contents of the receiver's path reversed.

```
public NSBezierPath bezierPathByReversingPath()
```

bounds

Returns the bounding box of the receiver's path.

```
public NSRect bounds()
```

Discussion
If the path contains curve segments, the bounding box encloses the curve but may not enclose the control points used to calculate the curve.

See Also
`controlPointBounds` (page 175)

cachesBezierPath

Returns `true` if this object maintains a cached image of its path; otherwise returns `false`.

```
public boolean cachesBezierPath()
```

Discussion
The cached image is stored in a display user object.

See Also
`setCachesBezierPath` (page 180)

closePath

Closes the most recently added subpath by appending a straight line segment from the current point to the subpath's starting point.

```
public void closePath()
```

Discussion
A subpath is a sequence of connected segments. A path may be made up of one or more disconnected subpaths. The current point is the ending point in the most recently added segment.

See Also
`fill` (page 177)

containsPoint

Returns `true` if the receiver contains *`aPoint`*, `false` otherwise.

```
public boolean containsPoint(NSPoint aPoint)
```

controlPointBounds

Returns the bounding box of the receiver's path, including any control points.

```
public NSRect controlPointBounds()
```

Discussion
If the path contains curve segments, the bounding box encloses the control points of the curves as well as the curves themselves.

See Also
`bounds` (page 175)

currentPoint

Returns the receiver's current point (the trailing point or ending point in the most recently added segment).

```
public NSPoint currentPoint()
```

Discussion
Throws `GenericException` if the receiver is empty.

See Also
`closePath` (page 175)
`curveToPoint` (page 176)
`lineToPoint` (page 178)
`moveToPoint` (page 179)

curveToPoint

Adds a Bezier cubic curve to the receiver's path from the current point to *aPoint*, using *controlPoint1* and *controlPoint2* as the Bezier cubic control points.

```
public void curveToPoint(NSPoint aPoint, NSPoint controlPoint1, NSPoint
    controlPoint2)
```

Discussion
The current point is the ending point in the most recently added segment. To create a relative curve, use `relativeCurveToPoint` (page 179).

See Also
`closePath` (page 175)
`lineToPoint` (page 178)
`moveToPoint` (page 179)
`relativeCurveToPoint` (page 179)
`setDefaultFlatness` (page 169)

elementCount

Returns the number of element types currently stored by the receiver's path.

```
public int elementCount()
```

Discussion
Each element type corresponds to one of the operations described in "Path Elements".

fill

Renders the receiver's path by painting the region enclosed by the path.

```
public void fill()
```

Discussion

Uses the winding rule, specified by invoking `setWindingRule` (page 183), and the current color to fill the path. Closes any open subpaths. A subpath is a sequence of connected segments. A path may be made up of one or more disconnected subpaths. A subpath is closed if the ending point is connected to the starting point (as in a polygon).

See Also

`stroke` (page 183)

`windingRule` (page 184)

`set` (page 376) (NSColor)

flatness

Returns the receiver's flatness.

```
public float flatness()
```

See Also

`setFlatness` (page 181)

`defaultFlatness` (page 167)

isEmpty

Returns whether the receiver is empty.

```
public boolean isEmpty()
```

lineCapStyle

Returns the receiver's line cap style.

```
public int lineCapStyle()
```

Discussion

See "Constants" (page 184) for a description of possible return values.

See Also

`defaultLineCapStyle` (page 167)

`setDefaultLineCapStyle` (page 169)

`setLineCapStyle` (page 181)

lineDashPattern

Returns the line-stroking pattern for the receiver.

```
public float[] lineDashPattern()
```

Discussion
The pattern is specified as alternating lengths of painted and unpainted line segments.

See Also
`setLineDash` (page 182)

lineDashPhase

Returns the line-stroking phase for the receiver.

```
public float lineDashPhase()
```

Discussion
The phase specifies how far, in view coordinate units, into a line dash pattern to start.

See Also
`setLineDash` (page 182)
`lineDashPattern` (page 177)

lineJoinStyle

Returns the receiver's line join style.

```
public int lineJoinStyle()
```

Discussion
See "Constants" (page 184) for a description of possible return values.

See Also
`defaultLineJoinStyle` (page 168)
`setDefaultLineJoinStyle` (page 170)
`setLineJoinStyle` (page 182)

lineToPoint

Appends a straight line to the receiver's path from the current point to *aPoint*.

```
public void lineToPoint(NSPoint aPoint)
```

Discussion
The current point is the last point in the receiver's most recently added segment.

See Also
`closePath` (page 175)
`curveToPoint` (page 176)
`moveToPoint` (page 179)

lineWidth

Returns the receiver's line width, in points.

```
public float lineWidth()
```

See Also

`setLineWidth` (page 182)

`defaultLineWidth` (page 168)

miterLimit

Returns the receiver's miter limit.

```
public float miterLimit()
```

See Also

`setMiterLimit` (page 182)

`defaultMiterLimit` (page 168)

moveToPoint

Moves the receiver's current point to *aPoint*, starting a new subpath, without adding any line segments.

```
public void moveToPoint(NSPoint aPoint)
```

Discussion

A subpath is a sequence of connected segments. A path may be made up of one or more disconnected subpaths. The current point is the ending point in the most recently added segment.

See Also

`closePath` (page 175)

`curveToPoint` (page 176)

`lineToPoint` (page 178)

relativeCurveToPoint

Adds a Bezier cubic curve to the receiver's path from the current point to a new location, which is specified as a relative distance from the current point.

```
public void relativeCurveToPoint(NSPoint aPoint, NSPoint controlPoint1, NSPoint
    controlPoint2)
```

Discussion

(The control points are similarly specified as relative distances from the current point.) The *aPoint* parameter specifies the endpoint of the curve as a relative distance from the current point. The *controlPoint1* and *controlPoint2* parameters specify the location of the two control points as relative distances from the current point. Throws `GenericException` if the path is empty.

See Also

`closePath` (page 175)

curveToPoint (page 176)

relativeLineToPoint (page 180)

relativeMoveToPoint (page 180)

relativeLineToPoint

Appends a straight line to the receiver's path from the current point to *aPoint*, which is specified as a relative distance from the current point.

```
public void relativeLineToPoint(NSPoint aPoint)
```

Discussion

Throws GenericException if the path is empty.

See Also

closePath (page 175)

lineToPoint (page 178)

relativeLineToPoint (page 180)

relativeMoveToPoint (page 180)

relativeMoveToPoint

Moves the receiver's current point to a new point, specified by the parameter *aPoint* as a relative distance from the current point.

```
public void relativeMoveToPoint(NSPoint aPoint)
```

Discussion

This method starts a new subpath without adding any line segments.

See Also

closePath (page 175)

moveToPoint (page 179)

relativeCurveToPoint (page 179)

relativeLineToPoint (page 180)

removeAllPoints

Remove all points from the receiver's path.

```
public void removeAllPoints()
```

setCachesBezierPath

Sets whether the receiver should cache its path information.

```
public void setCachesBezierPath(boolean flag)
```

Discussion

Caching improves subsequent drawing times but requires extra memory to store the cached path representation. If caching is being turned on (`flag` is `true`), the receiver's cache is marked as needing to be calculated. Otherwise, if caching is being turned off, any existing cached data is deleted.

See Also

`cachesBezierPath` (page 175)

setClip

Replaces the current clipping path with the area inside this path as determined by the winding rule.

```
public void setClip()
```

Discussion

This method is not a preferred method of adjusting the clipping path, as it may expand the clipping path beyond the bounds set by the enclosing NSView. The graphics state should be saved and restored before and after invoking this method.

See Also

`addClip` (page 172)

`clipRect` (page 167)

`saveGraphicsState` (page 735) (NSGraphicsContext)

`restoreGraphicsState` (page 734) (NSGraphicsContext)

setFlatness

Sets the receiver's flatness to *`flatness`*.

```
public void setFlatness(float flatness)
```

See Also

`flatness` (page 177)

`setDefaultFlatness` (page 169)

setLineCapStyle

Sets the receiver's line cap style to *`lineCapStyle`*.

```
public void setLineCapStyle(int lineCapStyle)
```

Discussion

See "Constants" (page 184) for a list of possible values for *`lineCapStyle`*.

See Also

`defaultLineCapStyle` (page 167)

`setDefaultLineCapStyle` (page 169)

`lineCapStyle` (page 177)

setLineDash

Sets the line-stroking pattern for the receiver.

```
public void setLineDash(float[] pattern, float phase)
```

Discussion

pattern specifies alternating lengths of painted and unpainted line segments.*phase* specifies how far, in view coordinate units, into *pattern* to start.

For example, to produce a supermarket coupon type of dashed line:

```
array[0] = 5.0; //segment painted with stroke color
array[1] = 2.0; //segment not painted with a color

path.setLineDash(array, 0.0);
```

In the above example, if you set *phase* to 5.0, the line dash would begin approximately five units into *pattern*, and the first filled segment would be effectively skipped.

See Also

`lineDashPattern` (page 177)

`lineDashPhase` (page 178)

setLineJoinStyle

Sets the receiver's line join style to *lineJoinStyle*.

```
public void setLineJoinStyle(int lineJoinStyle)
```

Discussion

See "Constants" (page 184) for a list of possible values for *lineJoinStyle*.

See Also

`defaultLineJoinStyle` (page 168)

`setDefaultLineJoinStyle` (page 170)

`lineJoinStyle` (page 178)

setLineWidth

Sets the receiver's line width to *lineWidth* points.

```
public void setLineWidth(float lineWidth)
```

See Also

`lineWidth` (page 179)

`setDefaultLineWidth` (page 171)

setMiterLimit

Sets the receiver's miter limit to *miterLimit*.

```
public void setMiterLimit(float miterLimit)
```

See Also

`miterLimit` (page 179)

`setDefaultMiterLimit` (page 171)

setWindingRule

Sets the winding rule used to fill the receiver's path, that is, paint the region enclosed by the path.

```
public void setWindingRule(int aWindingRule)
```

Discussion

Possible values for the *aWindingRule* parameter are `WindingRuleNonZero` and `WindingRuleEvenOdd`. See "Winding Rules and Filling Paths" for more information on how winding rules affect the appearance of filled paths.

See Also

`fill` (page 177)

`windingRule` (page 184)

`setDefaultWindingRule` (page 171)

stroke

Draws a line along the receiver's path using the current color and other drawing attributes (for example, line cap style, line join style, and line width).

```
public void stroke()
```

Discussion

The drawn line is centered on the path with sides (specified by the `setDefaultLineWidth` (page 171) static method) parallel to the path segment.

See Also

`fill` (page 177)

`setDefaultLineCapStyle` (page 169)

`setDefaultLineJoinStyle` (page 170)

`set` (page 376) (NSColor)

transformUsingAffineTransform

Transforms all points in the receiver using *aTransform*.

```
public void transformUsingAffineTransform(NSAffineTransform aTransform)
```

Discussion

The following code translates a line from 0,0 to 100,100 to a line from 10,10 to 110,110.

```
NSBezierPath bezierPath = NSBezierPath.bezierPath();
NSAffineTransform transform = NSAffineTransform.transform();
```

```
bezierPath.moveToPoint (new NSPoint(0.0, 0.0));
bezierPath.lineToPoint (new NSPoint(100.0, 100.0));

transform.translateXYBy (10.0, 10.0);
bezierPath.transformUsingAffineTransform (transform);
```

windingRule

Returns the winding rule used to fill the receiver's path, that is, paint the region enclosed by the path.

```
public int windingRule()
```

Discussion

Possible return values are `WindingRuleNonZero` and `WindingRuleEvenOdd`. See "Winding Rules and Filling Paths" for more information on how winding rules affect the appearance of filled paths.

See Also

`fill` (page 177)

`setWindingRule` (page 183)

`defaultWindingRule` (page 168)

Constants

As a convenience, the following constants are provided by NSBezierPath:

Constant	Description
`LineCapStyleButt`	Specifies the shape of endpoints for an open path when stroked. See `setDefaultLineCapStyle` (page 169) for an example of the appearance.
`LineCapStyle-ProjectingSquare`	Specifies the shape of endpoints for an open path when stroked. See `setDefaultLineCapStyle` (page 169) for an example of the appearance.
`LineCapStyleRound`	Specifies the shape of endpoints for an open path when stroked. See `setDefaultLineCapStyle` (page 169) for an example of the appearance.
`LineJoinStyleBevel`	Specifies the shape of the joints between connected segments of a stroked path. See `setDefaultLineJoinStyle` (page 170) for an example of the appearance.
`LineJoinStyleMiter`	Specifies the shape of the joints between connected segments of a stroked path. See `setDefaultLineJoinStyle` (page 170) for an example of the appearance.
`LineJoinStyleRound`	Specifies the shape of the joints between connected segments of a stroked path. See `setDefaultLineJoinStyle` (page 170) for an example of the appearance.

NSBitmapImageRep

Inherits from	NSImageRep : NSObject
Implements	NSCoding (NSImageRep)
Package:	com.apple.cocoa.application
Companion guide	Drawing and Images

Overview

An NSBitmapImageRep is an object that can render an image from bitmap data. Bitmap data formats supported include GIF, JPEG, TIFF, PNG, and various permutations of raw bitmap data.

Alpha Premultiplication

If a coverage (alpha) plane exists, a bitmap's color components are premultiplied with it. If you modify the contents of the bitmap, you are therefore responsible for premultiplying the data. For this reason, though, if you want to manipulate the actual data, NSBitmapImageRep is not recommended for storage. If you need to work with unpremultiplied data, you should use Quartz, specifically `CGImageCreate` with `kCGImageAlphaLast`.

Note that premultiplying does not affect the output quality. Given source bitmap pixel `s`, destination pixel `d`, and alpha value `a`, a blend is basically

```
d' = a * s + (1 - a) * d
```

All premultiplication does is precalculate `a * s`.

Tasks

Constructors

`NSBitmapImageRep` (page 188)
Throws an exception. Use one of the other constructors instead.

Creating an NSBitmapImageRep

imageRep (page 190)
Creates and returns an initialized NSBitmapImageRep corresponding to the first image in *bitmapData*, or null if NSBitmapImageRep is unable to interpret *bitmapData*.

imageRepsWithData (page 190)
Creates and returns an array of initialized NSBitmapImageRep objects corresponding to the images in *bitmapData*.

colorizeByMappingGray (page 194)
Support for colorization of grayscale images.

Getting Information About the Image

bitmapFormat (page 192)
Returns the bitmap format of the receiver.

bitsPerPixel (page 193)
Returns the number of bits allocated for each pixel in each plane of data.

bytesPerPlane (page 193)
Returns the number of bytes in each plane or channel of data.

bytesPerRow (page 193)
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.

compressionFactor (page 194)
Returns the receiver's compression factor.

compressionType (page 194)
Returns the receiver's compression type.

isPlanar (page 196)
Returns true if image data is segregated into a separate plane for each color and coverage component (planar configuration) and false if the data is integrated into a single plane (meshed configuration).

numberOfPlanes (page 196)
Returns the number of separate planes image data is organized into.

samplesPerPixel (page 197)
Returns the number of components in the data.

Getting Image Data

bitmapData (page 192)
Returns a pointer to the bitmap data.

bitmapDataPlanes (page 192)
Provides access to bitmap data for the receiver separated into planes.

setBitmapData (page 197)
Copies the data from *bits* to the image's bitmap.

setBitmapDataPlanes (page 197)
Copies the data from *planes* to a planar image's bitmap planes.

Producing a TIFF Representation of the Image

`TIFFRepresentationOfImageReps` (page 191)
: Returns a TIFF representation of the images in *`array`*, using the compression returned by `compressionFactor` (page 194) (if applicable).

`TIFFRepresentation` (page 198)
: Returns a TIFF representation of the image, using the compression that's returned by `compressionFactor` (page 194) (if applicable).

`representationOfImageRepsInArray` (page 191)
: Returns a bitmap version of the NSBitmapImageRep objects in *`imageReps`*, using the format *`storageType`* and the properties *`properties`*.

`representationUsingType` (page 196)
: Returns a bitmap version of the receiver, using the format *`storageType`* and the properties *`properties`*.

Setting and Checking Compression Types

`TIFFCompressionTypes` (page 191)
: Returns the list of available compression types, which are defined in "Constants" (page 199).

`localizedNameForTIFFCompressionType` (page 190)
: Returns a string containing the localized name for the compression type represented by *`compression`*, or `null` if *`compression`* is unrecognized.

`canBeCompressedUsingType` (page 193)
: Tests whether the receiver can be compressed by compression type *`compression`*.

`setCompressionWithFactor` (page 198)
: Sets the receiver's compression type and compression factor.

`valueForProperty` (page 199)
: Returns the value for *`property`*.

`setProperty` (page 198)
: Sets the image's *`property`* to *`value`*.

Incremental Image Loading

`incrementalLoadFromData` (page 195)
: Loads the current image data into an incrementally-loaded image representation and returns the current status of the image.

Getting and Setting Pixel Values

`color` (page 194)

`setColor` (page 197)

`getPixel` (page 195)
Provides access to the pixel data for the location *x*,*y* in the receiver.

`setPixel` (page 198)
Sets the receiver's pixel at the coordinate *x*,*y* to the raw pixel values in *pixelData*.

Constructors

NSBitmapImageRep

Throws an exception. Use one of the other constructors instead.

```
public NSBitmapImageRep()
```

Creates an NSBitmapImageRep, with bitmap data read from a rendered image.

```
public NSBitmapImageRep(NSRect rect)
```

Discussion
The image that's read is located in the current window and is bounded by the *rect* rectangle as specified in the current coordinate system.

This method uses imaging operators to read the image data into a buffer; the object is then created from that data. The object is initialized with information about the image obtained from the window server.

If for any reason the new object can't be created, this method returns `null`.

Creates an NSBitmapImageRep from the data found in *bitmapData*.

```
public NSBitmapImageRep(NSData bitmapData)
```

Discussion
The contents of *bitmapData* can be any supported bitmap format. For TIFF data, the NSBitmapImageRep is initialized from the first header and image data found in *bitmapData*.

This method returns an initialized NSBitmapImageRep if the creation was successful or `null` if it was unable to interpret the contents of *bitmapData*.

Creates an NSBitmapImageRep, so it can render the image described by the arguments. If the object can't be created, this method returns `null`.

```
public NSBitmapImageRep(int width, int height, int bps, int spp, boolean alpha,
    boolean isPlanar, String colorSpaceName, int rowBytes, int pixelBits)
```

Discussion
This constructor allocates enough memory to hold the image described by the arguments. You can then obtain pointers to this memory (with the `bitmapDataPlanes` (page 192) or `bitmapData` (page 192) method) and fill in the image data. If the image has an alpha channel, you are responsible for premultiplying the color samples. Each of the arguments informs the NSBitmapImageRep object about the image. They're explained below:

- The *width* and *height* arguments specify the size of the image in pixels. The size in each direction must be greater than 0.

- The *bps* (bits per sample) argument is the number of bits used to specify 1 pixel in a single component of the data. All components are assumed to have the same bits per sample. *bps* should be one of these values: 1, 2, 4, 8, 12, or 16.
- The *spp* (samples per pixel) argument 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 grayscale image that lacks a coverage component would have an *spp* of 1.
- The *alpha* argument should be `true` if one of the components counted in the number of samples per pixel (*spp*) is a coverage component and `false` if there is no coverage component. If `true`, the color samples you put into the bitmap must be premultiplied with their coverage component.
- The *isPlanar* argument should be `true` if the data components are laid out in a series of separate "planes" or channels ("planar configuration") and `false` 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.
- The *colorSpaceName* argument indicates how data values are to be interpreted. It should be one of the following enumerated values:
  - ❑ `NSGraphics.CalibratedWhiteColorSpace`
  - ❑ `NSGraphics.CalibratedBlackColorSpace`
  - ❑ `NSGraphics.CalibratedRGBColorSpace`
  - ❑ `NSGraphics.DeviceWhiteColorSpace`
  - ❑ `NSGraphics.DeviceBlackColorSpace`
  - ❑ `NSGraphics.DeviceRGBColorSpace`
  - ❑ `NSGraphics.DeviceCMYKColorSpace`
  - ❑ `NSGraphics.NamedColorSpace`
  - ❑ `NSGraphics.CustomColorSpace`

  If *bps* is 12, you cannot specify the monochrome color space.
- The *rowBytes* argument 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 there's no empty space at the end of a row.
- The *pixelBits* argument 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. NSBitmapImageRep supports only a

limited number of *pixelBits* values (other than the default): for RGB images with 4 *bps*, *pixelBits* may be 16; for RGB images with 8 *bps*, *pixelBits* may be 32. The legal values for *pixelBits* are system dependent.

If *pixelBits* is 0, the object will interpret the number of bits per pixel to be the expected value, without any meaningless bits.

Creates a NSBitmapImageRep object for incremental loading.

```
public NSBitmapImageRep(boolean bool)
```

Discussion
The receiver returns itself after setting its size and data buffer to zero. You can then call `incrementalLoadFromData` (page 195) to incrementally add image data.

Availability
Available in Mac OS X v10.2 and later.

See Also
`incrementalLoadFromData` (page 195)

Static Methods

imageRep

Creates and returns an initialized NSBitmapImageRep corresponding to the first image in *bitmapData*, or `null` if NSBitmapImageRep is unable to interpret *bitmapData*.

```
public static NSBitmapImageRep imageRep(NSData bitmapData)
```

Discussion
bitmapData can contain data in any supported bitmap format.

imageRepsWithData

Creates and returns an array of initialized NSBitmapImageRep objects corresponding to the images in *bitmapData*.

```
public static NSArray imageRepsWithData(NSData bitmapData)
```

Discussion
If NSBitmapImageRep is unable to interpret *bitmapData*, the returned array is empty. *bitmapData* can contain data in any supported bitmap format.

localizedNameForTIFFCompressionType

Returns a string containing the localized name for the compression type represented by *compression*, or `null` if *compression* is unrecognized.

```
public static String localizedNameForTIFFCompressionType(int compression)
```

Discussion

Compression types are listed in "Constants" (page 199). When implementing a user interface for selecting TIFF compression types, use `TIFFCompressionTypes` (page 191) to get the list of supported compression types, then use this method to get the localized names for each compression type.

See Also

`TIFFCompressionTypes` (page 191)

representationOfImageRepsInArray

Returns a bitmap version of the NSBitmapImageRep objects in *imageReps*, using the format *storageType* and the properties *properties*.

```
public static NSData representationOfImageRepsInArray(NSArray imageReps, int
    storageType, NSDictionary properties)
```

Discussion

The *storageType* can be `BMPFileType`, `GIFFileType`, `JPEGFileType`, `PNGFileType`, or `TIFFFileType`. The contents of the *properties* dictionary is described in "Constants" (page 199).

TIFFCompressionTypes

Returns the list of available compression types, which are defined in "Constants" (page 199).

```
public static int[] TIFFCompressionTypes()
```

TIFFRepresentationOfImageReps

Returns a TIFF representation of the images in *array*, using the compression returned by `compressionFactor` (page 194) (if applicable).

```
public static NSData TIFFRepresentationOfImageReps(NSArray array)
```

Discussion

If a problem is encountered during generation of the TIFF, throws a `TIFFException` or a `BadBitmapParametersException`.

Returns a TIFF representation of the images in *array*, which are compressed using the specified compression type and *factor*.

```
public static NSData TIFFRepresentationOfImageReps(NSArray array, int compression,
     float factor)
```

Discussion

Legal values for *compression* are described in "Constants" (page 199). *factor* provides a hint for those compression types that implement variable compression ratios; currently only JPEG compression uses a compression factor. JPEG compression in TIFF files is not supported, and *factor* is ignored.

If the specified compression isn't applicable, no compression is used. If a problem is encountered during generation of the TIFF, throws a `TIFFException` or a `BadBitmapParametersException`.

Instance Methods

bitmapData

Returns a pointer to the bitmap data.

```
public byte[] bitmapData()
```

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

See Also
`bitmapDataPlanes` (page 192)

bitmapDataPlanes

Provides access to bitmap data for the receiver separated into planes.

```
public byte[][] bitmapDataPlanes()
```

Discussion
Returns an array of five character pointers. If the receiver is in planar configuration, each pointer is initialized to point to one of the data planes. If there are less than five planes, the remaining pointers are set to `null`. If the receiver is in meshed configuration, only the first pointer is initialized; the others are `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. If a coverage plane exists, the bitmap's color components are premultiplied with it. If you modify the contents of the bitmap, you are responsible for premultiplying the data.

See Also
`isPlanar` (page 196)

bitmapFormat

Returns the bitmap format of the receiver.

```
public int bitmapFormat()
```

Discussion
The default is 0.

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

See Also
`bytesPerRow` (page 193)

bitsPerPixel

Returns the number of bits allocated for each pixel in each plane of data.

```
public int bitsPerPixel()
```

Discussion

This number 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 case extra memory is allocated for each pixel. This may be the case, for example, if pixel data is aligned on byte boundaries.

bytesPerPlane

Returns the number of bytes in each plane or channel of data.

```
public int bytesPerPlane()
```

Discussion

This number is calculated from the number of bytes per row and the height of the image.

See Also

`bytesPerRow` (page 193)

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.

```
public int bytesPerRow()
```

Discussion

If not explicitly set to another value , this number 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.

See Also

`bytesPerPlane` (page 193)

canBeCompressedUsingType

Tests whether the receiver can be compressed by compression type *compression*.

```
public boolean canBeCompressedUsingType(int compression)
```

Discussion

Legal values for *compression* can be found in `NSBitmapImageRep.h` and are described in "TIFF Compression in NSBitmapImageReps". This method returns `true` if the receiver's data matches *compression*; for example, if *compression* is `TIFFCompressionCCITTFAX3`, then the data must be 1 bit per sample and 1 sample per pixel. It returns `false` if the data doesn't match *compression* or if *compression* is unsupported.

color

```
public native NSColor color(int x, int y)
```

Discussion

Returns the color of the pixel located at the coordinates *x*,*y*.

Availability

Available in Mac OS X v10.4 and later.

See Also

`setColor` (page 197)

colorizeByMappingGray

Support for colorization of grayscale images.

```
public void colorizeByMappingGray(float midPoint, NSColor midPointColor, NSColor
    shadowColor, NSColor lightColor)
```

Discussion

Maps the receiver such that:

> Gray value of *`midPoint`* –> *`midPointColor`*;
>
> black –> *`shadowColor`*;
>
> white –> *`lightColor`*.

Works on images with 8-bit SPP, thus either 8-bit gray or 24-bit color (with optional alpha).

compressionFactor

Returns the receiver's compression factor.

```
public float compressionFactor()
```

Discussion

The compression factor is a value that is specific to the compression type; many types of compression don't support varying degrees of compression and thus ignore the factor. JPEG compression allows a compression factor ranging from 0.0 to 1.0, with 0.0 being the lowest and 1.0 being the highest.

See Also

`compressionType` (page 194)

compressionType

Returns the receiver's compression type.

```
public int compressionType()
```

Discussion

The compression type represents the compression type used on the data and corresponds to one of the values returned by the class method `TIFFCompressionTypes` (page 191).

See Also

`compressionFactor` (page 194)

getPixel

Provides access to the pixel data for the location *x*,*y* in the receiver.

```
public long[] getPixel(int x, int y)
```

Discussion

The array returned will contain raw pixel data in the appropriate order for the receiver's `bitmapFormat` (page 192). Smaller integer samples, such as 4-bit, are returned as an integer. Floating point values are cast to integer values and returned.

See Also

`setPixel` (page 198)

incrementalLoadFromData

Loads the current image data into an incrementally-loaded image representation and returns the current status of the image.

```
public int incrementalLoadFromData(NSData data, boolean complete)
```

Discussion

After initializing the receiver with the constructor with the boolean parameter, you should call this method to incrementally load the image. Call this method each time new data becomes available. Always pass the entire image data buffer in *data*, not just the newest data, because the image decompressor may need the original data in order to backtrack. This method will block until the data is decompressed; it will decompress as much of the image as possible based on the length of the data. The image rep does not retain *data*, so you must ensure that *data* is not released for the duration of this method call. Pass `false` for *complete* until the entire image is downloaded, at which time you should pass `true`. You should also pass `true` for *complete* if you have only partially downloaded the data, but cannot finish the download.

This method returns `ImageRepLoadStatusUnknownType` if you did not pass enough data to determine the image format; you should continue to invoke this method with additional data.

This method returns `ImageRepLoadStatusReadingHeader` if it has enough data to determine the image format, but needs more data to determine the size and depth and other characteristics of the image. You should continue to invoke this method with additional data.

This method returns `ImageRepLoadStatusWillNeedAllData` if the image format does not support incremental loading or the Application Kit does not yet implement incremental loading for the image format. You may continue to invoke this method in this case, but until you pass `true` for *complete*, this method will continue to return `ImageRepLoadStatusWillNeedAllData`, and will perform no decompression. Once you pass `true`, the image will be decompressed and one of the final three status messages will be returned.

If the image format does support incremental loading, then once enough data has been read, the image is decompressed from the top down a row at a time. In this case, instead of a status value, this method returns the number of pixel rows that have been decompressed, starting from the top of the image. You can use this information to draw the part of the image that is valid. The rest of the image is filled with opaque white.

Note that if the image is progressive (as in a progressive JPEG or 2D interlaced PNG), this method may quickly return the full height of the image, but the image may still be loading, so do not use this return value as an indication of how much of the image remains to be decompressed.

If an error occurred while decompressing, this method returns `ImageRepLoadStatusInvalidData`. If *`complete`* is `true` but not enough data was available for decompression, `ImageRepLoadStatusUnexpectedEOF` is returned. If enough data has been provided (regardless of the *`complete`* flag), then `ImageRepLoadStatusCompleted` is returned. When any of these three status results are returned, this method has adjusted the NSBitmapImageRep so that `pixelsHigh` (page 790) and `size` (page 792), as well as the bitmap data, only contains the pixels that are valid, if any.

To cancel decompression, just pass in the existing data or `null` and `true` for *`complete`*. This method stops decompression immediately, adjusts the image size, and returns `ImageRepLoadStatusUnexpectedEOF`.This method returns `ImageRepLoadStatusCompleted` if you call it after receiving any error results (`ImageRepLoadStatusInvalidData` or `ImageRepLoadStatusUnexpectedEOF`) or if you call it on an NSBitmapImageRep that was not initialized with the constructor with the boolean parameter.

Availability
Available in Mac OS X v10.2 and later.

isPlanar

Returns `true` if image data is segregated into a separate plane for each color and coverage component (planar configuration) and `false` if the data is integrated into a single plane (meshed configuration).

```
public boolean isPlanar()
```

See Also
`samplesPerPixel` (page 197)

numberOfPlanes

Returns the number of separate planes image data is organized into.

```
public int numberOfPlanes()
```

Discussion
This number is the number of samples per pixel if the data has a separate plane for each component (`isPlanar` (page 196) returns `true`) and 1 if the data is meshed (`isPlanar` (page 196) returns `false`).

See Also
`samplesPerPixel` (page 197)
`hasAlpha` (page 789) (NSImageRep)
`bitsPerSample` (page 788) (NSImageRep)

representationUsingType

Returns a bitmap version of the receiver, using the format *`storageType`* and the properties *`properties`*.

```
public NSData representationUsingType(int storageType, NSDictionary properties)
```

Discussion
The contents of the *properties* dictionary is described in "Constants" (page 199).

See Also
`TIFFRepresentationOfImageReps` (page 191)
`TIFFRepresentation` (page 198)
`TIFFRepresentation` (page 768) (NSImage)

samplesPerPixel

Returns the number of components in the data.

```
public int samplesPerPixel()
```

Discussion
It includes both color components and the coverage component, if present.

See Also
`hasAlpha` (page 789) (NSImageRep)
`bitsPerSample` (page 788) (NSImageRep)

setBitmapData

Copies the data from *bits* to the image's bitmap.

```
public void setBitmapData(byte[] bits)
```

setBitmapDataPlanes

Copies the data from *planes* to a planar image's bitmap planes.

```
public void setBitmapDataPlanes(byte[][] planes)
```

setColor

```
public void setColor(NSColor color, int x, int y)
```

Discussion
Sets the receiver's pixel located at the coordinates *x*,*y* to *color*.

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

See Also
`color` (page 194)

setCompressionWithFactor

Sets the receiver's compression type and compression factor.

```
public void setCompressionWithFactor(int compression, float factor)
```

Discussion

compression identifies one of the supported compression types as described in "Constants" (page 199). *factor* is a value specific to the compression type; many types of compression don't support varying degrees of compression and thus ignore *factor*. JPEG compression allows a compression factor ranging from 0.0 to 1.0, with 0.0 being the lowest and 1.0 being the highest.

When an NSBitmapImageRep is created, the instance stores the compression type and factor for the source data. `TIFFRepresentation` (page 198) and `TIFFRepresentationOfImageReps` (page 191) (static method) try to use the stored compression type and factor. Use this method to change the compression type and factor.

See Also

`canBeCompressedUsingType` (page 193)

setPixel

Sets the receiver's pixel at the coordinate *x*,*y* to the raw pixel values in *pixelData*.

```
public void setPixel(long[] pixelData, int x, int y)
```

Discussion

The values must be in an orderappropriate to the receiver's `bitmapFormat` (page 192). Small pixel sample values should be passed as an integer value. Floating point values should be cast int[].

See Also

`getPixel` (page 195)

setProperty

Sets the image's *property* to *value*.

```
public void setProperty(String property, Object value)
```

Discussion

The properties can affect how the image is read in and saved to file. They're described in "Constants" (page 199). If *value* is `null`, the value at *property* is cleared.

TIFFRepresentation

```
public NSData TIFFRepresentation()
```

Returns a TIFF representation of the image, using the compression that's returned by `compressionFactor` (page 194) (if applicable).

```
public NSData TIFFRepresentation(int compression, float factor)
```

Discussion
This method uses the stored compression type and factor retrieved from the initial image data or changed using `setCompressionWithFactor` (page 198). If the stored compression type isn't supported for writing TIFF data (for example, `TIFFCompressionNEXT`), the stored compression is changed to `TIFFCompressionNone` before the TIFF representation is generated. JPEG compression in TIFF files is not supported, and *factor* is ignored.

If a problem is encountered during generation of the TIFF, throws a `TIFFException` or a `BadBitmapParametersException`.

See Also
`TIFFRepresentationOfImageReps` (page 191)
`representationUsingType` (page 196)
`TIFFRepresentation` (page 768) (NSImage)

valueForProperty

Returns the value for *property*.

```
public Object valueForProperty(String property)
```

Discussion
The properties can affect how the image is read in and saved to file. They're described in "Constants" (page 199).

Constants

These constants name properties that are used by `representationOfImageRepsInArray` (page 191), `representationUsingType` (page 196), `setPixel` (page 198), and `valueForProperty` (page 199):

Constant	Description
`ImageColorSyncProfileData`	The ColorSync profile. It can be used for TIFF, JPEG, GIF, and PNG files. The value is NSData. It's set when reading in and used when writing out image data..
`ImageCompressionFactor`	The compression factor. Used only for JPEG files. JPEG compression in TIFF files is not supported, and the factor is ignored. The value is a float between 0.0 and 1.0, with 0.0 being the lowest and 1.0 being the highest. It's set when reading in and used when writing out.
`ImageCompressionMethod`	The compression method. Used only for TIFF files. The value is NSTIFFCompression, described below. It's set when reading in and used when writing out.
`ImageDitherTransparency`	Whether the image is dithered. Used only for GIF files. The value is Boolean. It's used when writing out.
`ImageInterlaced`	Whether the image is interlaced. Used only for PNG files. The value is Boolean. It's used when writing out.

Constant	Description
ImageRGBColorTable	The RGB color table. Used only for GIF files. It's stored as packed RGB. It's set when reading in and used when writing out.
ImageEXIFData	The EXIF data for the image. Used only for JPEG files. The value is a NSDictionary containing the EXIF keys and values. It's set when reading in and used when writing out. Available in Mac OS X v10.4 and later.
ImageFrameCount	The number of frames in an animated GIF file. The value is an integer. It's used when reading in.
ImageGamma	The gamma value for the image. Used only for PNG files. Values are between 0.0 and 1.0, with 0.0 being black and 1.0 being the maximum color. The value is a float . It's set when reading in and used when writing out. Available in Mac OS X v10.4 and later.
ImageCurrentFrame	The current frame for an animated GIF file. The first frame is 0. The value is an integer.
ImageCurrentFrameDuration	The duration in seconds of the current frame for an animated GIF image. The value is a float. It is used when reading in, but not writing out.
ImageProgressive	Whether the image uses progressive encoding. Used only for JPEG files. The value is a Boolean. It's set when reading in and used when writing out. Available in Mac OS X v10.4 and later.
ImageLoopCount	The number of loops to make when animating a GIF image. Zero indicates loop indefinitely. The value must be an integer. It is used when reading in but not when writing out the image. Available in Mac OS X v10.3 and later.

The following file type constants are provided as a convenience by NSBitmapImageRep:

Constant	Description
BMPFileType	Windows bitmap image (BMP) format
GIFFileType	Graphics Image Format (GIF), originally created by CompuServe for online downloads
JPEGFileType	JPEG format
JPEG2000FileType	JPEG 2000 file format. Available in Mac OS X v10.4 and later.
PNGFileType	Portable Network Graphics (PNG) format
TIFFFileType	Tagged Image File Format (TIFF)

The following constants represent the various TIFF data-compression schemes supported by NSBitmapImageRep and are returned by TIFFCompressionTypes (page 191):

Constant	Description
TIFFCompressionNone	No compression.
TIFFCompressionCCITTFAX3	CCITT Fax Group 3 compression. It's for 1-bit fax images sent over telephone lines.
TIFFCompressionCCITTFAX4	CCITT Fax Group 4 compression. It's for 1-bit fax images sent over ISDN lines.
TIFFCompressionLZW	LZW compression.
TIFFCompressionJPEG	JPEG compression. No longer supported for input or output.
TIFFCompressionNEXT	NeXT compressed. Used for input only.
TIFFCompressionPackBits	PackBits compression.
TIFFCompressionOldJPEG	Old JPEG compression. No longer supported for input or output.

The following constants represent the various bitmap component formats supported by NSBitmapImageRep. These values are returned by bitmapFormat (page 192).

Constant	Description
AlphaFirstBitmapFormat	If 0, alpha values are the last component. For example, CMYKA and RGBA. Available in Mac OS X v10.4 and later.
AlphaNonpremultiplied-BitmapFormat	If 0, alpha values are premultiplied. Available in Mac OS X v10.4 and later.
FloatingPointSamplesBitmapFormat	If 0, samples are integer values. Available in Mac OS X v10.4 and later.

The following constants represent the various status values returned by incrementalLoadFromData (page 195):

Constant	Description
ImageRepLoadStatus-UnknownType	Not enough data to determine image format. You should continue to provide more data.
ImageRepLoadStatus-ReadingHeader	The image format is known, but not enough data has been read to determine the size, depth, etc., of the image. You should continue to provide more data.
ImageRepLoadStatus-WillNeedAllData	Incremental loading cannot be supported. Until you call incrementalLoadFromData (page 195) with true, this status will be returned. You can continue to call the method but no decompression will take place. Once you do call the method with true, then the image will be decompressed and one of the final three status messages will be returned.

Constant	Description
ImageRepLoadStatus-InvalidData	An error occurred during image decompression. The image contains the portions of the data that have already been successfully decompressed, if any
ImageRepLoadStatus-UnexpectedEOF	incrementalLoadFromData (page 195) was called with true, but not enough data was available for decompression. The image contains the portions of the data that have already been succesfully decompressed, if any.
ImageRepLoadStatus-Completed	Enough data has been provided to successfully decompress the image (regardless of the complete flag).

NSBox

Inherits from	NSView : NSResponder : NSObject
Implements	NSCoding (NSResponder)
Package:	com.apple.cocoa.application
Companion guide	Boxes

Overview

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 NSViews.

Subclassing Notes

An NSBox object is a view that draws a line around its rectangular bounds and that displays a title on or near the line (or might display neither line nor title). You can adjust the style of the line (bezel, grooved, or plain) as well as the placement and font of the title. An NSBox also has a content view to which other views can be added; it thus offers a way for an application to group related views. You could create a custom subclass of NSBox that alters or augments its appearance or that modifies its grouping behavior. For example, you might add color to the lines or background, add a new line style, or have the views in the group automatically snap to an invisible grid when added.

Methods to Override

You must override the `drawRect` (page 1753) method (inherited from NSView) if you want to customize the appearance of your NSBox objects. Depending on the visual effect you're trying to achieve, you may have to invoke `super`'s implementation first. For example, if you are compositing a small image in a corner of the box, you would invoke the superclass implementation first. If you're adding a new style of line, you would provide a way to store a request for this line type (such as a boolean instance variable and related accessor methods). Then, in `drawRect`, if a request for this line type exists, you would draw the entire view yourself (that is, without calling `super`). Otherwise, you would invoke the superclass implementation.

If you wish to change NSBox's grouping behavior or other behavioral characteristics, consider overriding `setContentView` (page 207), `sizeToFit` (page 210), or `addSubview` (page 1739) (inherited from NSView).

Special Considerations

If you are drawing the custom NSBox entirely by yourself, and you want it to look exactly like the superclass object (except for your changes), it may take some effort and time to get the details right.

Tasks

Constructors

`NSBox` (page 205)
Creates an NSBox with a zero-sized frame rectangle.

Getting and Modifying the Border and Title

`borderRect` (page 205)
Returns the rectangle in which the receiver's border is drawn.

`borderType` (page 205)
Returns the receiver's border type.

`setBorderType` (page 207)
Sets the border type to *aType*, which must be a valid border type.

`boxType` (page 206)
Returns the receiver's box type.

`setBoxType` (page 207)
Sets the box type to *boxType*, which must be a valid box type.

`setTitle` (page 208)
Sets the title to *aString*, and marks the region of the receiver within the title rectangle as needing display.

`setTitleFont` (page 209)
Sets *aFont* as the NSFont object used to draw the receiver's title and marks the region of the receiver within the title rectangle as needing display.

`setTitlePosition` (page 209)
Sets the title position to *aPosition*, which can be one of the values described in "Constants" (page 211).

`setTitleWithMnemonic` (page 209)
Sets the title of the receiver with a character denoted as an access key.

`title` (page 210)
Returns the receiver's title.

`titleCell` (page 210)
Returns the NSCell used to display the receiver's title.

`titleFont` (page 210)
Returns the NSFont used to draw the receiver's title.

`titlePosition` (page 211)
Returns a constant representing the title position.

`titleRect` (page 211)
Returns the rectangle in which the receiver's title is drawn.

Setting and Placing the Content View

`contentView` (page 206)
Returns the receiver's content view.

`contentViewMargins` (page 207)
Returns the distances between the border and the content view.

`setContentView` (page 207)
Sets the receiver's content view to *aView*, resizing the NSView to fit within the box's current content area.

`setContentViewMargins` (page 208)
Sets the horizontal and vertical distance between the border of the receiver and its content view.

Resizing the Box

`setFrameFromContentFrame` (page 208)
Places the receiver so its content view lies on *contentFrame*, reckoned in the coordinate system of the box's superview.

`sizeToFit` (page 210)
Resizes and moves the receiver's content view so it just encloses its subviews.

Constructors

NSBox

Creates an NSBox with a zero-sized frame rectangle.

```
public NSBox()
```

Creates an NSBox with *frameRect* as its frame rectangle.

```
public NSBox(NSRect frameRect)
```

Instance Methods

borderRect

Returns the rectangle in which the receiver's border is drawn.

```
public NSRect borderRect()
```

borderType

Returns the receiver's border type.

```
public int borderType()
```

Discussion
Currently, the following border types are defined:

```
NSView.NoBorder
NSView.LineBorder
NSView.BezelBorder
NSView.GrooveBorder
```

By default, an NSBox's border type is `NSView.GrooveBorder`.

See Also
`setBorderType` (page 207)

boxType

Returns the receiver's box type.

```
public int boxType()
```

Discussion
Currently, the following box types are defined:

```
BoxPrimary
BoxSecondary
BoxSeparator
BoxOldStyle
```

By default, an NSBox's box type is `BoxPrimary`.

See Also
`setBoxType` (page 207)

contentView

Returns the receiver's content view.

```
public NSView contentView()
```

Discussion
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` (page 207) method.

See Also
`setContentView` (page 207)

contentViewMargins

Returns the distances between the border and the content view.

```
public NSSize contentViewMargins()
```

Discussion
By default, both the width (the horizontal distance between the innermost edge of the border and the content view) and the height (the vertical distance between the innermost edge of the border and the content view) of the returned NSSize are 5.0 in the box's coordinate system.

See Also
`setContentViewMargins` (page 208)

setBorderType

Sets the border type to *aType*, which must be a valid border type.

```
public void setBorderType(int aType)
```

Discussion
Currently, the following border types are defined:

```
NSView.NoBorder
NSView.LineBorder
NSView.BezelBorder
NSView.GrooveBorder
```

If the size of the new border is different from that of the old border, the content view is resized to absorb the difference, and the box is marked for redisplay.

See Also
`borderType` (page 205)
`setNeedsDisplay` (page 1779) (NSView)

setBoxType

Sets the box type to *boxType*, which must be a valid box type.

```
public void setBoxType(int boxType)
```

See Also
`boxType` (page 206)

setContentView

Sets the receiver's content view to *aView*, resizing the NSView to fit within the box's current content area.

```
public void setContentView(NSView aView)
```

Discussion

The box is marked for redisplay.

See Also

`contentView` (page 206)

`setFrameFromContentFrame` (page 208)

`sizeToFit` (page 210)

`setNeedsDisplay` (page 1779) (NSView)

setContentViewMargins

Sets the horizontal and vertical distance between the border of the receiver and its content view.

```
public void setContentViewMargins(NSSize offsetSize)
```

Discussion

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` (page 210) message to the box after changing the size of its offsets. This message causes the content view to remain unchanged while the box is sized to fit around it.

See Also

`contentViewMargins` (page 207)

setFrameFromContentFrame

Places the receiver so its content view lies on *contentFrame*, reckoned in the coordinate system of the box's superview.

```
public void setFrameFromContentFrame(NSRect contentFrame)
```

Discussion

The box is marked for redisplay.

See Also

`setContentViewMargins` (page 208)

`setFrame` (page 1776) (NSView)

`setNeedsDisplay` (page 1779) (NSView)

setTitle

Sets the title to *aString*, and marks the region of the receiver within the title rectangle as needing display.

```
public void setTitle(String aString)
```

Discussion

By default, an NSBox's title is "Title." If the size of the new title is different from that of the old title, the content view is resized to absorb the difference.

See Also

`title` (page 210)

`titleRect` (page 211)

`setNeedsDisplay` (page 1779) (NSView)

setTitleFont

Sets *aFont* as the NSFont object used to draw the receiver's title and marks the region of the receiver within the title rectangle as needing display.

```
public void setTitleFont(NSFont aFont)
```

Discussion

The title is drawn using the 12.0-point system font by default. If the size of the new font is different from that of the old font, the content view is resized to absorb the difference.

See Also

`titleFont` (page 210)

`setNeedsDisplay` (page 1779) (NSView)

setTitlePosition

Sets the title position to *aPosition*, which can be one of the values described in "Constants" (page 211).

```
public void setTitlePosition(int aPosition)
```

Discussion

The default position is `AtTop`.

If the new title position changes the size of the box's border area, the content view is resized to absorb the difference, and the box is marked as needing redisplay.

See Also

`titlePosition` (page 211)

`setNeedsDisplay` (page 1779) (NSView)

setTitleWithMnemonic

Sets the title of the receiver with a character denoted as an access key.

```
public void setTitleWithMnemonic(String aString)
```

Discussion

Mnemonics are not supported in Mac OS X.

By default, an NSBox's title is "Title." The content view is not automatically resized, and the box is not marked for redisplay.

See Also

`setTitleWithMnemonic` (page 332) (NSCell)

sizeToFit

Resizes and moves the receiver's content view so it just encloses its subviews.

```
public void sizeToFit()
```

Discussion

The receiver is then moved and resized to wrap around the content view. The receiver's width is constrained so 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 margins around the content view

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

Returns the receiver's title.

```
public String title()
```

Discussion

By default, a box's title is "Title."

See Also

`setTitle` (page 208)

titleCell

Returns the NSCell used to display the receiver's title.

```
public NSCell titleCell()
```

titleFont

Returns the NSFont used to draw the receiver's title.

```
public NSFont titleFont()
```

Discussion

The title is drawn using the 12.0-point system font by default.

See Also

`setTitleFont` (page 209)

titlePosition

Returns a constant representing the title position.

```
public int titlePosition()
```

Discussion
See "Constants" (page 211) for a list of the title position constants.

See Also
`setTitlePosition` (page 209)

titleRect

Returns the rectangle in which the receiver's title is drawn.

```
public NSRect titleRect()
```

See Also
`setTitlePosition` (page 209)
`setTitle` (page 208)
`setTitleFont` (page 209)
`setFrameFromContentFrame` (page 208)
`sizeToFit` (page 210)

Constants

The following constants are provided by NSBox to specify the location of a box's title with respect to its border:

Constant	Description
`NoTitle`	The box has no title.
`AboveTop`	Title positioned above the box's top border.
`AtTop`	Title positioned within the box's top border.
`BelowTop`	Title positioned below the box's top border.
`AboveBottom`	Title positioned above the box's bottom border.
`AtBottom`	Title positioned within the box's bottom border.
`BelowBottom`	Title positioned below the box's bottom border.

NSBrowser

Inherits from	NSControl : NSView : NSResponder : NSObject
Implements	NSCoding (NSResponder)
Package:	com.apple.cocoa.application
Companion guide	Browsers

Overview

NSBrowser provides a user interface for displaying and selecting items from a list of data or from hierarchically organized lists of data such as directory paths. When working with a hierarchy of data, the levels are displayed in columns, which are numbered from left to right.

NSBrowser uses NSBrowserCell (page 245) to implement its user interface.

Tasks

Constructors

`NSBrowser` (page 220)
Creates an NSBrowser with a zero-sized frame rectangle.

Setting Component Classes

`cellClass` (page 220)
Always returns the NSBrowserCell class (even if the developer has sent a `setNewCellClass` (page 235) message to a particular instance).

`cellPrototype` (page 222)
Returns the receiver's prototype NSCell.

`setCellPrototype` (page 233)
Sets the NSCell instance copied to display items in the matrices in the columns of the receiver.

`matrixClass` (page 227)
Returns the class of NSMatrix used in the receiver's columns.

`setNewCellClass` (page 235)
Sets the class of NSCell used in the columns of the receiver to *`aClass`*.

`setNewMatrixClass` (page 235)
: Sets the matrix class (NSMatrix or an NSMatrix subclass) used in the receiver's columns to *aClass*.

Getting Matrices, Cells, and Rows

`selectedCell` (page 230)
: Returns the last (rightmost and lowest) selected NSCell.

`selectedCellInColumn` (page 230)
: Returns the last (lowest) NSCell selected in *column*.

`selectedCells` (page 231)
: Returns all cells selected in the rightmost column.

`selectAll` (page 230)
: Selects all NSCells in the last column of the receiver.

`selectedRowInColumn` (page 231)
: Returns the row index of the selected cell in the column specified by index *column*.

`selectRowInColumn` (page 231)
: Selects the cell at index *row* in the column identified by index *column*.

`loadedCellAtLocation` (page 226)
: Loads if necessary and returns the NSCell at *row* in *column*.

`matrixInColumn` (page 227)
: Returns the matrix located in the column identified by index *column*.

Getting and Setting Paths

`path` (page 227)
: Returns the receiver's current path.

`setPath` (page 235)
: Sets the path displayed by the receiver to *path*.

`pathToColumn` (page 228)
: Returns a string representing the path from the first column up to, but not including, the column at index *column*.

`pathSeparator` (page 228)
: Returns the path separator.

`setPathSeparator` (page 236)
: Sets the path separator to *newString*.

Manipulating Columns

`addColumn` (page 221)
: Adds a column to the right of the last column.

`displayAllColumns` (page 224)
: Updates the receiver to display all loaded columns.

`displayColumn` (page 224)
Updates the receiver to display the column with the index *column*.

`columnOfMatrix` (page 222)
Returns the column number in which *matrix* is located.

`selectedColumn` (page 231)
Returns the index of the last column with a selected item.

`lastColumn` (page 226)
Returns the index of the last column loaded.

`setLastColumn` (page 234)
Sets the last column to *column*.

`firstVisibleColumn` (page 225)
Returns the index of the first visible column.

`numberOfVisibleColumns` (page 227)
Returns the number of columns visible.

`lastVisibleColumn` (page 226)
Returns the index of the last visible column.

`validateVisibleColumns` (page 239)
Invokes delegate method `browserIsColumnValid` (page 241) for visible columns.

Loading Columns

`isLoaded` (page 225)
Returns whether column 0 is loaded.

`loadColumnZero` (page 226)
Loads column 0; unloads previously loaded columns.

`reloadColumn` (page 229)
Reloads *column* if it exists and sets it to be the last column.

Setting Selection Characteristics

`allowsBranchSelection` (page 221)
Returns whether the user can select branch items when multiple selection is enabled.

`setAllowsBranchSelection` (page 232)
Sets whether the user can select branch items when multiple selection is enabled, depending on the Boolean value passed in the *flag*.

`allowsEmptySelection` (page 221)
Returns whether there can be nothing selected.

`setAllowsEmptySelection` (page 232)
Sets whether there can be nothing selected, depending on the Boolean value passed in the *flag*.

`allowsMultipleSelection` (page 222)
Returns whether the user can select multiple items.

`setAllowsMultipleSelection` (page 233)
Sets whether the user can select multiple items, depending on the Boolean value passed in the *flag*.

Setting Column Characteristics

`reusesColumns` (page 229)
Returns `true` if NSMatrix objects aren't freed when their columns are unloaded.

`setReusesColumns` (page 236)
If *`flag`* is `true`, prevents NSMatrix objects from being freed when their columns are unloaded, so they can be reused.

`maxVisibleColumns` (page 227)
Returns the maximum number of visible columns.

`setMaxVisibleColumns` (page 235)
Sets the maximum number of columns displayed to *`columnCount`*.

`minColumnWidth` (page 227)
Returns the minimum column width in pixels.

`setMinColumnWidth` (page 235)
Sets the minimum column width to *`columnWidth`*, specified in pixels.

`separatesColumns` (page 232)
Returns whether columns are separated by bezeled borders.

`setSeparatesColumns` (page 237)
Sets whether to separate columns with bezeled borders, depending on the Boolean value *`flag`*.

`takesTitleFromPreviousColumn` (page 238)
Returns `true` if the title of a column is set to the string value of the selected NSCell in the previous column.

`setTakesTitleFromPreviousColumn` (page 237)
Sets whether the title of a column is set to the string value of the selected NSCell in the previous column, depending on the Boolean value *`flag`*.

Manipulating Column Titles

`titleOfColumn` (page 239)

`setTitleOfColumn` (page 237)
Sets the title of the column at index *`column`* to *`aString`*.

`isTitled` (page 226)
Returns whether columns display titles.

`setTitled` (page 237)
Sets whether columns display titles, depending on the Boolean value *`flag`*.

`drawTitleOfColumn` (page 225)
Draws the title for the column at index *`column`* within the rectangle defined by *`aRect`*.

`titleHeight` (page 239)
Returns the height of column titles.

`titleFrameOfColumn` (page 238)
Returns the bounds of the title frame for the column at index *`column`*.

Scrolling an NSBrowser

`scrollColumnToVisible` (page 229)
Scrolls to make the column at index *column* visible.

`scrollColumnsLeftBy` (page 229)
Scrolls columns left by *shiftAmount* columns.

`scrollColumnsRightBy` (page 229)
Scrolls columns right by *shiftAmount* columns.

`updateScroller` (page 239)
Updates the horizontal scroller to reflect column positions.

`scrollViaScroller` (page 230)
Scrolls columns left or right based on an NSScroller.

Showing a Horizontal Scroller

`hasHorizontalScroller` (page 225)
Returns whether an NSScroller is used to scroll horizontally.

`setHasHorizontalScroller` (page 234)
Sets whether an NSScroller is used to scroll horizontally.

Setting the Behavior of Arrow Keys

`acceptsArrowKeys` (page 221)

`setAcceptsArrowKeys` (page 232)
Enables or disables the arrow keys as used for navigating within and between browsers, depending on the Boolean value passed in the *flag*.

`sendsActionOnArrowKeys` (page 232)

`setSendsActionOnArrowKeys` (page 237)
Sets whether pressing an arrow key will cause the action message to be sent (in addition to causing scrolling), depending on the Boolean value *flag*.

Getting Column Frames

`frameOfColumn` (page 225)
Returns the rectangle containing the column at index *column*.

`frameOfInsideOfColumn` (page 225)
Returns the rectangle containing the column at index *column*, not including borders.

Arranging Browser Components

`tile` (page 238)
Adjusts the various subviews of the receiver—scrollers, columns, titles, and so on—without redrawing.

Setting the Delegate

`delegate` (page 223)
Returns the receiver's delegate.

`setDelegate` (page 234)
Sets the receiver's delegate to *anObject*.

Target and Action

`doubleAction` (page 224)
Returns the receiver's double-click action method.

`setDoubleAction` (page 234)
Sets the receiver's double-click action to *aSelector*.

`sendAction` (page 231)
Sends the action message to the target.

Event Handling

`doClick` (page 224)
Responds to (single) mouse clicks in a column of the receiver.

`doDoubleClick` (page 224)
Responds to double clicks in a column of the receiver.

Resizing Columns

`columnContentWidthForColumnWidth` (page 222)
Given the column width (the entire scrolling text view), returns the content width (the matrix in the column).

`columnWidthForColumnContentWidth` (page 223)
Given the content width (the matrix in the column), returns the column width (the entire scrolling text view).

`setColumnResizingType` (page 233)
Sets the receiver's column resizing type.

`columnResizingType` (page 222)
Returns the receiver's column resizing type.

`setPrefersAllColumnUserResizing:` (page 236)
Specifies whether the browser resizes all columns simultaneously rather than resizing a single column at a time.

`prefersAllColumnUserResizing` (page 228)
: Returns `true` if the browser is set to resize all columns simultaneously rather than resizing a single column at a time.

`setWidthOfColumn` (page 238)
: Sets the width of the specified column.

`widthOfColumn` (page 239)
: Returns the width of the specified column.

`setColumnsAutosaveName` (page 233)
: Sets the name used to automatically save the receiver's column configuration.

`columnsAutosaveName` (page 223)
: Returns the name used to automatically save the receiver's column configuration.

`removeSavedColumnsWithAutosaveName` (page 221)
: Removes the column configuration data stored under *`name`* from the application's user defaults.

Creating rows

`browserCreateRowsForColumn` (page 240) *delegate method*
: Creates a row in *`matrix`* for each row of data to be displayed in *`column`* of the browser.

Displaying a cell

`browserWillDisplayCell` (page 243) *delegate method*
: This method gives the delegate the opportunity to modify the specified *`cell`* at *`row`* in *`column`* before it's displayed by the NSBrowser.

Getting information about a browser

`browserIsColumnValid` (page 241) *delegate method*
: Returns whether the contents of the column, specified by *`column`*, are valid.

`browserNumberOfRowsInColumn` (page 241) *delegate method*
: Returns the number of rows of data in the column at index *`column`*.

`browserTitleOfColumn` (page 243) *delegate method*
: Asks the delegate for the title to display above the column at index *`column`*.

Selecting

`browserSelectCellWithStringInColumn` (page 241) *delegate method*
: Asks the delegate to select the NSCell with title *`title`* in the column at index *`column`*.

`browserSelectRowInColumn` (page 241) *delegate method*
: Asks the delegate to select the NSCell at row *`row`* in the column at index *`column`*.

Scrolling

`browserDidScroll` (page 241) *delegate method*
Notifies the delegate when the NSBrowser has scrolled.

`browserWillScroll` (page 243) *delegate method*
Notifies the delegate when the NSBrowser will scroll.

Resizing columns

`browserShouldSizeColumnToWidth` (page 242) *delegate method*
Used for determining a column's initial size.

`browserSizeToFitWidthOfColumn` (page 242) *delegate method*
Returns the ideal width for a column

`browserColumnConfigurationDidChange` (page 240) *delegate method*
Used by clients to implement their own column width persistence.

Constructors

NSBrowser

Creates an NSBrowser with a zero-sized frame rectangle.

```
public NSBrowser()
```

Creates an NSBrowser in *frameRect*.

```
public NSBrowser(NSRect frameRect)
```

Static Methods

cellClass

Always returns the NSBrowserCell class (even if the developer has sent a `setNewCellClass` (page 235) message to a particular instance).

```
public static Class cellClass()
```

Discussion
This method is used by NSControl during initialization and is not meant to be used by applications.

See Also
`cellPrototype` (page 222)
`setCellPrototype` (page 233)

removeSavedColumnsWithAutosaveName

Removes the column configuration data stored under *name* from the application's user defaults.

```
public static void removeSavedColumnsWithAutosaveName(String name)
```

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

See Also
`columnsAutosaveName` (page 223)
`setColumnsAutosaveName` (page 233)

Instance Methods

acceptsArrowKeys

```
public boolean acceptsArrowKeys()
```

Discussion
Returns `true` if the arrow keys are enabled.

See Also
`setAcceptsArrowKeys` (page 232)

addColumn

Adds a column to the right of the last column.

```
public void addColumn()
```

See Also
`columnOfMatrix` (page 222)
`displayColumn` (page 224)
`selectedColumn` (page 231)

allowsBranchSelection

Returns whether the user can select branch items when multiple selection is enabled.

```
public boolean allowsBranchSelection()
```

See Also
`setAllowsBranchSelection` (page 232)

allowsEmptySelection

Returns whether there can be nothing selected.

```
public boolean allowsEmptySelection()
```

See Also
`setAllowsEmptySelection` (page 232)

allowsMultipleSelection

Returns whether the user can select multiple items.

```
public boolean allowsMultipleSelection()
```

See Also
`setAllowsMultipleSelection` (page 233)

cellPrototype

Returns the receiver's prototype NSCell.

```
public NSCell cellPrototype()
```

See Also
`setCellPrototype` (page 233)
`setNewCellClass` (page 235)

columnContentWidthForColumnWidth

Given the column width (the entire scrolling text view), returns the content width (the matrix in the column).

```
public float columnContentWidthForColumnWidth(float columnWidth)
```

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

See Also
`columnWidthForColumnContentWidth` (page 223)

columnOfMatrix

Returns the column number in which *`matrix`* is located.

```
public int columnOfMatrix(NSMatrix matrix)
```

See Also
`matrixInColumn` (page 227)

columnResizingType

Returns the receiver's column resizing type.

```
public int columnResizingType()
```

Discussion
Possible return values are described in "Constants" (page 240). The default is `AutoColumnResizing`.

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

See Also
`setColumnResizingType` (page 233)

columnsAutosaveName

Returns the name used to automatically save the receiver's column configuration.

```
public String columnsAutosaveName()
```

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

See Also
`setColumnsAutosaveName` (page 233)

columnWidthForColumnContentWidth

Given the content width (the matrix in the column), returns the column width (the entire scrolling text view).

```
public float columnWidthForColumnContentWidth(float columnContentWidth)
```

Discussion
For example, to guarantee that 16 pixels of your browser cell are always visible, call:

```
browser.setMinColumnWidth(browser.columnWidthForColumnContentWidth(16))
```

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

See Also
`columnContentWidthForColumnWidth` (page 222)

delegate

Returns the receiver's delegate.

```
public Object delegate()
```

See Also
`setDelegate` (page 234)

displayAllColumns

Updates the receiver to display all loaded columns.

```
public void displayAllColumns()
```

See Also

`addColumn` (page 221)

`validateVisibleColumns` (page 239)

displayColumn

Updates the receiver to display the column with the index *column*.

```
public void displayColumn(int column)
```

See Also

`addColumn` (page 221)

`validateVisibleColumns` (page 239)

doClick

Responds to (single) mouse clicks in a column of the receiver.

```
public void doClick(Object sender)
```

See Also

`sendAction` (page 231)

doDoubleClick

Responds to double clicks in a column of the receiver.

```
public void doDoubleClick(Object sender)
```

See Also

`setDoubleAction` (page 234)

doubleAction

Returns the receiver's double-click action method.

```
public NSSelector doubleAction()
```

See Also

`setDoubleAction` (page 234)

drawTitleOfColumn

Draws the title for the column at index *column* within the rectangle defined by *aRect*.

```
public void drawTitleOfColumn(int column, NSRect aRect)
```

See Also

`setTitleOfColumn` (page 237)

`titleFrameOfColumn` (page 238)

`titleHeight` (page 239)

firstVisibleColumn

Returns the index of the first visible column.

```
public int firstVisibleColumn()
```

See Also

`lastVisibleColumn` (page 226)

`numberOfVisibleColumns` (page 227)

frameOfColumn

Returns the rectangle containing the column at index *column*.

```
public NSRect frameOfColumn(int column)
```

frameOfInsideOfColumn

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

```
public NSRect frameOfInsideOfColumn(int column)
```

hasHorizontalScroller

Returns whether an NSScroller is used to scroll horizontally.

```
public boolean hasHorizontalScroller()
```

See Also

`setHasHorizontalScroller` (page 234)

isLoaded

Returns whether column 0 is loaded.

```
public boolean isLoaded()
```

See Also

`loadColumnZero` (page 226)

`reloadColumn` (page 229)

isTitled

Returns whether columns display titles.

```
public boolean isTitled()
```

See Also

`setTitled` (page 237)

lastColumn

Returns the index of the last column loaded.

```
public int lastColumn()
```

See Also

`selectedColumn` (page 231)

`setLastColumn` (page 234)

lastVisibleColumn

Returns the index of the last visible column.

```
public int lastVisibleColumn()
```

See Also

`firstVisibleColumn` (page 225)

`numberOfVisibleColumns` (page 227)

loadColumnZero

Loads column 0; unloads previously loaded columns.

```
public void loadColumnZero()
```

See Also

`isLoaded` (page 225)

`reloadColumn` (page 229)

loadedCellAtLocation

Loads if necessary and returns the NSCell at *row* in *column*.

```
public NSCell loadedCellAtLocation(int row, int column)
```

See Also
`selectedCellInColumn` (page 230)

matrixClass

Returns the class of NSMatrix used in the receiver's columns.

```
public Class matrixClass()
```

See Also
`setNewMatrixClass` (page 235)

matrixInColumn

Returns the matrix located in the column identified by index *column*.

```
public NSMatrix matrixInColumn(int column)
```

maxVisibleColumns

Returns the maximum number of visible columns.

```
public int maxVisibleColumns()
```

See Also
`setMaxVisibleColumns` (page 235)

minColumnWidth

Returns the minimum column width in pixels.

```
public float minColumnWidth()
```

See Also
`setMinColumnWidth` (page 235)

numberOfVisibleColumns

Returns the number of columns visible.

```
public int numberOfVisibleColumns()
```

See Also
`validateVisibleColumns` (page 239)

path

Returns the receiver's current path.

```
public String path()
```

Discussion
The components are separated with the string returned by `pathSeparator` (page 228).

Invoking this method is equivalent to invoking `pathToColumn` (page 228) for all columns.

See Also
`setPath` (page 235)

pathSeparator

Returns the path separator.

```
public String pathSeparator()
```

Discussion
The default is "/".

See Also
`setPathSeparator` (page 236)

pathToColumn

Returns a string representing the path from the first column up to, but not including, the column at index *column*.

```
public String pathToColumn(int column)
```

Discussion
The components are separated with the string returned by `pathSeparator` (page 228).

See Also
`path` (page 227)
`setPath` (page 235)

prefersAllColumnUserResizing

Returns `true` if the browser is set to resize all columns simultaneously rather than resizing a single column at a time.

```
public boolean prefersAllColumnUserResizing()
```

Discussion
The default is `false`. This setting applies only to browsers that allow the user to resize columns (see `UserColumnResizing` (page 240). Holding down the Option key while resizing switches the type of resizing used.

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

See Also

setPrefersAllColumnUserResizing: (page 236)

setColumnResizingType (page 233)

reloadColumn

Reloads *column* if it exists and sets it to be the last column.

```
public void reloadColumn(int column)
```

See Also

isLoaded (page 225)

loadColumnZero (page 226)

reusesColumns

Returns `true` if NSMatrix objects aren't freed when their columns are unloaded.

```
public boolean reusesColumns()
```

See Also

setReusesColumns (page 236)

scrollColumnsLeftBy

Scrolls columns left by *shiftAmount* columns.

```
public void scrollColumnsLeftBy(int shiftAmount)
```

See Also

scrollViaScroller (page 230)

updateScroller (page 239)

scrollColumnsRightBy

Scrolls columns right by *shiftAmount* columns.

```
public void scrollColumnsRightBy(int shiftAmount)
```

See Also

scrollViaScroller (page 230)

updateScroller (page 239)

scrollColumnToVisible

Scrolls to make the column at index *column* visible.

```
public void scrollColumnToVisible(int column)
```

See Also

`scrollViaScroller` (page 230)

`updateScroller` (page 239)

scrollViaScroller

Scrolls columns left or right based on an NSScroller.

```
public void scrollViaScroller(NSScroller sender)
```

See Also

`updateScroller` (page 239)

selectAll

Selects all NSCells in the last column of the receiver.

```
public void selectAll(Object sender)
```

See Also

`selectedCell` (page 230)

`selectedCells` (page 231)

`selectedColumn` (page 231)

selectedCell

Returns the last (rightmost and lowest) selected NSCell.

```
public NSCell selectedCell()
```

See Also

`loadedCellAtLocation` (page 226)

`selectedCell` (page 230)

`selectRowInColumn` (page 231)

selectedCellInColumn

Returns the last (lowest) NSCell selected in *column*.

```
public NSCell selectedCellInColumn(int column)
```

See Also

`loadedCellAtLocation` (page 226)

`selectedCell` (page 230)

`selectedRowInColumn` (page 231)

selectedCells

Returns all cells selected in the rightmost column.

```
public NSArray selectedCells()
```

See Also

`selectAll` (page 230)

`selectedCell` (page 230)

selectedColumn

Returns the index of the last column with a selected item.

```
public int selectedColumn()
```

See Also

`columnOfMatrix` (page 222)

`selectAll` (page 230)

selectedRowInColumn

Returns the row index of the selected cell in the column specified by index *column*.

```
public int selectedRowInColumn(int column)
```

See Also

`loadedCellAtLocation` (page 226)

`selectedCell` (page 230)

`selectedCellInColumn` (page 230)

selectRowInColumn

Selects the cell at index *row* in the column identified by index *column*.

```
public void selectRowInColumn(int row, int column)
```

See Also

`loadedCellAtLocation` (page 226)

sendAction

Sends the action message to the target.

```
public boolean sendAction()
```

Discussion

Returns `true` upon success, `false` if no target for the message could be found.

sendsActionOnArrowKeys

```
public boolean sendsActionOnArrowKeys()
```

Discussion

Returns `false` if pressing an arrow key only scrolls the receiver, `true` if it also sends the action message specified by `setAction` (page 455).

See Also

`acceptsArrowKeys` (page 221)
`setSendsActionOnArrowKeys` (page 237)

separatesColumns

Returns whether columns are separated by bezeled borders.

```
public boolean separatesColumns()
```

See Also

`setSeparatesColumns` (page 237)

setAcceptsArrowKeys

Enables or disables the arrow keys as used for navigating within and between browsers, depending on the Boolean value passed in the *`flag`*.

```
public void setAcceptsArrowKeys(boolean flag)
```

See Also

`acceptsArrowKeys` (page 221)
`sendsActionOnArrowKeys` (page 232)

setAllowsBranchSelection

Sets whether the user can select branch items when multiple selection is enabled, depending on the Boolean value passed in the *`flag`*.

```
public void setAllowsBranchSelection(boolean flag)
```

See Also

`allowsBranchSelection` (page 221)

setAllowsEmptySelection

Sets whether there can be nothing selected, depending on the Boolean value passed in the *`flag`*.

```
public void setAllowsEmptySelection(boolean flag)
```

See Also

`allowsEmptySelection` (page 221)

setAllowsMultipleSelection

Sets whether the user can select multiple items, depending on the Boolean value passed in the *flag*.

```
public void setAllowsMultipleSelection(boolean flag)
```

See Also
`allowsMultipleSelection` (page 222)

setCellPrototype

Sets the NSCell instance copied to display items in the matrices in the columns of the receiver.

```
public void setCellPrototype(NSCell aCell)
```

See Also
`cellClass` (page 220)
`cellPrototype` (page 222)

setColumnResizingType

Sets the receiver's column resizing type.

```
public void setColumnResizingType(int columnResizingType)
```

Discussion
Possible values for *columnResizingType* are described in "Constants" (page 240). The default is `AutoColumnResizing`. This setting is persistent.

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

See Also
`columnResizingType` (page 222)

setColumnsAutosaveName

Sets the name used to automatically save the receiver's column configuration.

```
public void setColumnsAutosaveName(String name)
```

Discussion
If *name* is different from the current name, this method also reads in any column configuration data previously saved under *name* and applies the values to the browser. Column configuration is defined as an array of column content widths. One width is saved for each level the user has reached. That is, the browser saves column width based on depth, not on unique paths. To do more complex column persistence, you should register for `ColumnConfigurationDidChangeNotification` (page 243) and handle persistence yourself. This setting is persistent.

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

See Also

columnsAutosaveName (page 223)

setDelegate

Sets the receiver's delegate to *anObject*.

```
public void setDelegate(Object anObject)
```

Discussion

Throws BrowserIllegalDelegateException if the delegate specified by *anObject* doesn't respond to browserWillDisplayCell (page 243) and either of the methods browserNumberOfRowsInColumn (page 241) or browserCreateRowsForColumn (page 240).

See Also

delegate (page 223)

setDoubleAction

Sets the receiver's double-click action to *aSelector*.

```
public void setDoubleAction(NSSelector aSelector)
```

Discussion

For the method to have any effect, the receiver's action and target must be set to the class in which the selector is declared. See *Action Messages* for additional information on action messages.

See Also

doubleAction (page 224)

sendAction (page 231)

setHasHorizontalScroller

Sets whether an NSScroller is used to scroll horizontally.

```
public void setHasHorizontalScroller(boolean flag)
```

See Also

hasHorizontalScroller (page 225)

setLastColumn

Sets the last column to *column*.

```
public void setLastColumn(int column)
```

See Also

lastColumn (page 226)

lastVisibleColumn (page 226)

setMaxVisibleColumns

Sets the maximum number of columns displayed to *columnCount*.

```
public void setMaxVisibleColumns(int columnCount)
```

See Also

`maxVisibleColumns` (page 227)

setMinColumnWidth

Sets the minimum column width to *columnWidth*, specified in pixels.

```
public void setMinColumnWidth(float columnWidth)
```

See Also

`minColumnWidth` (page 227)

setNewCellClass

Sets the class of NSCell used in the columns of the receiver to *aClass*.

```
public void setNewCellClass(Class aClass)
```

See Also

`cellClass` (page 220)

`cellPrototype` (page 222)

setNewMatrixClass

Sets the matrix class (NSMatrix or an NSMatrix subclass) used in the receiver's columns to *aClass*.

```
public void setNewMatrixClass(Class aClass)
```

See Also

`matrixClass` (page 227)

setPath

Sets the path displayed by the receiver to *path*.

```
public boolean setPath(String path)
```

Discussion

If *path* is prefixed by the path separator, the path is absolute, containing the full path from the receiver's first column. Otherwise, the path is relative, extending the receiver's current path starting at the last column. Returns `true` if *path* is valid.

While parsing *path*, the receiver compares each component with the entries in the current column. If an exact match is found, the matching entry is selected, and the next component is compared to the next column's entries. If no match is found for a component, the method exits and returns `false`; the final path is set to the valid portion of *path*. If each component of *path* specifies a valid branch or leaf in the receiver's hierarchy, the method returns `true`.

See Also

`path` (page 227)

`pathToColumn` (page 228)

`pathSeparator` (page 228)

`setPathSeparator` (page 236)

setPathSeparator

Sets the path separator to *newString*.

```
public void setPathSeparator(String newString)
```

See Also

`pathSeparator` (page 228)

setPrefersAllColumnUserResizing:

Specifies whether the browser resizes all columns simultaneously rather than resizing a single column at a time.

```
public void setPrefersAllColumnUserResizing(boolean prefersAllColumnResizing)
```

Discussion

Set to `true`, to cause the browser to resize all columns simultaneously; the default is single column resizing (`false`). This setting applies only to browsers that allow the user to resize columns (see `UserColumnResizing` (page 240). Holding down the Option key while resizing switches the type of resizing used. This setting is persistent.

Availability

Available in Mac OS X v10.3 and later.

See Also

`prefersAllColumnUserResizing` (page 228)

`setColumnResizingType` (page 233)

setReusesColumns

If *flag* is `true`, prevents NSMatrix objects from being freed when their columns are unloaded, so they can be reused.

```
public void setReusesColumns(boolean flag)
```

See Also

`reusesColumns` (page 229)

setSendsActionOnArrowKeys

Sets whether pressing an arrow key will cause the action message to be sent (in addition to causing scrolling), depending on the Boolean value *flag*.

```
public void setSendsActionOnArrowKeys(boolean flag)
```

See Also

sendsActionOnArrowKeys (page 232)

setSeparatesColumns

Sets whether to separate columns with bezeled borders, depending on the Boolean value *flag*.

```
public void setSeparatesColumns(boolean flag)
```

Discussion

This value is ignored if isTitled (page 226) does not return false.

See Also

separatesColumns (page 232)

setTakesTitleFromPreviousColumn

Sets whether the title of a column is set to the string value of the selected NSCell in the previous column, depending on the Boolean value *flag*.

```
public void setTakesTitleFromPreviousColumn(boolean flag)
```

See Also

takesTitleFromPreviousColumn (page 238)

setTitled

Sets whether columns display titles, depending on the Boolean value *flag*.

```
public void setTitled(boolean flag)
```

See Also

isTitled (page 226)

setTitleOfColumn

Sets the title of the column at index *column* to *aString*.

```
public void setTitleOfColumn(String aString, int column)
```

See Also

drawTitleOfColumn (page 225)

titleOfColumn (page 239)

setWidthOfColumn

Sets the width of the specified column.

```
public void setWidthOfColumn(float columnWidth, int columnIndex)
```

Discussion

This method can be used to set the initial width of browser columns unless the column sizing is automatic; `setWidthOfColumn` does nothing if `columnResizingType` (page 222) is `AutoColumnResizing`. To set the default width for new columns (that don't otherwise have initial widths from defaults or via the delegate), use a *`columnIndex`* of –1. A value set for *`columnIndex`* of –1 is persistent. An `ColumnConfigurationDidChangeNotification` (page 243) notification is posted (not immediately), if necessary, so that the receiver can autosave the new column configuration.

Availability

Available in Mac OS X v10.3 and later.

See Also

`widthOfColumn` (page 239)

`browserShouldSizeColumnToWidth` (page 242)

takesTitleFromPreviousColumn

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

```
public boolean takesTitleFromPreviousColumn()
```

See Also

`setTakesTitleFromPreviousColumn` (page 237)

tile

Adjusts the various subviews of the receiver—scrollers, columns, titles, and so on—without redrawing.

```
public void tile()
```

Discussion

Your code shouldn't send this message. It's invoked any time the appearance of the receiver changes.

titleFrameOfColumn

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

```
public NSRect titleFrameOfColumn(int column)
```

See Also

`drawTitleOfColumn` (page 225)

titleHeight

Returns the height of column titles.

```
public float titleHeight()
```

See Also
drawTitleOfColumn (page 225)

titleOfColumn

```
public String titleOfColumn(int column)
```

Discussion
Returns the title displayed for the column at index *column*.

See Also
setTitleOfColumn (page 237)

updateScroller

Updates the horizontal scroller to reflect column positions.

```
public void updateScroller()
```

See Also
scrollViaScroller (page 230)

validateVisibleColumns

Invokes delegate method browserIsColumnValid (page 241) for visible columns.

```
public void validateVisibleColumns()
```

See Also
numberOfVisibleColumns (page 227)

widthOfColumn

Returns the width of the specified column.

```
public float widthOfColumn(int)
```

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

See Also
setWidthOfColumn (page 238)

Constants

The following constants are defined by NSBrowser to describe types of browser column resizing, and are used by setColumnResizingType (page 233) and columnResizingType (page 222):

Constant	Description
NoColumnResizing	Neither NSBrowser nor the user can change the column width. The developer must explicitly set all column widths.
AutoColumnResizing	All columns have the same width, calculated using a combination of the minimum column width and maximum number of visible columns settings. The column width changes as the window size changes. The user cannot resize columns.
UserColumnResizing	The developer chooses the initial column widths, but users can resize all columns simultaneously or each column individually.

Delegate Methods

browserColumnConfigurationDidChange

Used by clients to implement their own column width persistence.

```
public abstract void browserColumnConfigurationDidChange(NSNotification notification)
```

Discussion
Implementation is optional, and used for browsers with resize type UserColumnResizing only. It is called when the method setWidthOfColumn (page 238) is used to change the width of any browser columns or when the user resizes any columns. If the user resizes more than one column, a single notification is posted when the user is finished resizing.

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

See Also
ColumnConfigurationDidChangeNotification (page 243)
setWidthOfColumn (page 238)

browserCreateRowsForColumn

Creates a row in *matrix* for each row of data to be displayed in *column* of the browser.

```
public abstract void browserCreateRowsForColumn(NSBrowser sender, int column,
    NSMatrix matrix)
```

Discussion
Either this method or browserNumberOfRowsInColumn (page 241) must be implemented, but not both (or a BrowserIllegalDelegateException will be thrown).

See Also

`browserWillDisplayCell` (page 243)

browserDidScroll

Notifies the delegate when the NSBrowser has scrolled.

```
public abstract void browserDidScroll(NSBrowser sender)
```

browserIsColumnValid

Returns whether the contents of the column, specified by *column*, are valid.

```
public abstract boolean browserIsColumnValid(NSBrowser sender, int column)
```

Discussion

If `false` is returned, *sender* reloads the column. This method is invoked in response to `validateVisibleColumns` (page 239) being sent to *sender*.

browserNumberOfRowsInColumn

Returns the number of rows of data in the column at index *column*.

```
public abstract int browserNumberOfRowsInColumn(NSBrowser sender, int column)
```

Discussion

Either this method or `browserCreateRowsForColumn` (page 240) must be implemented, but not both.

See Also

`browserWillDisplayCell` (page 243)

browserSelectCellWithStringInColumn

Asks the delegate to select the NSCell with title *title* in the column at index *column*.

```
public abstract boolean browserSelectCellWithStringInColumn(NSBrowser sender, String
     title, int column)
```

Discussion

It is the delegate's responsibility to select the cell, rather than the browser. If the delegate returns `false`, the NSCell was not selected; otherwise `true` is returned. Invoked in response to `setPath` (page 235) being received by *sender*.

See Also

`selectedCellInColumn` (page 230)

browserSelectRowInColumn

Asks the delegate to select the NSCell at row *row* in the column at index *column*.

```
public abstract boolean browserSelectRowInColumn(NSBrowser sender, int row, int
    column)
```

Discussion
It is the delegate's responsibility to select the cell, rather than the browser. If the delegate returns `false`, the NSCell was not selected; otherwise `true` is returned. Invoked in response to `selectRowInColumn` (page 231) being received by *sender*.

See Also
`selectedRowInColumn` (page 231)
`selectRowInColumn` (page 231)

browserShouldSizeColumnToWidth

Used for determining a column's initial size.

```
public abstract float browserShouldSizeColumnToWidth(NSBrowser browser, int
    columnIndex, boolean forUserResize, float suggestedWidth)
```

Discussion
Implementation is optional and applies only to browsers with resize type `NoColumnResizing` or `UserColumnResizing` (see "Constants" (page 240)). When this method is called, it includes a suggested width for the column. This method should return your desired initial width for a newly added column. If you want to accept the suggested width, return *suggestedWidth*. If you return 0 or a size too small to display the resize handle and a portion of the column, the actual size used will be larger than you requested.

As currently implemented, this method is always called with *forUserResize* set to `false`.

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

See Also
`setWidthOfColumn` (page 238)

browserSizeToFitWidthOfColumn

Returns the ideal width for a column

```
public abstract float browserSizeToFitWidthOfColumn(NSBrowser browser, int
    columnIndex)
```

Discussion
. Implementation is optional and is for browsers with resize type `UserColumnResizing` only. This method is used when performing a "right-size" operation; that is, when sizing a column to the smallest width that contains all the content without clipping or truncating. If *columnIndex* is –1, the result is used for a "right-size-all" operation. In that case, you should return a size that can be uniformly applied to all columns (that is, every column will be set to this size). It is assumed that the implementation may be expensive, so it will be called only when necessary.

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

browserTitleOfColumn

Asks the delegate for the title to display above the column at index *column*.

```
public abstract String browserTitleOfColumn(NSBrowser sender, int column)
```

See Also

`setTitleOfColumn` (page 237)

`titleOfColumn` (page 239)

browserWillDisplayCell

This method gives the delegate the opportunity to modify the specified *cell* at *row* in *column* before it's displayed by the NSBrowser.

```
public abstract void browserWillDisplayCell(NSBrowser sender, Object cell, int row,
      int column)
```

Discussion

The delegate should set any state necessary for the correct display of the cell.

See Also

`browserCreateRowsForColumn` (page 240)

`browserNumberOfRowsInColumn` (page 241)

browserWillScroll

Notifies the delegate when the NSBrowser will scroll.

```
public abstract void browserWillScroll(NSBrowser sender)
```

Notifications

ColumnConfigurationDidChangeNotification

Notifies the delegate when the width of a browser column has changed. The notification object is the browser whose column sizes need to be made persistent. This notification does not contain a `userInfo` dictionary. If the user resizes more than one column, a single notification is posted when the user is finished resizing.

Availability

Available in Mac OS X v10.3 and later.

See Also

`browserColumnConfigurationDidChange` (page 240)

NSBrowserCell

Inherits from	NSCell : NSObject
Implements	NSCoding (NSCell)
Package:	com.apple.cocoa.application
Companion guide	Browsers

Overview

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.)

NSBrowserCell implements the user interface of NSBrowser (page 213).

Tasks

Constructors

`NSBrowserCell` (page 246)
Creates an empty NSBrowserCell.

Accessing Graphics Images

`branchImage` (page 247)
Returns the default image for branch NSBrowserCells (a right-pointing triangle).

`highlightedBranchImage` (page 247)
Returns the default NSImage for branch NSBrowserCells that are highlighted (a lighter version of the image returned by `branchImage` (page 247)).

`alternateImage` (page 247)
Returns this receiver's image for the highlighted state or `null` if no image is set.

`setAlternateImage` (page 248)
Sets the receiver's image for the highlighted state.

Setting State

`reset` (page 248)
Unhighlights the receiver and unsets its state.

`set` (page 248)
Highlights the receiver and sets its state.

Determining Cell Attributes

`isLeaf` (page 248)
Returns whether the receiver is a leaf or a branch cell.

`setLeaf` (page 249)
Sets whether the receiver is a leaf or a branch cell, depending on the Boolean value *`flag`*.

`isLoaded` (page 248)
Returns `true` if the receiver's state has been set and the cell is ready to display.

`setLoaded` (page 249)
Sets whether the receiver's state has been set and the cell is ready to display, depending on the Boolean value *`flag`*.

`highlightColorInView` (page 247)
Returns the highlight color that the receiver wants to display as in *`controlView`*.

Constructors

NSBrowserCell

Creates an empty NSBrowserCell.

```
public NSBrowserCell()
```

Creates an NSBrowserCell initialized with *`aString`* and set to have the cell's default menu.

```
public NSBrowserCell(String aString)
```

Creates an NSBrowserCell initialized with *`anImage`* and set to have the cell's default menu.

```
public NSBrowserCell(NSImage anImage)
```

Discussion
If *`anImage`* is `null`, no image is set.

Static Methods

branchImage

Returns the default image for branch NSBrowserCells (a right-pointing triangle).

```
public static NSImage branchImage()
```

Discussion

Override this method if you want a different image. To have a branch NSBrowserCell with no image (and no space reserved for an image), override this method to return `null`.

See Also

`highlightedBranchImage` (page 247)

`alternateImage` (page 247)

`setAlternateImage` (page 248)

highlightedBranchImage

Returns the default NSImage for branch NSBrowserCells that are highlighted (a lighter version of the image returned by `branchImage` (page 247)).

```
public static NSImage highlightedBranchImage()
```

Discussion

Override this method if you want a different image.

See Also

`branchImage` (page 247)

`alternateImage` (page 247)

`setAlternateImage` (page 248)

Instance Methods

alternateImage

Returns this receiver's image for the highlighted state or `null` if no image is set.

```
public NSImage alternateImage()
```

See Also

`setAlternateImage` (page 248)

highlightColorInView

Returns the highlight color that the receiver wants to display as in *`controlView`*.

```
public NSColor highlightColorInView(NSView controlView)
```

isLeaf

Returns whether the receiver is a leaf or a branch cell.

```
public boolean isLeaf()
```

Discussion

A branch NSBrowserCell has an image near its right edge indicating that more, hierarchically related information is available; when the user selects the cell, the NSBrowser displays a new column of NSBrowserCells. A leaf NSBrowserCell has no image, indicating that the user has reached a terminal piece of information; it doesn't point to additional information.

See Also

`setLeaf` (page 249)

isLoaded

Returns `true` if the receiver's state has been set and the cell is ready to display.

```
public boolean isLoaded()
```

See Also

`setLoaded` (page 249)

reset

Unhighlights the receiver and unsets its state.

```
public void reset()
```

See Also

`set` (page 248)

set

Highlights the receiver and sets its state.

```
public void set()
```

See Also

`reset` (page 248)

setAlternateImage

Sets the receiver's image for the highlighted state.

```
public void setAlternateImage(NSImage newAltImage)
```

Discussion

If *newAltImage* is `null`, it removes the alternate image for the receiver. *newAltImage* is drawn vertically centered on the left edge of the browser cell. Note that *newAltImage* is drawn at the given size of the image. NSBrowserCell does not set the size of the image, nor does it clip the drawing of the image. Make sure *newAltImage* is the correct size for drawing in the browser cell.

See Also

`alternateImage` (page 247)

setLeaf

Sets whether the receiver is a leaf or a branch cell, depending on the Boolean value *flag*.

```
public void setLeaf(boolean flag)
```

Discussion

A branch NSBrowserCell has an image near its right edge indicating that more, hierarchically related information is available; when the user selects the cell, the NSBrowser displays a new column of NSBrowserCells. A leaf NSBrowserCell has no image, indicating that the user has reached a terminal piece of information; it doesn't point to additional information.

See Also

`isLeaf` (page 248)

setLoaded

Sets whether the receiver's state has been set and the cell is ready to display, depending on the Boolean value *flag*.

```
public void setLoaded(boolean flag)
```

See Also

`isLoaded` (page 248)

NSButton

Inherits from	NSControl : NSView : NSResponder : NSObject
Implements	NSCoding (NSResponder)
Package:	com.apple.cocoa.application
Companion guide	Button Programming Topics for Cocoa

Overview

NSButton is a subclass of NSControl that intercepts mouse-down events and sends an action message to a target object when it's clicked or pressed.

NSButton uses NSButtonCell (page 269) to implement its user interface.

NSButton and NSMatrix both provide a control view, which is needed to display an NSButtonCell object. However, while NSMatrix requires you to access the NSButtonCells 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` (page 261) method).

Tasks

Constructors

`NSButton` (page 254)
Creates an NSButton with a zero-sized frame rectangle.

Setting the Button Type

`setButtonType` (page 261)
Sets how the receiver button highlights while pressed and how it shows its state.

Setting the State

allowsMixedState (page 255)
: Returns true if the receiver has three states: on, off, and mixed.

setAllowsMixedState (page 259)

setNextState (page 263)
: Sets the receiver to its next state.

setState (page 264)
: Sets the cell's state to *value*, which can be NSCell.OnState, NSCell.OffState, or NSCell.MixedState.

state (page 266)
: Returns the receiver's state.

Setting the Repeat Interval

interval (page 257)
: Returns the amount of time (in seconds) between the periodic messages that a continuous button sends after its been pressed for a sufficient time.

periodicDelay (page 259)
: Returns the amount of time (in seconds) that a continuous button will pause before starting to periodically send action messages to the target object.

setPeriodicDelayAndInterval (page 263)
: Sets the message delay and interval for the receiver.

Setting the Titles

alternateTitle (page 255)
: Returns the string that appears on the receiver when it's in its alternate state, or the empty string if the receiver doesn't display an alternate title.

attributedAlternateTitle (page 256)
: Returns the string that appears on the receiver when it's in its alternate state as an NSAttributedString, or an empty attributed string if the receiver doesn't display an alternate title.

attributedTitle (page 256)
: Returns the string that appears on the receiver when it's in its normal state as an NSAttributedString, or an empty attributed string if the receiver doesn't display a title.

setAlternateTitle (page 260)
: Sets the string that appears on the receiver when it's in its alternate state to *aString*.

setAttributedAlternateTitle (page 260)
: Sets the string that appears on the receiver when it's in its alternate state to the attributed string *aString*.

setAttributedTitle (page 260)
: Sets the string that appears on the receiver when it's in its normal state to the attributed string *aString* and redraws the button.

`setTitle` (page 265)
: Sets the title displayed by the receiver when in its normal state to *aString* and, if necessary, redraws the button's contents.

`setTitleWithMnemonic` (page 265)
: Sets the title of a button with a character denoting an access key, as specified by *aString*.

`title` (page 267)
: Returns the title displayed on the receiver when it's in its normal state (this title is always displayed if the button doesn't use its alternate contents for highlighting or displaying the alternate state).

Setting the Images

`alternateImage` (page 255)
: Returns the image that appears on the receiver when it's in its alternate state, or `null` if there is no alternate image.

`image` (page 257)
: Returns the image that appears on the receiver when it's in its normal state, or `null` if there is no such image.

`imagePosition` (page 257)
: Returns the position of the receiver's image relative to its title.

`setAlternateImage` (page 260)
: Sets the image that appears on the receiver when it's in its alternate state to *image* and, if necessary, redraws the contents of the button.

`setImage` (page 262)
: Sets the receiver's image to *anImage* and redraws the button.

`setImagePosition` (page 262)
: Sets the position of the receiver's image relative to its title.

Modifying Graphics Attributes

`bezelStyle` (page 256)
: Returns the appearance of the receiver's border.

`isBordered` (page 258)
: Returns `true` if the receiver has a border, `false` otherwise.

`isTransparent` (page 258)
: Returns `true` if the receiver is transparent, `false` otherwise.

`setBordered` (page 261)
: Sets whether the receiver has a bezeled border.

`setBezelStyle` (page 261)
: Sets the appearance of the border, if the receiver has one.

`setShowsBorderOnlyWhileMouseInside` (page 264)
: Sets whether the receiver's border is displayed only when the cursor is over the button.

`setTransparent` (page 266)
: Sets whether the receiver is transparent, depending on the Boolean value *flag*, and redraws the receiver if necessary.

`showsBorderOnlyWhileMouseInside` (page 266)
: Returns `true` if the receiver's border is displayed only when the cursor is over the button and the button is active.

Displaying

`highlight` (page 256)
: Highlights (or unhighlights) the receiver according to *`flag`*.

Setting the Key Equivalent

`keyEquivalent` (page 258)
: Returns the key-equivalent character of the receiver, or the empty string if one hasn't been defined.

`keyEquivalentModifierMask` (page 258)
: Returns the mask indicating the modifier keys that are applied to the receiver's key equivalent.

`setKeyEquivalent` (page 262)
: Sets the key equivalent character of the receiver to the character represented by *`charCode`* and redraws the button's interior if it displays a key equivalent instead of an image.

`setKeyEquivalentModifierMask` (page 263)
: Sets the mask indicating the modifier keys to be applied to the receiver's key equivalent using the value of *`mask`*.

Handling Events and Action Messages

`performKeyEquivalent` (page 259)

Playing Sound

`setSound` (page 264)
: Sets the sound that's played when the user presses the button to *`aSound`*.

`sound` (page 266)
: Returns the sound that's played when the user presses the button.

Constructors

NSButton

Creates an NSButton with a zero-sized frame rectangle.

```
public NSButton()
```

Creates an NSButton with *`frameRect`* as its frame rectangle.

```
public NSButton(NSRect frameRect)
```

Instance Methods

allowsMixedState

Returns `true` if the receiver has three states: on, off, and mixed.

```
public boolean allowsMixedState()
```

Discussion
Returns `false` if the receiver has two states: on and off. The default is `false`.

See Also
`setAllowsMixedState` (page 259)
`setNextState` (page 263)

alternateImage

Returns the image that appears on the receiver when it's in its alternate state, or `null` if there is no alternate image.

```
public NSImage alternateImage()
```

Discussion
Note that some button types don't display an alternate image. Buttons don't display images by default.

See Also
`setAlternateImage` (page 260)
`image` (page 257)
`imagePosition` (page 257)
`keyEquivalent` (page 258)
`setButtonType` (page 261)

alternateTitle

Returns the string that appears on the receiver when it's in its alternate state, or the empty string if the receiver doesn't display an alternate title.

```
public String alternateTitle()
```

Discussion
Note that some button types don't display an alternate title. By default, a button's alternate title is "Button."

See Also
`setAlternateTitle` (page 260)
`attributedAlternateTitle` (page 256)
`setButtonType` (page 261)

`title` (page 267)

attributedAlternateTitle

Returns the string that appears on the receiver when it's in its alternate state as an NSAttributedString, or an empty attributed string if the receiver doesn't display an alternate title.

```
public NSAttributedString attributedAlternateTitle()
```

Discussion
Note that some button types don't display an alternate title. By default, a button's alternate title is "Button."

See Also
`setAttributedAlternateTitle` (page 260)
`attributedTitle` (page 256)
`setButtonType` (page 261)

attributedTitle

Returns the string that appears on the receiver when it's in its normal state as an NSAttributedString, or an empty attributed string if the receiver doesn't display a title.

```
public NSAttributedString attributedTitle()
```

Discussion
A button's title is always displayed if the button doesn't use its alternate contents for highlighting or displaying the alternate state. By default, a button's title is "Button."

See Also
`setAttributedTitle` (page 260)
`attributedAlternateTitle` (page 256)
`setButtonType` (page 261)

bezelStyle

Returns the appearance of the receiver's border.

```
public int bezelStyle()
```

Discussion
See NSButtonCell's "Constants" (page 289) section for the list of possible values.

See Also
`setBezelStyle` (page 261)

highlight

Highlights (or unhighlights) the receiver according to *flag*.

```
public void highlight(boolean flag)
```

Discussion

Highlighting may involve the button appearing "pushed in" to the screen, displaying its alternate title or image, or causing the button to appear to be "lit." If the current state of the button matches *flag*, no action is taken.

See Also

`setButtonType` (page 261)

image

Returns the image that appears on the receiver when it's in its normal state, or `null` if there is no such image.

```
public NSImage image()
```

Discussion

This image is always displayed on a button that doesn't change its contents when highlighting or showing its alternate state. Buttons don't display images by default.

See Also

`setImage` (page 262)

`alternateImage` (page 255)

`setButtonType` (page 261)

imagePosition

Returns the position of the receiver's image relative to its title.

```
public int imagePosition()
```

Discussion

The return value is one of the image positions described in NSCell's "Constants" (page 337) section.

If the title is above, below, or overlapping the image, or if there is no image, the text is horizontally centered within the button.

See Also

`setImagePosition` (page 262)

`setButtonType` (page 261)

`setImage` (page 262)

`setTitle` (page 265)

interval

Returns the amount of time (in seconds) between the periodic messages that a continuous button sends after its been pressed for a sufficient time.

```
public float interval()
```

Discussion

The default value is taken from a user's defaults (60 seconds maximum). If the user hasn't specified a value, it defaults to 0.075 seconds.

See Also

isContinuous (page 452) (NSControl)

periodicDelay (page 259)

setPeriodicDelayAndInterval (page 263)

isBordered

Returns true if the receiver has a border, false otherwise.

```
public boolean isBordered()
```

Discussion

A button's border isn't the single line of most other controls' borders—instead, it's a raised bezel. By default, buttons are bordered.

See Also

setBordered (page 261)

isTransparent

Returns true if the receiver is transparent, false otherwise.

```
public boolean isTransparent()
```

Discussion

A transparent button never draws itself, but it receives mouse-down events and tracks the mouse properly.

See Also

setTransparent (page 266)

keyEquivalent

Returns the key-equivalent character of the receiver, or the empty string if one hasn't been defined.

```
public String keyEquivalent()
```

Discussion

Buttons don't have a default key equivalent.

See Also

setKeyEquivalent (page 262)

performKeyEquivalent (page 259)

keyEquivalentFont (page 279) (NSButtonCell)

keyEquivalentModifierMask

Returns the mask indicating the modifier keys that are applied to the receiver's key equivalent.

```
public int keyEquivalentModifierMask()
```

Discussion

The only mask bits relevant in button key-equivalent modifier masks are `NSEvent.ControlKeyMask`, `NSEvent.AlternateKeyMask`, and `NSEvent.CommandKeyMask` bits.

See Also

`setKeyEquivalentModifierMask` (page 263)

`keyEquivalent` (page 258)

performKeyEquivalent

```
public boolean performKeyEquivalent(NSEvent anEvent)
```

Discussion

If the character in *anEvent* matches the receiver's key equivalent, and the modifier flags in *anEvent* match the key-equivalent modifier mask, `performKeyEquivalent` simulates the user clicking the button and returning `true`. Otherwise, `performKeyEquivalent` does nothing and returns `false`. `performKeyEquivalent` also returns `false` in the event that the receiver is blocked by a modal panel or the button is disabled.

See Also

`keyEquivalent` (page 258)

`keyEquivalentModifierMask` (page 258)

periodicDelay

Returns the amount of time (in seconds) that a continuous button will pause before starting to periodically send action messages to the target object.

```
public float periodicDelay()
```

Discussion

The default value is taken from a user's defaults (60 seconds maximum). If the user hasn't specified a value, it defaults to 0.4 seconds.

See Also

`interval` (page 257)

`setPeriodicDelayAndInterval` (page 263)

`isContinuous` (page 452) (NSControl)

setAllowsMixedState

```
public void setAllowsMixedState(boolean flag)
```

Discussion

If *flag* is `true`, the receiver has three states: on, off, and mixed. If flag is `false`, the receiver has two states: on and off.

See Also

`allowsMixedState` (page 255)

`setNextState` (page 263)

setAlternateImage

Sets the image that appears on the receiver when it's in its alternate state to *image* and, if necessary, redraws the contents of the button.

```
public void setAlternateImage(NSImage image)
```

Discussion
Note that some button types don't display an alternate image.

See Also
alternateImage (page 255)
setButtonType (page 261)
setImage (page 262)

setAlternateTitle

Sets the string that appears on the receiver when it's in its alternate state to *aString*.

```
public void setAlternateTitle(String aString)
```

Discussion
Note that some button types don't display an alternate title.

See Also
alternateTitle (page 255)
setTitle (page 265)
setTitleWithMnemonic (page 265)
setButtonType (page 261)
setFont (page 283) (NSButtonCell)

setAttributedAlternateTitle

Sets the string that appears on the receiver when it's in its alternate state to the attributed string *aString*.

```
public void setAttributedAlternateTitle(NSAttributedString aString)
```

Discussion
Note that some button types don't display an alternate title.

See Also
attributedAlternateTitle (page 256)
setAttributedTitle (page 260)
setButtonType (page 261)
setFont (page 283) (NSButtonCell)

setAttributedTitle

Sets the string that appears on the receiver when it's in its normal state to the attributed string *aString* and redraws the button.

```
public void setAttributedTitle(NSAttributedString aString)
```

Discussion

The title is always shown on buttons that don't use their alternate contents when highlighting or displaying their alternate state.

See Also

`attributedTitle` (page 256)

`setAttributedAlternateTitle` (page 260)

`setButtonType` (page 261)

`setFont` (page 283) (NSButtonCell)

setBezelStyle

Sets the appearance of the border, if the receiver has one.

```
public void setBezelStyle(int bezelStyle)
```

Discussion

`bezelStyle` must be one of the bezel styles described in NSButtonCell's "Constants" (page 289) section.

The button uses shading to look like it's sticking out or pushed in. You can set the shading with NSButtonCell's `setGradientType` (page 284) method.

If the button is not bordered, the bezel style is ignored.

See Also

`bezelStyle` (page 256)

setBordered

Sets whether the receiver has a bezeled border.

```
public void setBordered(boolean flag)
```

Discussion

If *`flag`* is `true`, the receiver displays a border; if *`flag`* is `false`, the receiver doesn't display a border. A button's border is not the single line of most other controls' borders—instead, it's a raised bezel. This method redraws the button if `setBordered` causes the bordered state to change.

See Also

`isBordered` (page 258)

setButtonType

Sets how the receiver button highlights while pressed and how it shows its state.

```
public void setButtonType(int aType)
```

Discussion

`setButtonType` redisplays the button before returning.

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` (page 284) and `setShowsStateBy` (page 287) methods.

The *aType* argument can be one of eight constants described in "Constants" (page 267).

See Also

`setAlternateImage` (page 260)

`setImage` (page 262)

`setButtonType` (page 283) (NSButtonCell)

setImage

Sets the receiver's image to *anImage* and redraws the button.

```
public void setImage(NSImage anImage)
```

Discussion

A button's image is displayed when the button is in its normal state, or all the time for a button that doesn't change its contents when highlighting or displaying its alternate state.

See Also

`image` (page 257)

`setImagePosition` (page 262)

`setAlternateImage` (page 260)

`setButtonType` (page 261)

setImagePosition

Sets the position of the receiver's image relative to its title.

```
public void setImagePosition(int aPosition)
```

Discussion

See NSCell's "Constants" (page 337) section for a listing of possible values for *aPosition*.

See Also

`imagePosition` (page 257)

setKeyEquivalent

Sets the key equivalent character of the receiver to the character represented by *charCode* and redraws the button's interior if it displays a key equivalent instead of an image.

```
public void setKeyEquivalent(String charCode)
```

Discussion

The key equivalent isn't displayed if the image position is set to `NSCell.NoImage`, `NSCell.ImageOnly`, or `NSCell.ImageOverlaps`; 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 `null`, then set the key equivalent, then set the image position.

See Also

`keyEquivalent` (page 258)

`performKeyEquivalent` (page 259)

`setAlternateImage` (page 260)

`setImage` (page 262)

`setImagePosition` (page 262)

`setKeyEquivalentFont` (page 285) (NSButtonCell)

setKeyEquivalentModifierMask

Sets the mask indicating the modifier keys to be applied to the receiver's key equivalent using the value of *mask*.

```
public void setKeyEquivalentModifierMask(int mask)
```

Discussion

The only mask bits relevant in button key-equivalent modifier masks are `NSEvent.ControlKeyMask`, `NSEvent.AlternateKeyMask`, and `NSEvent.CommandKeyMask` bits.

See Also

`keyEquivalentModifierMask` (page 258)

`setKeyEquivalent` (page 262)

setNextState

Sets the receiver to its next state.

```
public void setNextState()
```

Discussion

If the button has three states, it cycles through them in this order: on, off, mixed, on, and so forth. If the button has two states, it toggles between them.

See Also

`allowsMixedState` (page 255)

`setAllowsMixedState` (page 259)

setPeriodicDelayAndInterval

Sets the message delay and interval for the receiver.

```
public void setPeriodicDelayAndInterval(float delay, float interval)
```

Discussion

These two values are used if the button is configured (by a setContinuous (page 456) 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 *delay* and *interval* is 60.0 seconds; if a larger value is supplied, it is ignored, and 60.0 seconds is used.

See Also

setContinuous (page 456) (NSControl)

setShowsBorderOnlyWhileMouseInside

Sets whether the receiver's border is displayed only when the cursor is over the button.

```
public void setShowsBorderOnlyWhileMouseInside(boolean show)
```

Discussion

If *show* is true, the border is displayed only when the cursor is within the button's border and the button is active. If *show* is false, the button's border continues to be displayed when the cursor is outside button's bounds.

If isBordered (page 258) returns false, the border is never displayed, regardless of what this method returns.

See Also

showsBorderOnlyWhileMouseInside (page 266)

setSound

Sets the sound that's played when the user presses the button to *aSound*.

```
public void setSound(Object aSound)
```

Discussion

The sound is played during a mouse-down event, such as NSEvent.LeftMouseDown.

See Also

sound (page 266)

setState

Sets the cell's state to *value*, which can be NSCell.OnState, NSCell.OffState, or NSCell.MixedState.

```
public void setState(int value)
```

Discussion

If necessary, this method also redraws the receiver.

The cell can have two or three states. If it has two, *value* can be NSCell.OffState (the normal or unpressed state) and NSCell.OnState (the alternate or pressed state). If it has three, *value* can be NSCell.OnState (the feature is in effect everywhere), NSCell.OffState (the feature is in effect nowhere), or NSCell.MixedState (the feature is in effect somewhere). Note that if the cell has only two states and *value* is NSCell.MixedState, this method sets the cell's state to NSCell.OnState.

Although using the enumerated constants is preferred, *value* can also be an integer. If the cell has two states, 0 is treated as NSCell.OffState, and a nonzero value is treated as NSCell.OnState. If the cell has three states, 0 is treated as NSCell.OffState; a negative value, as NSCell.MixedState; and a positive value, as NSCell.OnState.

To check whether the button uses the mixed state, use the method allowsMixedState (page 255).

See Also

state (page 266)

setTitle

Sets the title displayed by the receiver when in its normal state to *aString* and, if necessary, redraws the button's contents.

```
public void setTitle(String aString)
```

Discussion

This title is always shown on buttons that don't use their alternate contents when highlighting or displaying their alternate state.

See Also

title (page 267)

setAlternateTitle (page 260)

setButtonType (page 261)

setTitleWithMnemonic (page 265)

setFont (page 283) (NSButtonCell)

setTitleWithMnemonic

Sets the title of a button with a character denoting an access key, as specified by *aString*.

```
public void setTitleWithMnemonic(String aString)
```

Discussion

Mnemonics are not supported in Mac OS X.

See Also

title (page 267)

setAlternateTitle (page 260)

setButtonType (page 261)

setTitle (page 265)

setFont (page 283) (NSButtonCell)

setTransparent

Sets whether the receiver is transparent, depending on the Boolean value *flag*, and redraws the receiver if necessary.

```
public void setTransparent(boolean flag)
```

Discussion

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 (page 258)

showsBorderOnlyWhileMouseInside

Returns true if the receiver's border is displayed only when the cursor is over the button and the button is active.

```
public boolean showsBorderOnlyWhileMouseInside()
```

Discussion

By default, this method returns false.

If isBordered (page 258) returns false, the border is never displayed, regardless of what this method returns.

See Also

setShowsBorderOnlyWhileMouseInside (page 264)

sound

Returns the sound that's played when the user presses the button.

```
public Object sound()
```

See Also

setSound (page 264)

state

Returns the receiver's state.

```
public int state()
```

Discussion

The button can have two or three states. If it has two, it returns either NSCell.OffState (the normal or unpressed state) or NSCell.OnState (the alternate or pressed state). If it has three, it returns NSCell.OnState (the feature is in effect everywhere), NSCell.OffState (the feature is in effect nowhere), or NSCell.MixedState (the feature is in effect somewhere).

To check whether the button uses the mixed state, use the method `allowsMixedState` (page 255).

See Also

`setState` (page 264)

title

Returns the title displayed on the receiver when it's in its normal state (this title is always displayed if the button doesn't use its alternate contents for highlighting or displaying the alternate state).

```
public String title()
```

Discussion

Returns the empty string if the button doesn't display a title. By default, a button's title is "Button."

See Also

`alternateTitle` (page 255)

`setButtonType` (page 261)

`setTitle` (page 265)

`setTitleWithMnemonic` (page 265)

Constants

NSButton defines a number of constants to indicate, using `setButtonType` (page 261), how a button highlights while pressed and how it shows its state:

Button Type	Description
`MomentaryLight`	While the button is held down it's shown as "lit." 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 Light" in Interface Builder's Button Inspector. This type is the default button type.
`MomentaryPush`	While the button is held down it's shown 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.
`MomentaryChange`	While the button is held down, the alternate image and alternate title are displayed. The normal image and title are displayed when the button isn't pressed. This option is called "Momentary Change" in Interface Builder's Button Inspector.
`PushOnPushOff`	The first click both highlights the button and causes it to be "pushed in" if the button is bordered. A second click returns it to its normal state. This option is called "Push On/Push Off" in Interface Builder's Button Inspector.
`OnOff`	The first click highlights the button. A second click returns it to the normal (unhighlighted) state. This option is called "On/Off" in Interface Builder's Button Inspector.

Button Type	Description
`Toggle`	The first click highlights the button, while a second click returns it to its normal state. Highlighting is performed by changing to the alternate title or image and showing the button as “pushed in” if the button is bordered. This option is called “Toggle” in Interface Builder’s Button Inspector.
`Switch`	This type is a variant of `Toggle` that has no border. This type of button is available as a separate palette item in Interface Builder.
`Radio`	Similar to `Switch`. This type of button is available as a separate palette item in Interface Builder.

NSButtonCell

Inherits from	NSActionCell : NSCell : NSObject
Implements	NSCoding (NSCell)
Package:	com.apple.cocoa.application
Companion guide	Button Programming Topics for Cocoa

Overview

NSButtonCell is a subclass of NSActionCell used to implement 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. The NSButton subclass of NSControl uses a single NSButtonCell.

NSButtonCell implements the user interface of NSButton (page 251).

Setting an NSButtonCell's integer, float, double, or object value results in a call to `setState` (page 330) with the value converted to integer. In the case of `setObjectValue` (page 328), `null` is equivalent to 0, and a non-`null` object that doesn't respond to `intValue` (page 313) sets the state to 1. Otherwise, the state is set to the object's `intValue`. Similarly, querying the integer, float, double, or object value of an NSButtonCell returns the current state in the requested representation. In the case of `objectValue` (page 317), this is a number containing `true` for on, `false` for off, and integer value -1 for the mixed state.

For more information on NSButtonCell's behavior, see the NSButton (page 251) and NSMatrix (page 875) class specifications.

Exceptions

In its implementation of the `compare` (page 308) method (declared in NSCell), NSButtonCell throws a `BadComparisonException` if the *`otherCell`* argument is not of the NSButtonCell class.

Tasks

Constructors

`NSButtonCell` (page 273)
Creates an empty NSButtonCell.

Setting the Titles

`alternateMnemonic` (page 274)
: Returns the character in the alternate title (the title displayed on the receiver when it's in its alternate state) that's marked as the "keyboard mnemonic."

`alternateMnemonicLocation` (page 274)
: Returns an unsigned integer indicating the character in the alternate title (the title displayed on the receiver when it's in its alternate state) that's marked as the "keyboard mnemonic."

`alternateTitle` (page 275)
: Returns the string that appears on the receiver when it's in its alternate state, or the empty string if the receiver doesn't display an alternate title.

`attributedAlternateTitle` (page 275)
: Returns the string that appears on the receiver when it's in its alternate state as an NSAttributedString, or an empty attributed string if the receiver doesn't display an alternate title.

`attributedTitle` (page 275)
: Returns the string that appears on the receiver when it's in its normal state as an NSAttributedString, or an empty attributed string if the receiver doesn't display a title.

`setAlternateMnemonicLocation` (page 281)
: Sets the character in the alternate title (the title displayed on the receiver when it's in its alternate state) that's to be marked as the "keyboard mnemonic."

`setAlternateTitle` (page 281)
: Sets the title that's displayed on the receiver when it's in its alternate state to *aString*.

`setAlternateTitleWithMnemonic` (page 281)
: Sets the title that is displayed on the receiver when it's in its alternate state to *aString*, taking into account the fact that an embedded "&" character is not a literal but instead marks the alternate state's "keyboard mnemonic."

`setAttributedAlternateTitle` (page 282)
: Sets the string that appears on the receiver when it's in its alternate state to the attributed string *aString*.

`setAttributedTitle` (page 282)
: Sets the string that appears on the receiver when it's in its normal state to the attributed string *aString* and redraws the button.

`setFont` (page 283)
: Sets the font used to display the title and alternate title to the font specified by *fontObj*.

`setTitle` (page 287)
: Sets the title displayed by the receiver when in its normal state to *aString* and, if necessary, redraws the receiver's contents.

`setTitleWithMnemonic` (page 288)
: Sets the title displayed on the receiver when it's in its normal state to *aString*, taking into account the fact that an embedded "&" character is not a literal but instead marks the normal state's "keyboard mnemonic."

`title` (page 289)
: Returns the title displayed on the receiver when it's in its normal state (this title is always displayed if the button doesn't use its alternate contents for highlighting or displaying the alternate state).

Setting the Images

alternateImage (page 274)
Returns the image that appears on the receiver when it's in its alternate state, or null if there is no alternate image.

imagePosition (page 278)
Returns the position of the receiver's image relative to its title.

setAlternateImage (page 280)
Sets the image that appears on the receiver when it's in its alternate state to *image* and, if necessary, redraws the contents of the receiver.

setImagePosition (page 285)
Sets the position of the receiver's image relative to its title.

Setting the Repeat Interval

setPeriodicDelayAndInterval (page 286)
Sets the message delay and interval for the receiver.

Setting the Key Equivalent

keyEquivalent (page 279)
Returns the key-equivalent character of the receiver, or the empty string if one hasn't been defined.

keyEquivalentFont (page 279)
Returns the font used to draw the key equivalent, or null if the receiver doesn't have a key equivalent.

keyEquivalentModifierMask (page 279)
Returns the mask indicating the modifier keys that are applied to the receiver's key equivalent.

setKeyEquivalent (page 285)
Sets the key equivalent character of the receiver, specified by *aKeyEquivalent*, and redraws the receiver's inside if it displays a key equivalent instead of an image.

setKeyEquivalentModifierMask (page 286)
Sets the mask indicating the modifier keys to be applied to the receiver's key equivalent using the value of *mask*.

setKeyEquivalentFont (page 285)
Sets the font used to draw the key equivalent to the font specified by *fontObj* and redisplays the receiver if necessary.

setKeyEquivalentFontAndSize (page 286)
Sets by name and size the font used to draw the key equivalent and redisplays the receiver if necessary.

Modifying Graphics Attributes

backgroundColor (page 276)
Returns the background color of the button.

bezelStyle (page 276)
Returns the appearance of the receiver's border.

`gradientType` (page 277)
Returns the gradient of the receiver's border.

`imageDimsWhenDisabled` (page 278)
Returns whether the receiver's image and text appear "dim" when the receiver is disabled.

`isOpaque` (page 278)
Returns `true` if the receiver draws over every pixel in its frame, `false` if not.

`isTransparent` (page 279)
Returns `true` if the receiver is transparent, `false` otherwise.

`setBackgroundColor` (page 282)
Sets the background color of the button.

`setBezelStyle` (page 283)
Sets the appearance of the border, if the receiver has one.

`setShowsBorderOnlyWhileMouseInside` (page 286)
Sets whether the receiver's border is displayed only when the cursor is over the button.

`setGradientType` (page 284)
Sets the type of gradient to use for the receiver.

`setImageDimsWhenDisabled` (page 284)
Sets whether the receiver's image appears "dim" when the button cell is disabled, depending on the Boolean value *flag*.

`setTransparent` (page 288)

`showsBorderOnlyWhileMouseInside` (page 288)
Returns `true` if the receiver's border is displayed only when the cursor is over the button and the button is active.

Displaying

`highlightsBy` (page 277)
Returns the logical OR of flags that indicate the way the receiver highlights when it receives a mouse-down event.

`setHighlightsBy` (page 284)
Sets the way the receiver highlights itself while pressed.

`setShowsStateBy` (page 287)
Sets the way the receiver indicates its alternate state.

`setButtonType` (page 283)
Sets how the receiver highlights while pressed and how it shows its state.

`showsStateBy` (page 289)
Returns the logical OR of flags that indicate the way the receiver shows its alternate state.

Playing Sound

`setSound` (page 287)
Sets the sound that's played when the user presses the receiver to *aSound*.

`sound` (page 289)
Returns the sound that's played when the user presses the button.

Handling Events and Action Messages

`mouseEntered` (page 280)
Draws the receiver's border.

`mouseExited` (page 280)
Erases the receiver's border.

`performClick` (page 280)
Simulates the user clicking the receiver with the cursor.

Drawing the Button Content

`drawBezel` (page 276)
Draws the border of the button using the current bezel style.

`drawImage` (page 276)
Draws the *`image`* associated with the button's current state.

`drawTitle` (page 277)
Draws the button's *`title`* centered vertically in the specified *`frame`* rectangle.

Constructors

NSButtonCell

Creates an empty NSButtonCell.

```
public NSButtonCell()
```

Creates an NSButtonCell initialized with *`aString`*.

```
public NSButtonCell(String aString)
```

Creates an NSButtonCell initialized with *`anImage`*.

```
public NSButtonCell(NSImage anImage)
```

Discussion
If *`anImage`* is `null`, no image is set.

Instance Methods

alternateImage

Returns the image that appears on the receiver when it's in its alternate state, or `null` if there is no alternate image.

```
public NSImage alternateImage()
```

Discussion

Note that some button types don't display an alternate image. Buttons don't display images by default.

See Also

`setAlternateImage` (page 280)

`imagePosition` (page 278)

`keyEquivalent` (page 279)

`setButtonType` (page 283)

`image` (page 313) (NSCell)

alternateMnemonic

Returns the character in the alternate title (the title displayed on the receiver when it's in its alternate state) that's marked as the "keyboard mnemonic."

```
public String alternateMnemonic()
```

Discussion

Mnemonics are not supported in Mac OS X.

See Also

`alternateMnemonicLocation` (page 274)

`setAlternateTitleWithMnemonic` (page 281)

`mnemonic` (page 316) (NSCell)

alternateMnemonicLocation

Returns an unsigned integer indicating the character in the alternate title (the title displayed on the receiver when it's in its alternate state) that's marked as the "keyboard mnemonic."

```
public int alternateMnemonicLocation()
```

Discussion

If the alternate title doesn't have a keyboard mnemonic, `NSArray.NotFound` is returned. Mnemonics are not supported in Mac OS X.

See Also

`setAlternateMnemonicLocation` (page 281)

`alternateMnemonic` (page 274)

`setAlternateTitleWithMnemonic` (page 281) mnemonicLocation (page 317) (NSCell)

alternateTitle

Returns the string that appears on the receiver when it's in its alternate state, or the empty string if the receiver doesn't display an alternate title.

```
public String alternateTitle()
```

Discussion

Note that some button types don't display an alternate title. By default, a button's alternate title is "Button."

See Also

setAlternateTitle (page 281)

alternateMnemonic (page 274)

attributedAlternateTitle (page 275)

setButtonType (page 283)

title (page 289)

attributedAlternateTitle

Returns the string that appears on the receiver when it's in its alternate state as an NSAttributedString, or an empty attributed string if the receiver doesn't display an alternate title.

```
public NSAttributedString attributedAlternateTitle()
```

Discussion

Note that some button types don't display an alternate title. By default, a button's alternate title is "Button."

See Also

setAttributedAlternateTitle (page 282)

alternateMnemonic (page 274)

attributedTitle (page 275)

setButtonType (page 283)

attributedTitle

Returns the string that appears on the receiver when it's in its normal state as an NSAttributedString, or an empty attributed string if the receiver doesn't display a title.

```
public NSAttributedString attributedTitle()
```

Discussion

A button's title is always displayed if the button doesn't use its alternate contents for highlighting or displaying the alternate state. By default, a button's title is "Button."

See Also

setAttributedTitle (page 282)

attributedAlternateTitle (page 275)

setButtonType (page 283)

`mnemonic` (page 316) (NSCell)

backgroundColor

Returns the background color of the button.

```
public NSColor backgroundColor()
```

Discussion
The background color is used only when drawing borderless buttons.

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

See Also
`setBackgroundColor` (page 282)

bezelStyle

Returns the appearance of the receiver's border.

```
public int bezelStyle()
```

Discussion
See "Constants" (page 289) for the list of the possible values.

See Also
`setBezelStyle` (page 283)

drawBezel

Draws the border of the button using the current bezel style.

```
public void drawBezel(NSRect frame, NSView controlView)
```

Discussion
This method is called automatically when the button is redrawn; you should not call it directly. The *frame* parameter contains the bounding rectangle of the button. The *controlView* parameter contains the control being drawn.

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

See Also
`setBezelStyle` (page 283)

drawImage

Draws the *image* associated with the button's current state.

```
public void drawImage(NSImage image, NSRect frame, NSView controlView)
```

Discussion
This method is called automatically when the button is redrawn; you should not call it directly. The `frame` parameter contains the bounding rectangle of the button. The `controlView` parameter contains the control being drawn.

You specify the primary and alternate images for the button using Interface Builder.

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

See Also
`setAlternateImage` (page 280)

drawTitle

Draws the button's `title` centered vertically in the specified `frame` rectangle.

```
public NSRect drawTitle(NSAttributedString title, NSRect frame, NSView controlView)
```

Discussion
This method is called automatically when the button is redrawn; you should not call it directly. The `controlView` parameter contains the control being drawn.

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

See Also
`setAlternateTitle` (page 281)
`setAttributedTitle` (page 282)

gradientType

Returns the gradient of the receiver's border.

```
public int gradientType()
```

Discussion
See "Constants" (page 289) for the list of the possible values.

See Also
`setGradientType` (page 284)

highlightsBy

Returns the logical OR of flags that indicate the way the receiver highlights when it receives a mouse-down event.

```
public int highlightsBy()
```

Discussion

See NSCell's "Constants" (page 337) section for the list of flags.

See Also

`setHighlightsBy` (page 284)
`showsStateBy` (page 289)

imageDimsWhenDisabled

Returns whether the receiver's image and text appear "dim" when the receiver is disabled.

```
public boolean imageDimsWhenDisabled()
```

Discussion

By default, all button types except `SwitchButton` and `RadioButton` do dim when disabled. When `SwitchButton`s and `RadioButton`s are disabled, only the associated text dims.

See Also

`setButtonType` (page 283)
`setImageDimsWhenDisabled` (page 284)

imagePosition

Returns the position of the receiver's image relative to its title.

```
public int imagePosition()
```

Discussion

The return value is one of the image positions described in NSCell's "Constants" (page 337) section.

If the title is above, below, or overlapping the image, or if there is no image, the text is horizontally centered within the button.

See Also

`setImagePosition` (page 285)
`setButtonType` (page 283)
`setTitle` (page 287)
`setImage` (page 327) (NSCell)

isOpaque

Returns `true` if the receiver draws over every pixel in its frame, `false` if not.

```
public boolean isOpaque()
```

Discussion

A button cell is opaque only if it isn't transparent and if it has a border.

See Also

`isTransparent` (page 279)
`setTransparent` (page 288)

isTransparent

Returns `true` if the receiver is transparent, `false` otherwise.

```
public boolean isTransparent()
```

Discussion

A transparent button never draws itself, but it receives mouse-down events and tracks the mouse properly.

See Also

`setTransparent` (page 288)

`isOpaque` (page 278)

keyEquivalent

Returns the key-equivalent character of the receiver, or the empty string if one hasn't been defined.

```
public String keyEquivalent()
```

Discussion

Buttons don't have a default key equivalent.

See Also

`setKeyEquivalent` (page 285)

`keyEquivalentFont` (page 279)

keyEquivalentFont

Returns the font used to draw the key equivalent, or `null` if the receiver doesn't have a key equivalent.

```
public NSFont keyEquivalentFont()
```

Discussion

The default font is the same as that used to draw the title.

See Also

`setKeyEquivalentFont` (page 285)

`setKeyEquivalentFontAndSize` (page 286)

`setFont` (page 283)

keyEquivalentModifierMask

Returns the mask indicating the modifier keys that are applied to the receiver's key equivalent.

```
public int keyEquivalentModifierMask()
```

Discussion

The only mask bits relevant in button key-equivalent modifier masks are `NSEvent.ControlKeyMask`, `NSEvent.AlternateKeyMask`, and `NSEvent.CommandKeyMask` bits.

See Also

`setKeyEquivalentModifierMask` (page 286) keyEquivalent (page 279)

mouseEntered

Draws the receiver's border.

```
public void mouseEntered(NSEvent event)
```

Discussion

The *event* argument is the event object generated by the mouse movement. This method is called only when the cursor moves onto the receiver and showsBorderOnlyWhileMouseInside (page 288) returns true.

mouseExited

Erases the receiver's border.

```
public void mouseExited(NSEvent event)
```

Discussion

The *event* argument is the event object generated by the mouse movement. This method is called only when the cursor moves off the receiver and showsBorderOnlyWhileMouseInside (page 288) returns true.

performClick

Simulates the user clicking the receiver with the cursor.

```
public void performClick(Object sender)
```

Discussion

This method essentially highlights the button, sends the button's action message to the target object, and then unhighlights the button. If an exception is thrown while the target object is processing the action message, the button is unhighlighted before the exception is propagated out of performClick.

setAlternateImage

Sets the image that appears on the receiver when it's in its alternate state to *image* and, if necessary, redraws the contents of the receiver.

```
public void setAlternateImage(NSImage image)
```

Discussion

Note that some button types don't display an alternate image.

See Also

alternateImage (page 274)

setButtonType (page 283)

setImage (page 327) (NSCell)

setAlternateMnemonicLocation

Sets the character in the alternate title (the title displayed on the receiver when it's in its alternate state) that's to be marked as the "keyboard mnemonic."

```
public void setAlternateMnemonicLocation(int location)
```

Discussion

If you don't want the alternate title to have a keyboard mnemonic, specify a location of `NSArray.NotFound`. Mnemonics are not supported in Mac OS X.

The `setAlternateMnemonicLocation` method doesn't cause the button cell to be redisplayed.

See Also

`alternateMnemonicLocation` (page 274)

`setAlternateTitleWithMnemonic` (page 281)

setAlternateTitle

Sets the title that's displayed on the receiver when it's in its alternate state to *aString*.

```
public void setAlternateTitle(String aString)
```

Discussion

Note that some button types don't display an alternate title.

See Also

`alternateTitle` (page 275)

`setAlternateMnemonicLocation` (page 281)

`setAlternateTitleWithMnemonic` (page 281)

`setTitle` (page 287)

`setButtonType` (page 283)

`setFont` (page 283)

setAlternateTitleWithMnemonic

Sets the title that is displayed on the receiver when it's in its alternate state to *aString*, taking into account the fact that an embedded "&" character is not a literal but instead marks the alternate state's "keyboard mnemonic."

```
public void setAlternateTitleWithMnemonic(String aString)
```

Discussion

Mnemonics are not supported in Mac OS X.

If necessary, `setAlternateTitleWithMnemonic` redraws the button cell. Note that some button types don't display an alternate title.

See Also

`setAlternateMnemonicLocation` (page 281)

`setTitleWithMnemonic` (page 288)

setAttributedAlternateTitle

Sets the string that appears on the receiver when it's in its alternate state to the attributed string *aString*.

```
public void setAttributedAlternateTitle(NSAttributedString aString)
```

Discussion

Note that some button types don't display an alternate title.

See Also

`attributedAlternateTitle` (page 275)

`setAlternateMnemonicLocation` (page 281)

`setAlternateTitleWithMnemonic` (page 281)

`setAttributedTitle` (page 282)

`setButtonType` (page 283)

`setFont` (page 283)

setAttributedTitle

Sets the string that appears on the receiver when it's in its normal state to the attributed string *aString* and redraws the button.

```
public void setAttributedTitle(NSAttributedString aString)
```

Discussion

The title is always shown on buttons that don't use their alternate contents when highlighting or displaying their alternate state.

See Also

`attributedTitle` (page 275)

`setAttributedAlternateTitle` (page 282)

`setButtonType` (page 283)

`setFont` (page 283)

`setMnemonicLocation` (page 328) (NSCell)

setBackgroundColor

Sets the background color of the button.

```
public void setBackgroundColor(NSColor)
```

Discussion

The background color is used only when drawing borderless buttons.

Availability

Available in Mac OS X v10.4 and later.

See Also

`backgroundColor` (page 276)

setBezelStyle

Sets the appearance of the border, if the receiver has one.

```
public void setBezelStyle(int bezelStyle)
```

Discussion

`bezelStyle` must be one of the values specified in "Constants" (page 289).

A button uses shading to look like it's sticking out or pushed in. You can set the shading with `setGradientType` (page 284).

If the receiver is not bordered, the bezel style is ignored.

See Also

`bezelStyle` (page 276)

setButtonType

Sets how the receiver highlights while pressed and how it shows its state.

```
public void setButtonType(int aType)
```

Discussion

`setButtonType` redisplays the receiver before returning.

The types available are for the most common button types, which are also accessible in Interface Builder; you can configure different behavior with the `setHighlightsBy` (page 284) and `setShowsStateBy` (page 287) methods.

The `aType` argument can be one of the constants defined in "Constants" (page 289).

See Also

`setAlternateImage` (page 280)

`setImage` (page 327) (NSCell)

setFont

Sets the font used to display the title and alternate title to the font specified by `fontObj`.

```
public void setFont(NSFont fontObj)
```

Discussion

Does nothing if the receiver 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 font.

See Also

`setKeyEquivalentFont` (page 285)

`setKeyEquivalentFontAndSize` (page 286)

`font` (page 311) (NSCell)

setGradientType

Sets the type of gradient to use for the receiver.

```
public void setGradientType(int gradientType)
```

Discussion
If the receiver has no border, this method has no effect on its appearance. A concave gradient is darkest in the top-left corner; a convex gradient is darkest in the bottom-right corner. Weak versus strong is how much contrast exists between the colors used in opposite corners.

The *gradientType* argument can be one of the constants defined in "Constants" (page 289).

See Also
`gradientType` (page 277)

setHighlightsBy

Sets the way the receiver highlights itself while pressed.

```
public void setHighlightsBy(int aType)
```

Discussion
aType can be the logical OR of one or more of the cell masks described in NSCell's "Constants" (page 337) section.

If both `NSCell.ChangeGrayCellMask` and `NSCell.ChangeBackgroundCellMask` are specified, both are recorded, but which behavior is used depends on the button cell's image. If the button has no image, or if the image has no alpha (transparency) data, `NSCell.ChangeGrayCellMask` is used. If the image does have alpha data, `NSCell.ChangeBackgroundCellMask` is used; this arrangement allows the color swap of the background to show through the image's transparent pixels.

See Also
`highlightsBy` (page 277)
`setShowsStateBy` (page 287)

setImageDimsWhenDisabled

Sets whether the receiver's image appears "dim" when the button cell is disabled, depending on the Boolean value *flag*.

```
public void setImageDimsWhenDisabled(boolean flag)
```

Discussion
By default, all button types except `SwitchButton` and `RadioButton` do dim when disabled. When `SwitchButton`s and `RadioButton`s are disabled, only the associated text dims. The default setting for this condition is reasserted whenever you invoke `setButtonType` (page 283), so be sure to specify the button cell's type before you invoke `setImageDimsWhenDisabled`.

See Also
`imageDimsWhenDisabled` (page 278)

setImagePosition

Sets the position of the receiver's image relative to its title.

```
public void setImagePosition(int aPosition)
```

Discussion

See NSCell's "Constants" (page 337) section for a listing of possible values for *aPosition*.

See Also

`imagePosition` (page 278)

setKeyEquivalent

Sets the key equivalent character of the receiver, specified by *aKeyEquivalent*, and redraws the receiver's inside if it displays a key equivalent instead of an image.

```
public void setKeyEquivalent(String aKeyEquivalent)
```

Discussion

The key equivalent isn't displayed if the image position is set to `NSCell.NoImage`, `NSCell.ImageOnly`, or `NSCell.ImageOverlaps`; 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 `null`, then set the key equivalent, then set the image position.

See Also

`keyEquivalent` (page 279)

`setAlternateImage` (page 280)

`setImagePosition` (page 285)

`setKeyEquivalentFont` (page 285)

`setImage` (page 327) (NSCell)

setKeyEquivalentFont

Sets the font used to draw the key equivalent to the font specified by *fontObj* and redisplays the receiver if necessary.

```
public void setKeyEquivalentFont(NSFont fontObj)
```

Discussion

Does nothing if the receiver doesn't have a key equivalent associated with it. The default font is the same as that used to draw the title.

See Also

`keyEquivalentFont` (page 279)

`setFont` (page 283)

setKeyEquivalentFontAndSize

Sets by name and size the font used to draw the key equivalent and redisplays the receiver if necessary.

```
public void setKeyEquivalentFontAndSize(String fontName, float fontSize)
```

Discussion

Does nothing if the receiver doesn't have a key equivalent associated with it. The default font is the same as that used to draw the title.

See Also

`keyEquivalentFont` (page 279)

`setFont` (page 283)

setKeyEquivalentModifierMask

Sets the mask indicating the modifier keys to be applied to the receiver's key equivalent using the value of *mask*.

```
public void setKeyEquivalentModifierMask(int mask)
```

Discussion

The only mask bits relevant in button key-equivalent modifier masks are `NSEvent.ControlKeyMask`, `NSEvent.AlternateKeyMask`, and `NSEvent.CommandKeyMask` bits.

See Also

`keyEquivalentModifierMask` (page 279)

`setKeyEquivalent` (page 285)

setPeriodicDelayAndInterval

Sets the message delay and interval for the receiver.

```
public void setPeriodicDelayAndInterval(float delay, float interval)
```

Discussion

These two values are used if the receiver is configured (by a `setContinuous` (page 322) 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 *delay* and *interval* is 60.0 seconds; if a larger value is supplied, it's ignored, and 60.0 seconds is used.

See Also

`setContinuous` (page 322) (NSCell)

setShowsBorderOnlyWhileMouseInside

Sets whether the receiver's border is displayed only when the cursor is over the button.

```
public void setShowsBorderOnlyWhileMouseInside(boolean show)
```

Discussion

If *show* is `true`, the border is displayed only when the cursor is within the receiver's border and the button is active. If *show* is `false`, the receiver's border continues to be displayed when the cursor is outside button's bounds.

See Also

`showsBorderOnlyWhileMouseInside` (page 288)

setShowsStateBy

Sets the way the receiver indicates its alternate state.

```
public void setShowsStateBy(int aType)
```

Discussion

aType should be the logical OR of one or more of the cell masks described in NSCell's "Constants" (page 337) section.

If both `NSCell.ChangeGrayCellMask` and `NSCell.ChangeBackgroundCellMask` are specified, both are recorded, but the actual behavior depends on the button cell's image. If the button has no image, or if the image has no alpha (transparency) data, `NSCell.ChangeGrayCellMask` is used. If the image exists and has alpha data, `NSCell.ChangeBackgroundCellMask` is used; this arrangement allows the color swap of the background to show through the image's transparent pixels.

See Also

`setHighlightsBy` (page 284)

`showsStateBy` (page 289)

setSound

Sets the sound that's played when the user presses the receiver to *aSound*.

```
public void setSound(Object aSound)
```

Discussion

The sound is played during a mouse-down event, such as `NSEvent.LeftMouseDown`.

See Also

`sound` (page 289)

setTitle

Sets the title displayed by the receiver when in its normal state to *aString* and, if necessary, redraws the receiver's contents.

```
public void setTitle(String aString)
```

Discussion

This title is always shown on buttons that don't use their alternate contents when highlighting or displaying their alternate state.

See Also

`title` (page 289)
`setAlternateTitle` (page 281)
`setButtonType` (page 283)
`setFont` (page 283)
`setTitleWithMnemonic` (page 288)

setTitleWithMnemonic

Sets the title displayed on the receiver when it's in its normal state to *aString*, taking into account the fact that an embedded "&" character is not a literal but instead marks the normal state's "keyboard mnemonic."

```
public void setTitleWithMnemonic(String aString)
```

Discussion

If necessary, `setTitleWithMnemonic` redraws the button cell. This title is always shown on buttons that don't use their alternate contents when highlighting or displaying their alternate state. Mnemonics are not supported in Mac OS X.

See Also

`setAlternateTitleWithMnemonic` (page 281)
`setTitleWithMnemonic` (page 332) (NSCell)
`setMnemonicLocation` (page 328) (NSCell)

setTransparent

```
public void setTransparent(boolean flag)
```

Discussion

Sets whether the receiver is transparent, depending on the Boolean value *flag*, and redraws the receiver if necessary. 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` (page 279)
`isOpaque` (page 278)

showsBorderOnlyWhileMouseInside

Returns `true` if the receiver's border is displayed only when the cursor is over the button and the button is active.

```
public boolean showsBorderOnlyWhileMouseInside()
```

Discussion

By default, this method returns `false`.

See Also

`setShowsBorderOnlyWhileMouseInside` (page 286)

showsStateBy

Returns the logical OR of flags that indicate the way the receiver shows its alternate state.

```
public int showsStateBy()
```

Discussion
See NSCell's "Constants" (page 337) section for the list of flags.

See Also
`highlightsBy` (page 277)
`setShowsStateBy` (page 287)

sound

Returns the sound that's played when the user presses the button.

```
public Object sound()
```

See Also
`setSound` (page 287)

title

Returns the title displayed on the receiver when it's in its normal state (this title is always displayed if the button doesn't use its alternate contents for highlighting or displaying the alternate state).

```
public String title()
```

Discussion
Returns the empty string if the button doesn't display a title. By default, a button's title is "Button."

See Also
`setTitle` (page 287)
`alternateTitle` (page 275)
`setButtonType` (page 283)
`mnemonic` (page 316) (NSCell)
`mnemonicLocation` (page 317) (NSCell)

Constants

These are the bezel styles used by `bezelStyle` (page 276) and `setBezelStyle` (page 283):

Constant	Description
`RoundedBezelStyle`	A rounded rectangle button, designed for text.
`RegularSquareBezelStyle`	A rectangular button with a 2 pixel border, designed for icons.

Constant	Description
ThickSquareBezelStyle	A rectangular button with a 3 pixel border, designed for icons.
ThickerSquareBezelStyle	A rectangular button with a 4 pixel border, designed for icons.
DisclosureBezelStyle	A bezel style for use with a disclosure triangle. To create the disclosure triangle, set the button bezel style to DisclosureBezelStyle and the button type to OnOffButton. Available for Mac OS X v10.3 or later.
ShadowlessSquareBezelStyle	Similar to RegularSquareBezelStyle, but has no shadow so you can abut the cells without overlapping shadows. This style would be used in a tool palette, for example.
CircularBezelStyle	A round button with room for a small icon or a single character. This style has both regular and small variants, but the large variant is available only in gray at this time.
TexturedSquareBezelStyle	A bezel style appropriate for use with textured (that is, metal) windows. Available for Mac OS X v10.3 and later.
HelpButtonBezelStyle	A round button with a question mark providing the standard help button look. Available for Mac OS X v10.3 and later.
SmallSquareBezelStyle	A simple square bezel style. Buttons using this style can be scaled to any size. Available for Mac OS X v10.4 and later.
TexturedRoundedBezelStyle	A textured (metal) bezel style similar in appearance to the Finder's action (gear) button. The height of this button is fixed. (Available for Mac OS X v10.4 and later.)

These are the button types that can be specified using setButtonType (page 283)

Button Type	Description
MomentaryChangeButton	While the button is held down, the alternate image and alternate title are displayed. The normal image and title are displayed when the button isn't pressed. This option is called "Momentary Change" in Interface Builder's Button Inspector.
PushOnPushOffButton	The first click both highlights and causes the button to be "pushed in" if the button is bordered. A second click returns it to its normal state. This option is called "Push On/Push Off" in Interface Builder's Button Inspector.
OnOffButton	The first click highlights the button. A second click returns it to the normal (unhighlighted) state. This option is called "On/Off" in Interface Builder's Button Inspector.

Button Type	Description
`ToggleButton`	After the first click, the button displays its alternate image or title. A second click returns the button to its normal state. This option is called "Toggle" in Interface Builder's Button Inspector.
`SwitchButton`	This style is a variant of `ToggleButton` that has no border. This type of button is available as a separate palette item in Interface Builder.
`RadioButton`	Similar to `SwitchButton`. This type of button is available as a separate palette item in Interface Builder.
`MomentaryLight`	While the button is held down it's shown as "lit." 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.
`MomentaryPushButton`	While the button is held down it's shown 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.

NSCachedImageRep

Inherits from	NSImageRep : NSObject
Implements	NSCoding (NSImageRep)
Package:	com.apple.cocoa.application
Companion guide	Drawing and Images

Overview

NSCachedImageRep defines an object that stores its source data as a rendered image in a window, typically a window that stays offscreen. The only data available for reproducing the image is the image itself. Thus an NSCachedImageRep differs from the other kinds of NSImageReps 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.

Tasks

Constructors

`NSCachedImageRep` (page 294)
Throws an exception. Use one of the other constructors instead.

Getting the Representation

`rect` (page 294)
Returns the rectangle where the receiver is cached.

`window` (page 294)
Returns the window where the receiver is cached.

Constructors

NSCachedImageRep

Throws an exception. Use one of the other constructors instead.

```
public NSCachedImageRep()
```

Creates a new NSCachedImageRep for an image that will be rendered within the *aRect* rectangle in the window *aWindow*.

```
public NSCachedImageRep(NSWindow aWindow, NSRect aRect)
```

Discussion
The rectangle is specified in *aWindow*'s base coordinate system. 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.

Creates a new NSCachedImageRep for an image of the specified *size* and *depth*.

```
public NSCachedImageRep(NSSize size, int depth, boolean flag, boolean alpha)
```

Discussion
The *flag* argument indicates 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), *alpha* should be set according to whether the image will have a channel for transparency information.

Instance Methods

rect

Returns the rectangle where the receiver is cached.

```
public NSRect rect()
```

See Also
`size` (page 792) (NSImageRep)

window

Returns the window where the receiver is cached.

```
public NSWindow window()
```

NSCell

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.application
Companion guide	Control and Cell Programming Topics for Cocoa

Overview

The NSCell class provides a mechanism for displaying text or images in an NSView without the overhead of a full NSView subclass. It's used heavily by most of the NSControl classes to implement their internal workings.

Tasks

Constructors

`NSCell` (page 303)
Creates an empty NSCell.

Setting and Getting Cell Values

`setObjectValue` (page 328)
Sets the receiver's object value to *`object`*.

`objectValue` (page 317)
Returns the receiver's value as an object if a valid object has been associated with the receiver; otherwise, returns `null`.

`hasValidObjectValue` (page 312)
Returns whether the object associated with the receiver has a valid object value.

`setIntValue` (page 327)
Sets the value of the receiver to an object *`anInt`*, representing an integer value.

`intValue` (page 313)
Returns the receiver's value as an `int`.

`setStringValue` (page 331)
Sets the value of the receiver to *`aString`*.

`stringValue` (page 334)
Returns the receiver's value as a String as converted by the receiver's formatter, if one exists.

`setDoubleValue` (page 323)
Sets the value of the receiver to an object *aDouble*, representing a double value.

`doubleValue` (page 309)
Returns the receiver's value as a `double`.

`setFloatValue` (page 325)
Sets the value of the receiver to an object *aFloat*, representing a float value.

`floatValue` (page 311)
Returns the receiver's value as a `float`.

Setting and Getting Cell Attributes

`setCellAttribute` (page 322)
Sets a cell attribute identified by *aParameter*—such as the receiver's state and whether it's disabled, editable, or highlighted—to *value*.

`cellAttribute` (page 307)
Depending on *aParameter*, returns a setting for a cell attribute, such as the receiver's state, and whether it's disabled, editable, or highlighted.

`setType` (page 332)
If the type of the receiver is different from *aType*, sets it to *aType*, which must be one of `TextCellType`, `ImageCellType`, or `NullCellType`.

`type` (page 337)
Returns the type of the receiver, one of `TextCellType`, `ImageCellType`, or `NullCellType`.

`setEnabled` (page 324)
Sets whether the receiver is enabled or disabled, depending on the Boolean value *flag*.

`isEnabled` (page 314)
Returns whether the receiver responds to mouse events.

`setBezeled` (page 321)
Sets whether the receiver draws itself with a bezeled border, depending on the Boolean value *flag*.

`isBezeled` (page 314)
Returns whether the receiver has a bezeled border.

`setBordered` (page 321)
Sets whether the receiver draws itself outlined with a plain border, depending on the Boolean value *flag*.

`isBordered` (page 314)
Returns whether the receiver has a plain border.

`isOpaque` (page 315)
Returns whether the receiver is opaque (nontransparent).

`allowsUndo` (page 306)
Returns `true` if undo operations are handled directly by the cell.

`setAllowsUndo` (page 321)
If *allowsUndo* is `true`, the receiver assumes responsibility for undo operations within the cell.

Setting the State

allowsMixedState (page 306)
Returns true if the receiver has three states: on, off, and mixed.

nextState (page 317)
Returns the receiver's next state.

setAllowsMixedState (page 320)

setNextState (page 328)
Sets the receiver to its next state.

setState (page 330)
Sets the receiver's state to *value*, which can be OnState, OffState, or MixedState.

state (page 333)
Returns the receiver's state.

Modifying Textual Attributes of Cells

setEditable (page 323)
Sets whether the user can edit the receiver's text, depending on the Boolean value *flag*.

isEditable (page 314)
Returns whether the receiver is editable.

setSelectable (page 329)
Sets whether text in the receiver can be selected, depending on the Boolean value *flag*.

isSelectable (page 315)
Returns whether the text of the receiver can be selected.

setScrollable (page 329)
Sets whether excess text in the receiver is scrolled past the cell's bounds.

isScrollable (page 315)
Returns whether the receiver scrolls typed text that exceeds the cell's bounds.

setAlignment (page 320)
Sets the alignment of text in the receiver.

alignment (page 305)
Returns the alignment of text in the receiver: NSText.LeftTextAlignment, NSText.RightTextAlignment, NSText.CenterTextAlignment, NSText.JustifiedTextAlignment, or NSText.NaturalTextAlignment.

setFont (page 326)
Sets the font to be used when the receiver displays text.

font (page 311)
Returns the font used to display text in the receiver or null if the receiver is not a text-type cell.

lineBreakMode (page 316)
Returns the line break mode currently used when drawing text.

setLineBreakMode (page 327)
Sets the line break mode used when drawing text to *mode*.

`setWraps` (page 333)
Sets whether text in the receiver wraps when its length exceeds the frame of the cell.

`wraps` (page 337)
Returns whether text of the receiver wraps when it exceeds the borders of the cell.

`setAttributedStringValue` (page 321)
Sets the value of the receiver to the attributed string *attribStr*.

`attributedStringValue` (page 306)
Returns the value of the receiver as an attributed string (that is, a string with attributes), using the receiver's formatter object (if one exists) to create the attributed string.

`setAllowsEditingTextAttributes` (page 320)
Sets whether the textual attributes of the receiver can be modified by the user.

`allowsEditingTextAttributes` (page 305)
Returns whether the receiver allows user editing of textual attributes.

`setImportsGraphics` (page 327)
Sets whether the receiver can import images into its text (that is, whether it supports RTFD text).

`importsGraphics` (page 313)
Returns whether the text of the receiver (if a text-type cell) is of Rich Text Format (RTF) and thus can import graphics.

`setUpFieldEditorAttributes` (page 332)
Sets textual and background attributes of *textObj*, depending on certain attributes.

`setTitle` (page 332)
Sets the title of the receiver to *aString*.

Setting the Target and Action

`setAction` (page 319)

`action` (page 305)
Implemented by NSActionCell and its subclasses to return the selector of the receiver's action method. The default implementation returns a null selector.

`setTarget` (page 331)
Implemented by NSActionCell to set the receiver's target object receiving the action message to *anObject*.

`target` (page 336)

`setContinuous` (page 322)
Sets whether the receiver continuously sends its action message to its target while it tracks the mouse, depending on the Boolean value *flag*.

`isContinuous` (page 314)
Returns whether the receiver sends its action message continuously on mouse down.

`setEventMaskForSendingAction` (page 324)
Sets the conditions on which the receiver sends action messages to its target and returns a bit mask with which to detect the previous settings.

Setting and Getting an Image

`setImage` (page 327)
Sets the image to be displayed by the receiver.

`image` (page 313)
Returns the image displayed by the receiver or `null` if the receiver is not an image-type cell.

Assigning a Tag

`setTag` (page 331)
Implemented by NSActionCell to set the receiver’s tag integer to *anInt*.

`tag` (page 334)
Implemented by NSActionCell to return the receiver’s tag integer.

Formatting and Validating Data

`setFormatter` (page 326)
Sets the formatter object used to format the textual representation of the receiver’s object value and to validate cell input and convert it to that object value.

`formatter` (page 311)
Returns the formatter object (a kind of NSFormatter) associated with the receiver.

`setEntryType` (page 324)
Sets how numeric data is formatted in the receiver and places restrictions on acceptable input.

`entryType` (page 310)
Returns the type of data the user can type into the receiver.

`isEntryAcceptable` (page 314)
Returns whether a string representing a numeric or date value (*aString*) is formatted in a way suitable to the entry type.

`setFloatingPointFormat` (page 325)
Sets whether floating-point numbers are autoranged in the receiver and sets the sizes of the fields to the left and right of the decimal point.

Managing Menus for Cells

`defaultMenu` (page 304)
Returns the default menu for instances of the receiver.

`setMenu` (page 328)
Associates a menu with the cell that has commands contextually related to the receiver.

`menu` (page 316)
Returns the menu with commands contextually related to the cell or `null` if no menu is associated.

`menuForEvent` (page 316)
Returns the NSMenu associated with the receiver through the `setMenu` (page 328) method and related to *anEvent* when the cursor is detected within *cellFrame*.

Comparing Cells

`compare` (page 308)
Compares the string values of the receiver and *otherCell* (which must be a kind of NSCell), disregarding case.

Making Cells Respond to Keyboard Events

`acceptsFirstResponder` (page 305)
Returns whether or not the receiver will accept first responder status.

`setShowsFirstResponder` (page 330)
Sets whether the receiver draws some indication of its first responder status.

`showsFirstResponder` (page 333)
Returns whether the receiver should draw some indication when it assumes first responder status.

`setTitleWithMnemonic` (page 332)
Sets the title of the receiver to *aString* with a character denoted as an access key.

`mnemonic` (page 316)
Returns the character in the receiver's title that appears underlined for use as a mnemonic.

`refusesFirstResponder` (page 318)
Returns `true` if the receiver can never become the first responder.

`setMnemonicLocation` (page 328)
Sets the character of the receiver's title identified by *location* to be underlined.

`setRefusesFirstResponder` (page 329)
Sets whether the receiver can become the first responder.

`mnemonicLocation` (page 317)
Returns the position of the underlined character in the receiver's title used as a mnemonic.

`performClick` (page 318)
Can be used to simulate a single mouse click on the receiver.

Deriving Values from Other Cells

`takeObjectValue` (page 335)
Sets the receiver's own value as an object using the object value of *sender*.

`takeIntValue` (page 335)
Sets the receiver's own value as an `int` using the `int` value of *sender*.

`takeStringValue` (page 335)
Sets the receiver's own value as a string object using the String value of *sender*.

`takeDoubleValue` (page 335)
Sets the receiver's own value as a `double` using the `double` value of *sender*.

`takeFloatValue` (page 335)
Sets the receiver's own value as a `float` using the `float` value of *sender*.

Representing an Object with a Cell

`setRepresentedObject` (page 329)
: Sets the object represented by the receiver—f

`representedObject` (page 318)
: Returns the object the receiver represents.

Tracking the Mouse

`trackMouse` (page 336)
: Invoked by an NSControl to initiate the tracking behavior of one of its NSCells.

`startTrackingMouse` (page 333)

`continueTrackingMouse` (page 308)
: Returns whether mouse tracking should continue in the receiving cell based on *`lastPoint`* and *`currentPoint`* within *`controlView`*.

`stopTrackingMouse` (page 334)

`mouseDownFlags` (page 317)
: Returns the modifier flags for the last (left) mouse-down event, or 0 if tracking hasn't occurred yet for the cell or no modifier keys accompanied the mouse-down event.

`prefersTrackingUntilMouseUp` (page 304)
: The default implementation returns `false`, so tracking stops when the cursor leaves the NSCell; subclasses may override.

Managing the Cursor

`resetCursorRect` (page 319)
: Sets the receiver to show the I-beam cursor within *`cellFrame`* while it tracks the mouse.

Managing Cell Messaging

`interval` (page 313)
: Returns the amount of time that the cell pauses between messages when it's sending action messages continuously to target objects.

`periodicDelay` (page 318)
: Returns the amount of time that the cell pauses before sending its first message when it's sending action messages continuously to target objects.

Handling Keyboard Alternatives

`keyEquivalent` (page 315)
: Implemented by subclasses to return a key equivalent to clicking the cell.

Managing Focus Rings

defaultFocusRingType (page 304)
Returns the default type of focus ring.

setFocusRingType (page 325)
Sets the type of focus ring to be used.

focusRingType (page 311)
Returns the type of focus ring currently set.

Determining Cell Sizes

calcDrawInfo (page 306)
Implemented by subclasses to recalculate drawing sizes with reference to *aRect*.

cellSize (page 307)
Returns the minimum size needed to display the receiver, taking account of the size of the image or text within a certain offset determined by the border type.

cellSizeForBounds (page 307)
Returns the minimum size needed to display the receiver, taking account of the size of the image or text within an offset determined by the border type.

drawingRectForBounds (page 309)
Returns the rectangle within which the receiver draws itself; this rectangle is slightly inset from *theRect* on all sides to take the border into account.

imageRectForBounds (page 313)
Returns the rectangle the receiver's image is drawn in, which is slightly offset from *theRect*.

titleRectForBounds (page 336)

controlSize (page 308)
Returns the size of the receiver.

setControlSize (page 322)
Sets the size of the receiver.

Drawing and Highlighting Cells

drawWithFrameInView (page 310)
Draws the receiver's regular or bezeled border (if those attributes are set) and then draws the interior of the cell by invoking drawInteriorWithFrameInView (page 309).

highlightColorWithFrameInView (page 312)
Returns the color to use when drawing the receiver's selection highlight in *cellFrame*.

drawInteriorWithFrameInView (page 309)
Draws the "inside" of the receiver—including the image or text within the receiver's frame in *controlView* (usually the cell's NSControl) but excluding the border.

controlView (page 309)
Implemented by subclasses to return the NSView last drawn in (normally an NSControl).

setControlView (page 323)
Sets the receiver's control view.

highlightWithFrameInView (page 312)
If the receiver's highlight status is different from *flag*, sets that status to *flag* and, if *flag* is true, highlights the rectangle *cellFrame* in the NSControl (*controlView*).

setHighlighted (page 326)
Sets whether the receiver has a highlighted appearance, depending on the Boolean value *flag*.

isHighlighted (page 315)
Returns whether the receiver is highlighted.

setControlTint (page 323)
Sets the receiver's control tint.

controlTint (page 308)
Returns the receiver's control tint.

Editing and Selecting Cell Text

editWithFrameInView (page 310)
Begins editing of the receiver's text using the field editor *textObj*.

selectAndEditWithFrameInView (page 319)
Uses the field editor *textObj* to select text in a range marked by *selStart* and *selLength*, which will be highlighted and selected as though the user had dragged the cursor over it.

sendsActionOnEndEditing (page 319)
Returns whether the receiver's NSControl object sends its action message whenever the user finishes editing the cell's text.

setSendsActionOnEndEditing (page 330)
Sets whether the receiver's NSControl object sends its action message whenever the user finishes editing the cell's text.

endEditing (page 310)
Ends any editing of text, using the field editor *textObj*, occurring in the receiver begun with editWithFrameInView (page 310) and selectAndEditWithFrameInView (page 319).

Constructors

NSCell

Creates an empty NSCell.

```
public NSCell()
```

Creates an NSCell initialized with *aString* and set to have the cell's default menu.

```
public NSCell(String aString)
```

Discussion
If no field editor (a shared NSText object) has been created for all NSCells, one is created.

Creates an NSCell initialized with *anImage* and set to have the cell's default menu.

```
public NSCell(NSImage anImage)
```

Discussion
If *anImage* is `null`, no image is set.

Static Methods

defaultFocusRingType

Returns the default type of focus ring.

```
public static int defaultFocusRingType()
```

Discussion
Possible values are listed in the "Constants" (page 727) section of NSGraphics.

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

defaultMenu

Returns the default menu for instances of the receiver.

```
public static NSMenu defaultMenu()
```

Discussion
The default implementation returns `null`.

See Also
`menu` (page 316)
`setMenu` (page 328)

prefersTrackingUntilMouseUp

The default implementation returns `false`, so tracking stops when the cursor leaves the NSCell; subclasses may override.

```
public static boolean prefersTrackingUntilMouseUp()
```

See Also
`trackMouse` (page 336)

Instance Methods

acceptsFirstResponder

Returns whether or not the receiver will accept first responder status.

```
public boolean acceptsFirstResponder()
```

Discussion

The default implementation returns `true` if the receiver is enabled, and `refusesFirstResponder` (page 318) returns `false`; subclasses can override.

See Also

`performClick` (page 318)
`setShowsFirstResponder` (page 330)
`setTitleWithMnemonic` (page 332)

action

Implemented by NSActionCell and its subclasses to return the selector of the receiver's action method. The default implementation returns a null selector.

```
public NSSelector action()
```

See Also

`setAction` (page 319)
`setTarget` (page 331)
`target` (page 336)

alignment

Returns the alignment of text in the receiver: `NSText.LeftTextAlignment`, `NSText.RightTextAlignment`, `NSText.CenterTextAlignment`, `NSText.JustifiedTextAlignment`, or `NSText.NaturalTextAlignment`.

```
public int alignment()
```

See Also

`setAlignment` (page 320)

allowsEditingTextAttributes

Returns whether the receiver allows user editing of textual attributes.

```
public boolean allowsEditingTextAttributes()
```

See Also

`setAllowsEditingTextAttributes` (page 320)

allowsMixedState

Returns `true` if the receiver has three states: on, off, and mixed.

```
public boolean allowsMixedState()
```

Discussion
Returns `false` if the receiver has two states: on and off.

See Also
`nextState` (page 317)
`setAllowsMixedState` (page 320)
`setNextState` (page 328)

allowsUndo

Returns `true` if undo operations are handled directly by the cell.

```
public boolean allowsUndo()
```

Discussion
By default, the NSTextFieldCell class uses this feature to handle undo operations for edited text. Other controls set a value that is appropriate for their implementation.

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

See Also
`setAllowsUndo` (page 321)

attributedStringValue

Returns the value of the receiver as an attributed string (that is, a string with attributes), using the receiver's formatter object (if one exists) to create the attributed string.

```
public NSAttributedString attributedStringValue()
```

Discussion
The textual attributes are the default paragraph style, the receiver's font and alignment, and whether the receiver is enabled and scrollable.

For Mac OS X v10.3 and later: If you use a class that responds to the selector `attributedStringValue` for the object value of a cell, then the cell will use that method to fetch the string to draw rather than using `stringValue` (page 334).

See Also
`setAttributedStringValue` (page 321)

calcDrawInfo

Implemented by subclasses to recalculate drawing sizes with reference to *`aRect`*.

```
public void calcDrawInfo(NSRect aRect)
```

Discussion

Objects (such as NSControls) that manage NSCells generally maintain a flag that informs them if any of their cells has been modified in such a way that the location or size of the cell should be recomputed. If so, NSControl's `calcSize` (page 449) method is automatically invoked prior to the display of the NSCell, and that method invokes the NSCell's `calcDrawInfo` method. The default implementation does nothing.

See Also

`cellSize` (page 307)

`drawingRectForBounds` (page 309)

cellAttribute

Depending on *`aParameter`*, returns a setting for a cell attribute, such as the receiver's state, and whether it's disabled, editable, or highlighted.

```
public int cellAttribute(int aParameter)
```

See Also

`setCellAttribute` (page 322)

cellSize

Returns the minimum size needed to display the receiver, taking account of the size of the image or text within a certain offset determined by the border type.

```
public NSSize cellSize()
```

Discussion

If the receiver is of neither image nor text type, a size of 10000.0 x 10000.0 is returned; if the receiver is of image type, and no image has been set, `NSSize.ZeroSize` is returned.

See Also

`drawingRectForBounds` (page 309)

cellSizeForBounds

Returns the minimum size needed to display the receiver, taking account of the size of the image or text within an offset determined by the border type.

```
public NSSize cellSizeForBounds(NSRect aRect)
```

Discussion

If the receiver is of text type, the text is resized to fit within *`aRect`* (as much as *`aRect`* is within the bounds of the cell). If the receiver is of neither image nor text type, the size of *`aRect`* parameter is returned; if the receiver is of image type, and no image has been set, a size with zero width and height is returned.

See Also

`drawingRectForBounds` (page 309)

compare

Compares the string values of the receiver and *otherCell* (which must be a kind of NSCell), disregarding case.

```
public int compare(Object otherCell)
```

Discussion
Throws BadComparisonException if *otherCell* is not of the NSCell class.

continueTrackingMouse

Returns whether mouse tracking should continue in the receiving cell based on *lastPoint* and *currentPoint* within *controlView*.

```
public boolean continueTrackingMouse(NSPoint lastPoint, NSPoint currentPoint, NSView
      controlView)
```

Discussion
currentPoint is the current location of the cursor while *lastPoint* is either the initial location of the cursor or the previous *currentPoint*. This method is invoked in trackMouse (page 336). The default implementation returns true if the cell is set to continuously send action messages to its target when the mouse button is down or the mouse is being dragged. Subclasses can override this method to provide more sophisticated tracking behavior.

See Also
startTrackingMouse (page 333)
stopTrackingMouse (page 334)

controlSize

Returns the size of the receiver.

```
public int controlSize()
```

Discussion
Valid return values are described in "Constants" (page 337).

See Also
setControlSize (page 322)

controlTint

Returns the receiver's control tint.

```
public int controlTint()
```

Discussion
Valid return values are described in "Constants" (page 337).

See Also
setControlTint (page 323)

controlView

Implemented by subclasses to return the NSView last drawn in (normally an NSControl).

```
public NSView controlView()
```

Discussion

The default implementation returns `null`.

See Also

`drawWithFrameInView` (page 310)

`setControlView` (page 323)

doubleValue

Returns the receiver's value as a `double`.

```
public double doubleValue()
```

Discussion

If the receiver is not a text-type cell or the cell value is not scannable, the method returns 0.

drawingRectForBounds

Returns the rectangle within which the receiver draws itself; this rectangle is slightly inset from *theRect* on all sides to take the border into account.

```
public NSRect drawingRectForBounds(NSRect theRect)
```

See Also

`calcSize` (page 449) (NSControl)

drawInteriorWithFrameInView

Draws the "inside" of the receiver—including the image or text within the receiver's frame in *controlView* (usually the cell's NSControl) but excluding the border.

```
public void drawInteriorWithFrameInView(NSRect cellFrame, NSView controlView)
```

Discussion

cellFrame is the frame of the NSCell or, in some cases, a portion of it. Text-type NSCells display their contents in a rectangle slightly inset from *cellFrame* using a global NSText object. Image-type NSCells display their contents centered within *cellFrame*. If the proper attributes are set, it also displays the dotted-line rectangle to indicate first responder and highlights the cell. This method is invoked from NSControl's `drawCellInside` (page 450) to visually update what the NSCell displays when its contents change. This drawing is minimal and becomes more complex in objects such as NSButtonCell and NSSliderCell.

This method draws the cell in the currently focused view, which can be different from the *controlView* passed in. Taking advantage of this is not recommended.

Subclasses often override this method to provide more sophisticated drawing of cell contents. Because `drawWithFrameInView` (page 310) invokes `drawInteriorWithFrameInView` after it draws the NSCell's border, don't invoke `drawWithFrameInView` (page 310) in your override implementation.

See Also

`isHighlighted` (page 315)

`setShowsFirstResponder` (page 330)

drawWithFrameInView

Draws the receiver's regular or bezeled border (if those attributes are set) and then draws the interior of the cell by invoking `drawInteriorWithFrameInView` (page 309).

```
public void drawWithFrameInView(NSRect cellFrame, NSView controlView)
```

Discussion

This method draws the cell in the currently focused view, which can be different from the *controlView* passed in. Taking advantage of this is not recommended.

editWithFrameInView

Begins editing of the receiver's text using the field editor *textObj*.

```
public void editWithFrameInView(NSRect aRect, NSView controlView, NSText textObj,
     Object anObject, NSEvent theEvent)
```

Discussion

This method is usually invoked in response to a mouse-down event. *aRect* must be the rectangle used for displaying the NSCell. *theEvent* is the `NSEvent.LeftMouseDown` event. *anObject* is made the delegate of *textObj* and so will receive various NSText delegation and notification messages.

If the receiver isn't a text-type NSCell, no editing is performed. Otherwise, *textObj* is sized to *aRect* and its superview is set to *controlView*, so it exactly covers the NSCell. Then it's activated and editing begins. It's the responsibility of the delegate to end editing when responding to `textShouldEndEditing`. In doing this, it should remove any data from *textObj*.

See Also

`endEditing` (page 310)

`selectAndEditWithFrameInView` (page 319)

endEditing

Ends any editing of text, using the field editor *textObj*, occurring in the receiver begun with `editWithFrameInView` (page 310) and `selectAndEditWithFrameInView` (page 319).

```
public void endEditing(NSText textObj)
```

entryType

Returns the type of data the user can type into the receiver.

```
public int entryType()
```

Discussion

If the receiver is not a text-type cell, or if no type has been set, `AnyType` is returned. See "Constants" (page 337) for a list of type constants.

See Also

`isEntryAcceptable` (page 314)

floatValue

Returns the receiver's value as a `float`.

```
public float floatValue()
```

Discussion

If the receiver is not a text-type cell or the cell value is not scannable, the method returns 0.

focusRingType

Returns the type of focus ring currently set.

```
public int focusRingType()
```

Discussion

Possible values are listed in the "Constants" (page 727) section of NSGraphics You can disable a view's focus ring drawing by overriding this method so it always returns `NSGraphics.FocusRingTypeNone`, or by calling `setFocusRingType` (page 325) with `NSGraphics.FocusRingTypeNone`. You should only disable a view from drawing its focus ring if you want to draw your own focus ring, or if there isn't sufficient space to display a focus ring in the default location.

Availability

Available in Mac OS X v10.3 and later.

See Also

`setFocusRingType` (page 325)

`defaultFocusRingType` (page 304)

font

Returns the font used to display text in the receiver or `null` if the receiver is not a text-type cell.

```
public NSFont font()
```

See Also

`setFont` (page 326)

formatter

Returns the formatter object (a kind of NSFormatter) associated with the receiver.

```
public NSFormatter formatter()
```

Discussion

This object handles translation of the receiver's contents between its onscreen representation and its object value.

See Also

`setFormatter` (page 326)

hasValidObjectValue

Returns whether the object associated with the receiver has a valid object value.

```
public boolean hasValidObjectValue()
```

Discussion

A valid object value is one that the receiver's formatter can "understand." Objects are always assumed to be valid unless they are rejected by the formatter. Invalid objects can still be accepted by the delegate of the receiver's NSControl (in `controlDidFailToFormatStringErrorDescription` (page 463)).

See Also

`objectValue` (page 317)

`setObjectValue` (page 328)

highlightColorWithFrameInView

Returns the color to use when drawing the receiver's selection highlight in *cellFrame*.

```
public NSColor highlightColorWithFrameInView(NSRect cellFrame, NSView controlView)
```

Discussion

You should not assume that a cell would necessarily want to draw itself with the value returned from `selectedControlColor` (page 366). A cell may wish to draw with different a selection highlight color depending on such things as the key state of its *controlView*.

highlightWithFrameInView

If the receiver's highlight status is different from *flag*, sets that status to *flag* and, if *flag* is `true`, highlights the rectangle *cellFrame* in the NSControl (*controlView*).

```
public void highlightWithFrameInView(boolean flag, NSRect cellFrame, NSView
    controlView)
```

Discussion

Note that NSCell's highlighting does not appear when highlighted cells are printed (although instances of NSTextFieldCell, NSButtonCell, and others can print themselves highlighted). Generally, you cannot depend on highlighting being printed because implementations of this method may choose (or not choose) to use transparency.

See Also

`drawWithFrameInView` (page 310)

`isHighlighted` (page 315)

image

Returns the image displayed by the receiver or `null` if the receiver is not an image-type cell.

```
public NSImage image()
```

See Also

`setImage` (page 327)

imageRectForBounds

Returns the rectangle the receiver's image is drawn in, which is slightly offset from *`theRect`*.

```
public NSRect imageRectForBounds(NSRect theRect)
```

See Also

`cellSizeForBounds` (page 307)

`drawingRectForBounds` (page 309)

importsGraphics

Returns whether the text of the receiver (if a text-type cell) is of Rich Text Format (RTF) and thus can import graphics.

```
public boolean importsGraphics()
```

See Also

`setImportsGraphics` (page 327)

interval

Returns the amount of time that the cell pauses between messages when it's sending action messages continuously to target objects.

```
public float interval()
```

Discussion

Override this method to supply your own interval value.

See Also

`periodicDelay` (page 318)

intValue

Returns the receiver's value as an `int`.

```
public int intValue()
```

Discussion

If the receiver is not a text-type cell or the cell value is not scannable, the method returns 0.

isBezeled

Returns whether the receiver has a bezeled border.

```
public boolean isBezeled()
```

See Also
`setBezeled` (page 321)

isBordered

Returns whether the receiver has a plain border.

```
public boolean isBordered()
```

See Also
`setBordered` (page 321)

isContinuous

Returns whether the receiver sends its action message continuously on mouse down.

```
public boolean isContinuous()
```

See Also
`setContinuous` (page 322)

isEditable

Returns whether the receiver is editable.

```
public boolean isEditable()
```

See Also
`setEditable` (page 323)

isEnabled

Returns whether the receiver responds to mouse events.

```
public boolean isEnabled()
```

See Also
`setEnabled` (page 324)

isEntryAcceptable

Returns whether a string representing a numeric or date value (*aString*) is formatted in a way suitable to the entry type.

```
public boolean isEntryAcceptable(String aString)
```

Discussion

> **Note:** This method is being deprecated in favor of a new class of formatter objects. For more information, see NSFormatter. This documentation is provided only for developers who need to modify older applications.

See Also

`entryType` (page 310)

`setEntryType` (page 324)

isHighlighted

Returns whether the receiver is highlighted.

```
public boolean isHighlighted()
```

isOpaque

Returns whether the receiver is opaque (nontransparent).

```
public boolean isOpaque()
```

isScrollable

Returns whether the receiver scrolls typed text that exceeds the cell's bounds.

```
public boolean isScrollable()
```

See Also

`setScrollable` (page 329)

isSelectable

Returns whether the text of the receiver can be selected.

```
public boolean isSelectable()
```

See Also

`setSelectable` (page 329)

keyEquivalent

Implemented by subclasses to return a key equivalent to clicking the cell.

```
public String keyEquivalent()
```

Discussion

The default implementation returns an empty string object.

lineBreakMode

Returns the line break mode currently used when drawing text.

```
public int lineBreakMode()
```

Availability

Available in Mac OS X v10.4 and later.

See Also

`setLineBreakMode` (page 327)

menu

Returns the menu with commands contextually related to the cell or `null` if no menu is associated.

```
public NSMenu menu()
```

See Also

`setMenu` (page 328)

menuForEvent

Returns the NSMenu associated with the receiver through the `setMenu` (page 328) method and related to *anEvent* when the cursor is detected within *cellFrame*.

```
public NSMenu menuForEvent(NSEvent anEvent, NSRect cellFrame, NSView aView)
```

Discussion

It is usually invoked by the NSControl (*aView*) managing the receiver. The default implementation simply invokes `menu` (page 316) and returns `null` if no menu has been set. Subclasses can override to customize the returned menu according to the event received and the area in which the mouse event occurs.

mnemonic

Returns the character in the receiver's title that appears underlined for use as a mnemonic.

```
public String mnemonic()
```

Discussion

If there is no mnemonic character, returns an empty string. Mnemonics are not supported in Mac OS X

See Also

`setTitleWithMnemonic` (page 332)

mnemonicLocation

Returns the position of the underlined character in the receiver's title used as a mnemonic.

```
public int mnemonicLocation()
```

Discussion

If there is no mnemonic character, returns `NSArray.NotFound`. Mnemonics are not supported in Mac OS X.

See Also

`setMnemonicLocation` (page 328)

mouseDownFlags

Returns the modifier flags for the last (left) mouse-down event, or 0 if tracking hasn't occurred yet for the cell or no modifier keys accompanied the mouse-down event.

```
public int mouseDownFlags()
```

See Also

`modifierFlags` (page 616) (NSEvent)

nextState

Returns the receiver's next state.

```
public int nextState()
```

Discussion

If the receiver has three states, it cycles through them in this order: on, off, mixed, on, and so forth. If the receiver has two states, it toggles between them.

See Also

`allowsMixedState` (page 306)

`setAllowsMixedState` (page 320)

`setNextState` (page 328)

objectValue

Returns the receiver's value as an object if a valid object has been associated with the receiver; otherwise, returns `null`.

```
public Object objectValue()
```

Discussion

To be valid, the receiver must have a formatter capable of converting the object to and from its textual representation.

performClick

Can be used to simulate a single mouse click on the receiver.

```
public void performClick(Object sender)
```

Discussion

The receiver must be enabled and either the receiver's current `controlView` (page 309) must be valid or *`sender`*, which must be a subclass of NSView, is used instead. The receiver's action is performed on its target. Throws an exception if the action message cannot be successfully sent.

periodicDelay

Returns the amount of time that the cell pauses before sending its first message when it's sending action messages continuously to target objects.

```
public float periodicDelay()
```

Discussion

Override this method to supply your own delay value.

See Also

`interval` (page 313)

refusesFirstResponder

Returns `true` if the receiver can never become the first responder.

```
public boolean refusesFirstResponder()
```

Discussion

To find out whether the receiver can become first responder at this time, use the method `acceptsFirstResponder` (page 305).

See Also

`setRefusesFirstResponder` (page 329)

representedObject

Returns the object the receiver represents.

```
public Object representedObject()
```

Discussion

For example, you could have a pop-up list of color names, and the represented objects could be the appropriate NSColor objects.

See Also

`setRepresentedObject` (page 329)

resetCursorRect

Sets the receiver to show the I-beam cursor within `cellFrame` while it tracks the mouse.

```
public void resetCursorRect(NSRect cellFrame, NSView controlView)
```

Discussion

The receiver must be an enabled and selectable (or editable) text-type cell. `controlView` is the NSControl that manages the cell.

This method is invoked by `resetCursorRects` (page 1770) and in general you do not need to call this method unless you have a custom NSView that uses a cell.

selectAndEditWithFrameInView

Uses the field editor `textObj` to select text in a range marked by `selStart` and `selLength`, which will be highlighted and selected as though the user had dragged the cursor over it.

```
public void selectAndEditWithFrameInView(NSRect aRect, NSView controlView, NSText
     textObj, Object anObject, int selStart, int selLength)
```

Discussion

This method is similar to `editWithFrameInView` (page 310), except that it can be invoked in any situation, not only on a mouse-down event. `aRect` is the rectangle in which the selection should occur, `controlView` is the NSControl managing the receiver, and `anObject` is the delegate of the field editor. Returns without doing anything if `controlView`, `textObj`, or the receiver is `null`, or if the receiver has no font set for it.

sendsActionOnEndEditing

Returns whether the receiver's NSControl object sends its action message whenever the user finishes editing the cell's text.

```
public boolean sendsActionOnEndEditing()
```

Discussion

If it returns `true`, the receiver's NSControl object sends its action message when the user does one of the following:

- Presses the Return key
- Presses the Tab key to move out of the field
- Clicks another text field

If it returns `false`, the cell's NSControl object sends its action message only when the user presses the Return key.

See Also

`setSendsActionOnEndEditing` (page 330)

setAction

```
public void setAction(NSSelector aSelector)
```

Discussion

In NSCell, throws `InternalInconsistencyException`. However, NSActionCell overrides this method to set the action method, depending on the value of *`aSelector`*, as part of the implementation of the target/action mechanism.

See Also

`action` (page 305)

`setTarget` (page 331)

`target` (page 336)

setAlignment

Sets the alignment of text in the receiver.

```
public void setAlignment(int mode)
```

Discussion

`mode` is one of five constants: `NSText.LeftTextAlignment`, `NSText.RightTextAlignment`, `NSText.CenterTextAlignment`, `NSText.JustifiedTextAlignment`, and `NSText.NaturalTextAlignment` (the default alignment for the text).

See Also

`alignment` (page 305)

`setWraps` (page 333)

setAllowsEditingTextAttributes

Sets whether the textual attributes of the receiver can be modified by the user.

```
public void setAllowsEditingTextAttributes(boolean flag)
```

Discussion

If *`flag`* is `false`, the receiver also cannot import graphics (that is, it does not support RTFD text).

See Also

`allowsEditingTextAttributes` (page 305)

`setImportsGraphics` (page 327)

setAllowsMixedState

```
public void setAllowsMixedState(boolean flag)
```

Discussion

If *`flag`* is `true`, the receiver has three states: on, off, and mixed. If *`flag`* is `false`, the receiver has two states: on and off.

See Also

`allowsMixedState` (page 306)

`nextState` (page 317)

`setNextState` (page 328)

setAllowsUndo

If *allowsUndo* is true, the receiver assumes responsibility for undo operations within the cell.

```
public void setAllowsUndo(boolean allowsUndo)
```

Discussion
If *allowsUndo* is false, undo operations are handled by the application's custom undo manager.

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

See Also
allowsUndo (page 306)

setAttributedStringValue

Sets the value of the receiver to the attributed string *attribStr*.

```
public void setAttributedStringValue(NSAttributedString attribStr)
```

Discussion
If a formatter is set for the receiver, but the formatter does not understand the attributed string, it marks *attribStr* as an invalid object. If the receiver is not a text-type cell, it's converted to one.

For Mac OS X v10.3 and later: If you use a class that responds to the selector attributedStringValue (page 306) for the object value of a cell, then the cell will use that method to fetch the string to draw rather than using stringValue (page 334).

See Also
attributedStringValue (page 306)

setBezeled

Sets whether the receiver draws itself with a bezeled border, depending on the Boolean value *flag*.

```
public void setBezeled(boolean flag)
```

Discussion
The setBezeled and setBordered (page 321) methods are mutually exclusive (that is, a border can be only plain or bezeled). Invoking this method results in setBordered (page 321) being sent with a value of false.

See Also
isBezeled (page 314)

setBordered

Sets whether the receiver draws itself outlined with a plain border, depending on the Boolean value *flag*.

```
public void setBordered(boolean flag)
```

Discussion

The `setBezeled` (page 321) and `setBordered` methods are mutually exclusive (that is, a border can be only plain or bezeled). Invoking this method results in `setBezeled` (page 321) being sent with a value of `false`.

See Also

`isBordered` (page 314)

setCellAttribute

Sets a cell attribute identified by *aParameter*—such as the receiver's state and whether it's disabled, editable, or highlighted—to *value*.

```
public void setCellAttribute(int aParameter, int value)
```

See Also

`cellAttribute` (page 307)

setContinuous

Sets whether the receiver continuously sends its action message to its target while it tracks the mouse, depending on the Boolean value *flag*.

```
public void setContinuous(boolean flag)
```

Discussion

In practice, the continuous setting has meaning only for instances of NSActionCell and its subclasses, which implement the target/action mechanism. Some NSControl subclasses, notably NSMatrix, send a default action to a default target when a cell doesn't provide a target or action.

See Also

`isContinuous` (page 314)

`setEventMaskForSendingAction` (page 324)

setControlSize

Sets the size of the receiver.

```
public void setControlSize(int size)
```

Discussion

Valid values for *size* are described in "Constants" (page 337).

Changing the cell's control size does not change the font of the cell. Use the NSFont class method `systemFontSizeForControlSize` (page 652) to obtain the system font based on the new control size and set it.

See Also

`controlSize` (page 308)

setControlTint

Sets the receiver's control tint.

```
public void setControlTint(int controlTint)
```

Discussion

Valid values for *controlTint* are described in "Constants" (page 337).

See Also

`controlTint` (page 308)

setControlView

Sets the receiver's control view.

```
public void setControlView(NSView view)
```

Discussion

The control view represents the control currently being rendered by the cell.

Availability

Available in Mac OS X v10.4 and later.

See Also

`controlView` (page 309)

setDoubleValue

Sets the value of the receiver to an object *aDouble*, representing a double value.

```
public void setDoubleValue(double aDouble)
```

Discussion

In its implementation, this method invokes `setObjectValue` (page 328). Does nothing if the receiver is not a text-type cell.

See Also

`doubleValue` (page 309)

setEditable

Sets whether the user can edit the receiver's text, depending on the Boolean value *flag*.

```
public void setEditable(boolean flag)
```

Discussion

If *flag* is `true`, the text is also made selectable. If *flag* is `false`, the selectable attribute is restored to the value it was before the cell was last made editable.

See Also

`isEditable` (page 314)

`setSelectable` (page 329)

setEnabled

Sets whether the receiver is enabled or disabled, depending on the Boolean value *flag*.

```
public void setEnabled(boolean flag)
```

Discussion
The text of disabled cells is changed to gray. If a cell is disabled, it cannot be highlighted, does not support mouse tracking (and thus cannot participate in target/action functionality), and cannot be edited. However, you can still alter many attributes of a disabled cell programmatically (`setState` (page 330), for instance, will still work).

See Also
`isEnabled` (page 314)

setEntryType

Sets how numeric data is formatted in the receiver and places restrictions on acceptable input.

```
public void setEntryType(int aType)
```

Discussion
aType can be one of the types defined in "Constants" (page 337). The formatter associated with the receiver is replaced with a newly instantiated formatter appropriate to the entry type.

If the receiver isn't a text-type cell, this method converts it to one; in the process, it makes its title "Cell" and sets its font to the user's system font at 12 points.

You can check whether formatted strings conform to the entry types of cells with the `isEntryAcceptable` (page 314) method. NSControl subclasses also use `isEntryAcceptable` (page 314) to validate what users have typed in editable cells. You can control the format of values accepted and displayed in cells by creating a custom subclass of NSFormatter and associating an instance of that class with cells (through `setFormatter` (page 326)). In custom NSCell subclasses, you can also override `isEntryAcceptable` (page 314) to check for the validity of data entered into cells.

> **Note:** This method is being deprecated in favor of a new class of formatter objects. For more information, see NSFormatter. This documentation is provided only for developers who need to modify older applications.

See Also
`entryType` (page 310)

setEventMaskForSendingAction

Sets the conditions on which the receiver sends action messages to its target and returns a bit mask with which to detect the previous settings.

```
public int setEventMaskForSendingAction(int mask)
```

Discussion

`setEventMaskForSendingAction` is used during mouse tracking when the mouse button state changes, the mouse moves, or, if the cell is marked to send its action continuously while tracking, periodically. Because of this, the only bits checked in *mask* are `NSEvent.LeftMouseDownMask`, `NSEvent.LeftMouseUpMask`, `NSEvent.LeftMouseDraggedMask`, and `NSEvent.PeriodicMask`.

You can use `setContinuous` (page 322) to turn on the flag corresponding to `NSEvent.PeriodicMask` or `NSEvent.LeftMouseDraggedMask`, whichever is appropriate to the given subclass of NSCell.

See Also

`action` (page 305)

setFloatingPointFormat

Sets whether floating-point numbers are autoranged in the receiver and sets the sizes of the fields to the left and right of the decimal point.

```
public void setFloatingPointFormat(boolean autoRange, int leftDigits, int
    rightDigits)
```

Discussion

If *autoRange* is `false`, *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 `true`, *leftDigits* and *rightDigits* are simply added to form a maximum total field width for the receiver (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.

See Also

`setEntryType` (page 324)

setFloatValue

Sets the value of the receiver to an object *aFloat*, representing a float value.

```
public void setFloatValue(float aFloat)
```

Discussion

In its implementation, this method invokes `setObjectValue` (page 328). Does nothing if the receiver is not a text-type cell.

See Also

`floatValue` (page 311)

setFocusRingType

Sets the type of focus ring to be used.

```
public void setFocusRingType(int focusRingType)
```

Discussion
Possible values are listed in the "Constants" (page 727) section of NSGraphics. You can disable a view's focus ring drawing by calling this method with `NSGraphics.FocusRingTypeNone`. You should only disable a view from drawing its focus ring if you want to draw your own focus ring, or if there isn't sufficient space to display a focus ring in the default location.

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

See Also
`focusRingType` (page 311)
`defaultFocusRingType` (page 304)

setFont

Sets the font to be used when the receiver displays text.

```
public void setFont(NSFont fontObj)
```

Discussion
If the receiver is not a text-type cell, the method converts it to that type.

See Also
`font` (page 311)

setFormatter

Sets the formatter object used to format the textual representation of the receiver's object value and to validate cell input and convert it to that object value.

```
public void setFormatter(NSFormatter newFormatter)
```

Discussion
If the new formatter cannot interpret the receiver's current object value, that value is converted to a string object. If *newFormatter* is `null`, the receiver is disassociated from the current formatter.

See Also
`formatter` (page 311)

setHighlighted

Sets whether the receiver has a highlighted appearance, depending on the Boolean value *flag*.

```
public void setHighlighted(boolean flag)
```

Discussion
By default, it does nothing. NSButtonCell overrides this method to draw the button with the appearance specified by `LightsByBackground`, `LightsByContents`, or `LightsByGray`.

setImage

Sets the image to be displayed by the receiver.

```
public void setImage(NSImage image)
```

Discussion
If the receiver is not an image-type cell, the method converts it to that type.

See Also
`image` (page 313)

setImportsGraphics

Sets whether the receiver can import images into its text (that is, whether it supports RTFD text).

```
public void setImportsGraphics(boolean flag)
```

Discussion
If *`flag`* is `true`, the receiver is also set to allow editing of text attributes (`setAllowsEditingTextAttributes` (page 320)).

See Also
`importsGraphics` (page 313)

setIntValue

Sets the value of the receiver to an object *`anInt`*, representing an integer value.

```
public void setIntValue(int anInt)
```

Discussion
In its implementation, this method invokes `setObjectValue` (page 328). Does nothing if the receiver is not a text-type cell.

See Also
`intValue` (page 313)

setLineBreakMode

Sets the line break mode used when drawing text to *`mode`*.

```
public void setLineBreakMode(int mode)
```

Discussion
The line break mode can also be modified by calling the `setWraps` (page 333) method.

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

See Also
`lineBreakMode` (page 316)

`setWraps` (page 333)

setMenu

Associates a menu with the cell that has commands contextually related to the receiver.

```
public void setMenu(NSMenu aMenu)
```

Discussion
The associated menu is retained. If *aMenu* is `null`, any association with a previous menu is removed.

See Also
`menu` (page 316)

setMnemonicLocation

Sets the character of the receiver's title identified by *location* to be underlined.

```
public void setMnemonicLocation(int location)
```

Discussion
Mnemonics are not supported in Mac OS X.

See Also
`mnemonicLocation` (page 317)

setNextState

Sets the receiver to its next state.

```
public void setNextState()
```

Discussion
If the receiver has three states, it cycles through them in this order: on, off, mixed, on, and so forth. If the receiver has two states, it toggles between them.

See Also
`allowsMixedState` (page 306)
`nextState` (page 317)
`setAllowsMixedState` (page 320)

setObjectValue

Sets the receiver's object value to *object*.

```
public void setObjectValue(Object object)
```

See Also
`objectValue` (page 317)
`setRepresentedObject` (page 329)

setRefusesFirstResponder

Sets whether the receiver can become the first responder.

```
public void setRefusesFirstResponder(boolean flag)
```

Discussion

If *flag* is true, the receiver cannot become the first responder.

If refusesFirstResponder (page 318) returns false and the cell is enabled, the method acceptsFirstResponder (page 305) returns true, allowing the cell to become first responder.

setRepresentedObject

Sets the object represented by the receiver—f

```
public void setRepresentedObject(Object anObject)
```

Discussion

or example, an NSColor object for a cell with a title of "Blue."

See Also

setObjectValue (page 328)

representedObject (page 318)

setScrollable

Sets whether excess text in the receiver is scrolled past the cell's bounds.

```
public void setScrollable(boolean flag)
```

Discussion

If *flag* is true, wrapping is turned off.

See Also

isScrollable (page 315)

setSelectable

Sets whether text in the receiver can be selected, depending on the Boolean value *flag*.

```
public void setSelectable(boolean flag)
```

Discussion

If *flag* is false, editability is also disabled. If *flag* is true, editability is not affected.

See Also

isSelectable (page 315)

setEditable (page 323)

setSendsActionOnEndEditing

Sets whether the receiver's NSControl object sends its action message whenever the user finishes editing the cell's text.

```
public void setSendsActionOnEndEditing(boolean flag)
```

Discussion

If *flag* is `true`, the receiver's NSControl object sends its action message when the user does one of the following:

- Presses the Return key
- Presses the Tab key to move out of the field
- Clicks another text field

If *flag* is `false`, the cell's NSControl object sends its action message only when the user presses the Return key.

See Also

`sendsActionOnEndEditing` (page 319)

setShowsFirstResponder

Sets whether the receiver draws some indication of its first responder status.

```
public void setShowsFirstResponder(boolean flag)
```

Discussion

NSCell itself does not draw any indication but subclasses may use *flag* to decide to draw a focus ring.

See Also

`showsFirstResponder` (page 333)

setState

Sets the receiver's state to *value*, which can be `OnState`, `OffState`, or `MixedState`.

```
public void setState(int value)
```

Discussion

A cell can have two or three states. If it has two, *value* can be `OffState` (the normal or unpressed state) or `OnState` (the alternate or pressed state). If it has three, *value* can be `OnState` (the feature is in effect everywhere), `OffState` (the feature is in effect nowhere), or `MixedState` (the feature is in effect somewhere). Note that if the cell has only two states and *value* is `MixedState`, this method sets the cell's state to `OnState`.

Although using the enumerated constants is preferred, *value* can also be an integer. If the cell has two states, 0 is treated as `OffState`, and a nonzero value is treated as `OnState`. If the cell has three states, 0 is treated as `OffState`; a negative value, as `MixedState`; and a positive value, as `OnState`.

Note that the value `state` (page 333) returns may not be the same value you passed into `setState`.

To check whether the cell has three states (and uses the mixed state), invoke the method `allowsMixedState` (page 306).

See Also
`state` (page 333)

setStringValue

Sets the value of the receiver to *aString*.

```
public void setStringValue(String aString)
```

Discussion
In its implementation, this method invokes `setObjectValue` (page 328). If no formatter is assigned to the receiver or if the formatter cannot “translate” *aString* to an underlying object, the receiver is flagged as having an invalid object. If the receiver is not a text-type cell, this method converts it to one before setting the object value.

For Mac OS X v10.3 and later: If you use a class that responds to the selector `attributedStringValue` (page 306) for the object value of a cell, then the cell will use that method to fetch the string to draw rather than using `stringValue` (page 334).

See Also
`stringValue` (page 334)

setTag

Implemented by NSActionCell to set the receiver’s tag integer to *anInt*.

```
public void setTag(int anInt)
```

Discussion
NSCell’s implementation throws `InternalInconsistencyException`. Tags allow you to identify particular cells. Tag values are not used internally; they are only changed by external invocations of `setTag`. You typically set tag values in Interface Builder. When you set the tag of a control with a single cell in Interface Builder, it sets the tags of both the control and the cell to the same value as a convenience.

See Also
`tag` (page 334)

setTarget

Implemented by NSActionCell to set the receiver’s target object receiving the action message to *anObject*.

```
public void setTarget(Object anObject)
```

Discussion
NSCell’s implementation throws `InternalInconsistencyException`.

See Also
`target` (page 336)

setTitle

Sets the title of the receiver to *aString*.

```
public void setTitle(String aString)
```

setTitleWithMnemonic

Sets the title of the receiver to *aString* with a character denoted as an access key.

```
public void setTitleWithMnemonic(String aString)
```

Discussion
Mnemonics are not supported in Mac OS X.

See Also
`mnemonic` (page 316)
`setMnemonicLocation` (page 328)

setType

If the type of the receiver is different from *aType*, sets it to *aType*, which must be one of `TextCellType`, `ImageCellType`, or `NullCellType`.

```
public void setType(int aType)
```

Discussion
If *aType* is `TextCellType`, converts the receiver to a cell of that type, giving it a default title and setting the font to the system font at the default size. If *aType* is `ImageCellType`, sets a `null` image. If *aType* is set to `ImageCellType` and the image is set to `null`, the cell type will be reported as `NullCellType` until a new, non-`null` image is set.

See Also
`type` (page 337)

setUpFieldEditorAttributes

Sets textual and background attributes of *textObj*, depending on certain attributes.

```
public NSText setUpFieldEditorAttributes(NSText textObj)
```

Discussion
If the receiver is disabled, sets the text color to dark gray; otherwise sets it to the default color. If the receiver has a bezeled border, sets the background to the default color for text backgrounds; otherwise, sets it to the color of the receiver's NSControl.

You should not use this method to substitute a new field editor. `setUpFieldEditorAttributes` (page 332) is intended to modify the attributes of the text object (that is, the field editor) passed into it and return that text object. If you want to substitute your own field editor, use the NSWindow method `fieldEditorForObject` (page 1832) or the NSWindow delegate method `windowWillReturnFieldEditor` (page 1882).

setWraps

Sets whether text in the receiver wraps when its length exceeds the frame of the cell.

```
public void setWraps(boolean flag)
```

Discussion

If *flag* is `true`, then it also sets the receiver to be nonscrollable.

If the text of the receiver is an attributed string value you must explicitly set the paragraph style line break mode. Calling this method with the value `true` is equivalent to calling the `setLineBreakMode:` method with the value NSLineBreakByWordWrapping.

See Also

`setLineBreakMode` (page 327)

`wraps` (page 337)

showsFirstResponder

Returns whether the receiver should draw some indication when it assumes first responder status.

```
public boolean showsFirstResponder()
```

See Also

`setShowsFirstResponder` (page 330)

startTrackingMouse

```
public boolean startTrackingMouse(NSPoint startPoint, NSView controlView)
```

Discussion

NSCell's implementation of `trackMouse` (page 336) invokes this method when tracking begins. *startPoint* is the point the cursor is currently at, and *controlView* is the NSControl managing the receiver. NSCell's default implementation returns `true` if the receiver is either set to respond continuously or set to respond when the mouse is dragged. Otherwise, `false` is returned. Subclasses override this method to implement special mouse-tracking behavior at the beginning of mouse tracking—for example, displaying a special cursor.

See Also

`continueTrackingMouse` (page 308)

`stopTrackingMouse` (page 334)

state

Returns the receiver's state.

```
public int state()
```

Discussion

The receiver can have two or three states. If it has two, it returns either `OffState` (the normal or unpressed state) or `OnState` (the alternate or pressed state). If it has three, it returns `OnState` (the feature is in effect everywhere), `OffState` (the feature is in effect nowhere), or `MixedState` (the feature is in effect somewhere).

To check whether the receiver uses the mixed state, use the method `allowsMixedState` (page 306).

Note that the value `state` (page 333) returns may not be the same value you passed into `setState` (page 330).

See Also

`setState` (page 330)

stopTrackingMouse

```
public void stopTrackingMouse(NSPoint lastPoint, NSPoint stopPoint, NSView
    controlView, boolean flag)
```

Discussion

NSCell's implementation of `trackMouse` (page 336) invokes this method when the cursor has left the bounds of the receiver or the mouse button goes up (in which case *`flag`* is `true`). *`lastPoint`* is the point the cursor was at, and *`stopPoint`* is its current point. *`controlView`* is the NSControl managing the receiver. NSCell's default implementation does nothing. Subclasses often override this method to provide customized tracking behavior.

See Also

`startTrackingMouse` (page 333)

`stopTrackingMouse` (page 334)

stringValue

Returns the receiver's value as a String as converted by the receiver's formatter, if one exists.

```
public String stringValue()
```

Discussion

If no formatter exists and the value is a String, returns the value as a plain, attributed, or localized formatted string. If the value is not a String or can't be converted to one, returns an empty string.

For Mac OS X v10.3 and later: If you use a class that responds to the selector `attributedStringValue` (page 306) for the object value of a cell, then the cell will use that method to fetch the string to draw rather than using `stringValue`.

See Also

`setStringValue` (page 331)

tag

Implemented by NSActionCell to return the receiver's tag integer.

```
public int tag()
```

Discussion

NSCell's implementation returns –1. Tags allow you to identify particular cells. You typically set tag values in Interface Builder. When you set the tag of a control with a single cell in Interface Builder, it sets the tags of both the control and the cell to the same value as a convenience.

See Also

`setTag` (page 331)

takeDoubleValue

Sets the receiver's own value as a `double` using the `double` value of *`sender`*.

```
public void takeDoubleValue(Object sender)
```

See Also

`setDoubleValue` (page 323)

takeFloatValue

Sets the receiver's own value as a `float` using the `float` value of *`sender`*.

```
public void takeFloatValue(Object sender)
```

See Also

`setFloatValue` (page 325)

takeIntValue

Sets the receiver's own value as an `int` using the `int` value of *`sender`*.

```
public void takeIntValue(Object sender)
```

See Also

`setIntValue` (page 327)

takeObjectValue

Sets the receiver's own value as an object using the object value of *`sender`*.

```
public void takeObjectValue(Object sender)
```

See Also

`setObjectValue` (page 328)

takeStringValue

Sets the receiver's own value as a string object using the String value of *`sender`*.

```
public void takeStringValue(Object sender)
```

See Also
setStringValue (page 331)

target

```
public Object target()
```

Discussion
Implemented by NSActionCell to return the target object to which the receiver's action message is sent. NSCell's implementation returns null.

See Also
setTarget (page 331)

titleRectForBounds

```
public NSRect titleRectForBounds(NSRect theRect)
```

Discussion
If the receiver is a text-type cell, resizes the drawing rectangle for the title (*theRect*) inward by a small offset to accommodate the cell border. If the receiver is not a text-type cell, the method does nothing.

See Also
imageRectForBounds (page 313)

trackMouse

Invoked by an NSControl to initiate the tracking behavior of one of its NSCells.

```
public boolean trackMouse(NSEvent theEvent, NSRect cellFrame, NSView controlView,
     boolean untilMouseUp)
```

Discussion
It's generally not overridden because the default implementation invokes other NSCell methods that can be overridden to handle specific events in a dragging session. This method's return value depends on the *untilMouseUp* flag. If *untilMouseUp* is set to true, this method returns true if the mouse button goes up while the cursor is anywhere; false, otherwise. If *untilMouseUp* is set to false, this method returns true if the mouse button goes up while the cursor is within *cellFrame*; false, otherwise. The argument *theEvent* is typically the mouse event received by the initiating NSControl, usually identified by *controlView*. The *untilMouseUp* argument indicates whether tracking should continue until the mouse button goes up; if it's false, tracking ends when the mouse leaves the cell frame even if the mouse button isn't released.

This method first invokes startTrackingMouse (page 333). If that method returns true, then as mouse-dragged events are intercepted, continueTrackingMouse (page 308) is invoked until either the method returns false or the mouse is released. Finally, stopTrackingMouse (page 334) is invoked if the mouse is released. If *untilMouseUp* is true, it's invoked when the mouse button goes up while the cursor is anywhere. If *untilMouseUp* is false, it's invoked when the mouse button goes up while the cursor is within *cellFrame*. (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.

type

Returns the type of the receiver, one of `TextCellType`, `ImageCellType`, or `NullCellType`.

```
public int type()
```

Discussion
If the receiver's type is set to `ImageCellType` and its image is set to `null`, the cell type will be reported as `NullCellType` until a new, non-`null` image is set.

See Also
`setType` (page 332)

wraps

Returns whether text of the receiver wraps when it exceeds the borders of the cell.

```
public boolean wraps()
```

See Also
`setWraps` (page 333)

Constants

These constants specify how a cell formats numeric data. They're used by `setEntryType` (page 324) and `entryType` (page 310).

Constant	Description
`IntType`	Must be between `INT_MIN` and `INT_MAX`.
`PositiveIntType`	Must be between 1 and `INT_MAX`.
`FloatType`	Must be between `-FLT_MAX` and `FLT_MAX`.
`PositiveFloatType`	Must be between `FLT_MIN` and `FLT_MAX`.
`DoubleType`	Must be between `-DBL_MAX` and `DBL_MAX`.
`PositiveDoubleType`	Must be between `DBL_MAX` and `DBL_MAX`.
`AnyType`	Any value is allowed.

These constants specify what a cell contains. They're used by `setType` (page 332) and `type` (page 337).

Constant	Description
`NullCellType`	Cell displays nothing.
`TextCellType`	Cell displays text.

Constant	Description
ImageCellType	Cell displays images.

These constants specify how a button behaves when pressed and how it displays its state. They're used by NSButton and NSButtonCell:

Constant	Description
ChangeBackground	If the cell's state is MixedState or OnState, changes the cell's background color from gray to white.
ChangesContents	If the cell's state is MixedState or OnState, displays the cell's alternate image.
ChangeGray	If the cell's state is MixedState or OnState, displays the cell's image as darkened.
Disabled	Does not let the user manipulate the cell.
Editable	Lets the user edit the cell's contents.
HasImageHorizontal HasImageOnLeftOrBottom HasOverlappingImage	Controls the position of the cell's image and text. To place the image above, set none of them. To place the image below, set HasImageOnLeftOrBottom. To place the image to the right, set HasImageHorizontal. To place the image to the left, set HasImageHorizontal and HasImageOnLeftOrBottom. To place the image directly over, set HasOverlappingImage.
Highlighted	Draws the cell with a highlighted appearance. This constant is deprecated. Use setHighlighted (page 326) instead.
IsBordered	Draws a border around the cell.
IsInsetButton	Insets the cell's contents from the border. By default, they're inset by 2 pixels. This constant is ignored if the cell is unbordered.
LightsByBackground	If the cell is pushed in, changes the cell's background color from gray to white.
LightsByContents	If the cell is pushed in, displays the cell's alternate image.
LightsByGray	If the cell is pushed in, displays the cell's image as darkened.
PushIn	Determines whether the cell's image and text appear to be shifted down and to the right.
State	The cell's state. It can be MixedState, OffState, or OnState.

These constants specify the position of a button's image relative to its title. They're used by NSButton's and NSButtonCell's setImagePosition (page 262) and imagePosition (page 257).

Constant	Description
NoImage	The cell doesn't display an image.
ImageOnly	The cell displays an image, but not a title.
ImageLeft	The image is to the left of the title.
ImageRight	The image is to the right of the title.
ImageBelow	The image is below the title.
ImageAbove	The image is above the title.
ImageOverlaps	The image overlaps the title.

These constants specify a cell's state and are used mostly for buttons. They're described in "Cell States".

Constant	Description
MixedState	The corresponding feature is in effect somewhere.
OffState	The corresponding feature is in effect nowhere.
OnState	The corresponding feature is in effect everywhere.

These constants specify what happens when a button is pressed or is displaying its alternate state. They're used by NSButtonCell's highlightsBy (page 277) and showsStateBy (page 289).

Constant	Description
NoCellMask	The button cell doesn't change.
PushInCellMask	The button cell "pushes in" if it has a border.
ContentsCellMask	The button cell displays its alternate icon and/or title.
ChangeGrayCellMask	The button cell swaps the "control color" (NSColor's controlColor (page 361)) and white pixels on its background and icon.
ChangeBackgroundCellMask	Same as ChangeGrayCellMask, but only background pixels are changed.

These constants specify a cell's tint. They're used by controlTint (page 308) and setControlTint (page 323).

Constant	Description
DefaultControlTint	The current default tint setting
ClearControlTint	Clear control tint
BlueControlTint	Aqua control tint

Constant	Description
GraphiteControlTint	Graphite control tint

These constants specify a cell's size. They're used by controlSize (page 308) and setControlSize (page 322).

Constant	Description
RegularControlSize	The control is sized as regular.
SmallControlSize	The control has a smaller size. This constant is for controls that cannot be resized in one direction, such as push buttons, radio buttons, checkboxes, sliders, scroll bars, pop-up buttons, tabs, and progress indicators. You should use a small system font with a small control.
MiniControlSize	The control has a smaller size than SmallControlSize.

NSClipView

Inherits from	NSView : NSResponder : NSObject
Implements	NSCoding (NSResponder)
Package:	com.apple.cocoa.application
Companion guide	Drawing and Views

Class at a Glance

An NSClipView contains and scrolls the document view displayed by an NSScrollView. You normally don't need to program with NSClipViews, as NSScrollView handles most of the details of their operation.

Principal Attributes

- Efficient scrolling by copying drawn portions of the document view
- Monitoring of document view for automatic update

Interface Builder
Constructor
Creates the NSClipView.

Commonly Used Methods

`setDocumentView` (page 347)
Sets the view scrolled within the NSClipView.

`setCopiesOnScroll` (page 347)
Sets whether the NSClipView copies drawn portions of the document view during scrolling.

Overview

An NSClipView holds the document view of an NSScrollView, clipping the document view to its frame, handling the details of scrolling in an efficient manner, and updating the NSScrollView when the document view's size or position changes. 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.

Interaction With NSScrollView

When using an NSClipView within an NSScrollView (the usual configuration), you should issue messages that control background drawing state to the NSScrollView, rather than messaging the NSClipView directly. This recommendation applies to the following methods:

- `- setBackgroundColor:`
- `- backgroundColor`
- `- setDrawsBackground:`
- `- drawsBackground`

The NSClipView methods are intended for when the NSClipView is used independently of a containing NSScrollView. In the usual case, NSScrollView should be allowed to manage the background-drawing properties of its associated NSClipView.

There is only one background-drawing state per NSScrollView/NSClipView pair. The two objects do not maintain independent and distinct `drawsBackground` and `backgroundColor` properties; rather, NSScrollView's accessors for these properties largely defer to the associated NSClipView and allow the NSClipView to maintain the state. In Mac OS X v10.2 and earlier system versions, NSScrollView maintained a cache of the last state it set for its NSClipView. If the NSClipView was sent a `setDrawsBackground:` message directly, the cache might not reflect the state accurately. This caching of state has been removed in Mac OS X v10.3.

It is also important to note that sending a `setDrawsBackground:` message with a parameter of false to an NSScrollView has the added effect of sending the NSClipView a `setCopiesOnScroll:` message with a parameter of false. The side effect of sending the `setDrawsBackground:` message directly to the NSClipView is the appearance of "trails" (vestiges of previous drawing) in the document view as it is scrolled.

Tasks

Constructors

`NSClipView` (page 344)
Creates an NSClipView with a zero-sized frame rectangle.

Setting the Document View

`setDocumentView` (page 347)
Sets the receiver's document view to *aView*, removing any previous document view, and sets the origin of the receiver's bounds rectangle to the origin of *aView*'s frame rectangle.

`documentView` (page 346)
Returns the receiver's document view.

Scrolling

`scrollToPoint` (page 347)
Changes the origin of the receiver's bounds rectangle to *newOrigin*.

`autoscroll` (page 344)
Scrolls the receiver proportionally to *theEvent*'s distance outside of it.

`constrainScrollPoint` (page 345)
Returns a scroll point adjusted from *proposedNewOrigin*, if necessary, to guarantee the receiver will still lie within its document view.

Determining Scrolling Efficiency

`setCopiesOnScroll` (page 347)
Controls whether the receiver copies rendered images while scrolling.

`copiesOnScroll` (page 345)
Returns `true` if the receiver copies its existing rendered image while scrolling (only drawing exposed portions of its document view), `false` if it forces its contents to be redrawn each time.

Getting the Visible Portion

`documentRect` (page 346)
Returns the rectangle defining the document view's frame, adjusted to the size of the receiver if the document view is smaller.

`documentVisibleRect` (page 346)
Returns the exposed rectangle of the receiver's document view, in the document view's own coordinate system.

Setting the Document Cursor

`setDocumentCursor` (page 347)
Sets the cursor object used over the receiver to *aCursor*.

`documentCursor` (page 345)
Returns the cursor object used when the cursor lies over the receiver.

Working with Background Color

`drawsBackground` (page 346)
Returns `true` if the receiver draws its background color.

`setDrawsBackground` (page 348)
Sets whether the receiver draws its background color, depending on the Boolean value *`flag`*.

`setBackgroundColor` (page 347)
Sets the receiver's background color to *`aColor`*.

`backgroundColor` (page 345)
Returns the color of the receiver's background.

Overriding NSView Methods

`viewBoundsChanged` (page 348)
Handles a `ViewBoundsDidChangeNotification` (page 1788), passed in the *`aNotification`* argument, by updating a containing NSScrollView based on the new bounds.

`viewFrameChanged` (page 348)
Handles a `ViewFrameDidChangeNotification` (page 1789), passed in the *`aNotification`* argument, by updating a containing NSScrollView based on the new frame.

Constructors

NSClipView

Creates an NSClipView with a zero-sized frame rectangle.

```
public NSClipView()
```

Creates an NSClipView with *`frameRect`* as its frame rectangle.

```
public NSClipView(NSRect frameRect)
```

Instance Methods

autoscroll

Scrolls the receiver proportionally to *`theEvent`*'s distance outside of it.

```
public boolean autoscroll(NSEvent theEvent)
```

Discussion
`theEvent`'s location should be expressed in the window's base coordinate system (which it normally is), not the receiving NSClipView's. Returns `true` if any scrolling is performed; otherwise returns `false`.

Never invoke this method directly; instead, the NSClipView's document view should repeatedly send itself `autoscroll` (page 1742) messages when the cursor is dragged outside the NSClipView's frame during a modal event loop initiated by a mouse-down event. The NSView class implements `autoscroll` (page 1742) to forward the message to the receiver's superview; thus the message is ultimately forwarded to the NSClipView.

backgroundColor

Returns the color of the receiver's background.

```
public NSColor backgroundColor()
```

See Also

`setBackgroundColor` (page 347)

constrainScrollPoint

Returns a scroll point adjusted from *proposedNewOrigin*, if necessary, to guarantee the receiver will still lie within its document view.

```
public NSPoint constrainScrollPoint(NSPoint proposedNewOrigin)
```

Discussion

For example, if *proposedNewOrigin*'s y coordinate lies to the left of the document view's origin, then the y coordinate returned is set to that of the document view's origin.

See Also

`scrollToPoint` (page 347)

copiesOnScroll

Returns `true` if the receiver copies its existing rendered image while scrolling (only drawing exposed portions of its document view), `false` if it forces its contents to be redrawn each time.

```
public boolean copiesOnScroll()
```

See Also

`setCopiesOnScroll` (page 347)

documentCursor

Returns the cursor object used when the cursor lies over the receiver.

```
public NSCursor documentCursor()
```

See Also

`setDocumentCursor` (page 347)

documentRect

Returns the rectangle defining the document view's frame, adjusted to the size of the receiver if the document view is smaller.

```
public NSRect documentRect()
```

Discussion

In other words, this rectangle is always at least as large as the receiver itself.

The document rectangle is used in conjunction with an NSClipView's bounds rectangle to determine values for the indicators of relative position and size between the NSClipView and its document view. For example, NSScrollView uses these rectangles to set the size and position of the knobs in its scrollers. When the document view is much larger than the NSClipView, the knob is small; when the document view is near the same size, the knob is large; and when the document view is the same size or smaller, there is no knob.

See Also

`reflectScrolledClipView` (page 1276) (NSScrollView)

`documentVisibleRect` (page 346)

documentView

Returns the receiver's document view.

```
public NSView documentView()
```

See Also

`setDocumentView` (page 347)

documentVisibleRect

Returns the exposed rectangle of the receiver's document view, in the document view's own coordinate system.

```
public NSRect documentVisibleRect()
```

Discussion

Note that this rectangle doesn't reflect the effects of any clipping that may occur above the NSClipView itself. To get the portion of the document view that's guaranteed to be visible, send it a `visibleRect` message.

See Also

`documentRect` (page 346)

drawsBackground

Returns `true` if the receiver draws its background color.

```
public boolean drawsBackground()
```

See Also

`setDrawsBackground` (page 348)

scrollToPoint

Changes the origin of the receiver's bounds rectangle to *newOrigin*.

```
public void scrollToPoint(NSPoint newOrigin)
```

See Also
constrainScrollPoint (page 345)

setBackgroundColor

Sets the receiver's background color to *aColor*.

```
public void setBackgroundColor(NSColor aColor)
```

See Also
backgroundColor (page 345)

setCopiesOnScroll

Controls whether the receiver copies rendered images while scrolling.

```
public void setCopiesOnScroll(boolean flag)
```

Discussion
If *flag* is true, the receiver copies the existing rendered image to its new location while scrolling and only draws exposed portions of its document view. If *flag* is false, the receiver always forces its document view to draw itself on scrolling.

See Also
copiesOnScroll (page 345)

setDocumentCursor

Sets the cursor object used over the receiver to *aCursor*.

```
public void setDocumentCursor(NSCursor aCursor)
```

See Also
documentCursor (page 345)

setDocumentView

Sets the receiver's document view to *aView*, removing any previous document view, and sets the origin of the receiver's bounds rectangle to the origin of *aView*'s frame rectangle.

```
public void setDocumentView(NSView aView)
```

Discussion
If the receiver is contained in an NSScrollView, you should send the NSScrollView a setDocumentView (page 1279) message instead, so it can perform whatever updating it needs.

In the process of setting the document view, this method registers the receiver for the notifications `ViewFrameDidChangeNotification` (page 1789) and `ViewBoundsDidChangeNotification` (page 1788), adjusts the key view loop to include the new document view, and updates a parent NSScrollView's display if needed using `reflectScrolledClipView` (page 1276).

See Also

`documentView` (page 346)

setDrawsBackground

Sets whether the receiver draws its background color, depending on the Boolean value *`flag`*.

```
public void setDrawsBackground(boolean flag)
```

Discussion

If your NSClipView is enclosed in an NSScrollView, you should send the `setDrawsBackground:` message to the NSScrollView. Sending a `setDrawsBackground:` message with a parameter of false to an NSScrollView has the added effect of sending the NSClipView a `setCopiesOnScroll:` message with a parameter of false. The side effect of sending the `setDrawsBackground:` message directly to the NSClipView is the appearance of "trails" (vestiges of previous drawing) in the document view as it is scrolled.

See Also

`drawsBackground` (page 346)

viewBoundsChanged

Handles a `ViewBoundsDidChangeNotification` (page 1788), passed in the *`aNotification`* argument, by updating a containing NSScrollView based on the new bounds.

```
public void viewBoundsChanged(NSNotification aNotification)
```

viewFrameChanged

Handles a `ViewFrameDidChangeNotification` (page 1789), passed in the *`aNotification`* argument, by updating a containing NSScrollView based on the new frame.

```
public void viewFrameChanged(NSNotification aNotification)
```

NSColor

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.application
Companion guide	Color Programming Topics for Cocoa

Class at a Glance

An NSColor object represents a color, which is defined in a color space, each point of which has a set of components (such as red, green, and blue) that uniquely define a color.

Principal Attributes

- Color space
- Color components

Various `colorWith...` and `colorUsing...` methods.
Preset colors: `blackColor` (page 356), `blueColor` (page 357), and so on.

Commonly Used Methods

`colorUsingColorSpaceName` (page 372)
Creates an NSColor in the specified color space.

`set` (page 376)
Sets the drawing color.

Overview

An NSColor object represents color and sometimes opacity (alpha). By sending a `set` (page 376) message to an NSColor instance, you set the color for the current drawing context. Setting the color causes subsequently drawn graphics to have the color represented by the NSColor instance.

It is invalid to use an accessor method related to components of a particular color space on an NSColor object that is not in that color space. For example, methods such as `redComponent` (page 375) work on color objects in the calibrated and device RGB color spaces. If you send such a message to an NSColor object in the CMYK color space, an exception is raised. If you have an NSColor object in an unknown color space and you want to extract its components, you should first convert the color object to a known color space before using the component accessor methods of that color space.

For definitons of NSColor constants, as well as a discussion of their usage, see "About Color Spaces" in *Color Programming Topics for Cocoa*.

Tasks

Constructors

`NSColor` (page 356)
: Creates an empty NSColor.

Creating an NSColor Object from Component Values

`colorWithCalibratedHSB` (page 357)
: Creates and returns an NSColor whose opacity value is *alpha* and whose components in HSB space would be *hue*, *saturation*, and *brightness*.

`colorWithCalibratedRGB` (page 358)
: Creates and returns an NSColor whose opacity value is *alpha* and whose RGB components are *red*, *green*, and *blue*.

`colorWithCalibratedWhite` (page 358)
: Creates and returns an NSColor whose opacity value is *alpha* and whose grayscale value is *white*.

`colorWithCatalogName` (page 358)
: Creates and returns an NSColor by finding the color named *colorName* in the catalog named *listName*, which may be a standard catalog.

`colorWithDeviceCMYK` (page 359)
: Creates and returns an NSColor whose opacity value is *alpha* and whose CMYK components are *cyan*, *magenta*, *yellow*, and *black*.

`colorWithDeviceHSB` (page 359)
: Creates and returns an NSColor whose opacity value is *alpha* and whose components in HSB space would be *hue*, *saturation*, and *brightness*.

`colorWithDeviceRGB` (page 359)
: Creates and returns an NSColor whose opacity value is *alpha* and whose RGB components are *red*, *green*, and *blue*.

`colorWithDeviceWhite` (page 360)
: Creates and returns an NSColor whose opacity value is *alpha* and whose grayscale value is *white*.

`colorWithColorSpace` (page 359)
: Returns an NSColor object created from the specified *components* of color space *space*

Creating an NSColor with Preset Components

`blackColor` (page 356)
: Returns an NSColor whose grayscale value is 0.0 and whose alpha value is 1.0.

`blueColor` (page 357)
: Returns an NSColor whose RGB value is 0.0, 0.0, 1.0 and whose alpha value is 1.0.

`brownColor` (page 357)
: Returns an NSColor whose RGB value is 0.6, 0.4, 0.2 and whose alpha value is 1.0.

`clearColor` (page 357)
: Returns an NSColor whose grayscale and alpha values are both 0.0.

`cyanColor` (page 362)
: Returns an NSColor whose RGB value is 0.0, 1.0, 1.0 and whose alpha value is 1.0.

`darkGrayColor` (page 363)
: Returns an NSColor whose grayscale value is 1/3 and whose alpha value is 1.0.

`grayColor` (page 363)
: Returns an NSColor whose grayscale value is 0.5 and whose alpha value is 1.0.

`greenColor` (page 363)
: Returns an NSColor whose RGB value is 0.0, 1.0, 0.0 and whose alpha value is 1.0.

`lightGrayColor` (page 365)
: Returns an NSColor whose grayscale value is 2/3 and whose alpha value is 1.0.

`magentaColor` (page 365)
: Returns an NSColor whose RGB value is 1.0, 0.0, 1.0 and whose alpha value is 1.0.

`orangeColor` (page 365)
: Returns an NSColor whose RGB value is 1.0, 0.5, 0.0 and whose alpha value is 1.0.

`purpleColor` (page 365)
: Returns an NSColor whose RGB value is 0.5, 0.0, 0.5 and whose alpha value is 1.0.

`redColor` (page 365)
: Returns an NSColor whose RGB value is 1.0, 0.0, 0.0 and whose alpha value is 1.0.

`whiteColor` (page 369)
: Returns an NSColor whose grayscale and alpha values are both 1.0.

`yellowColor` (page 369)
: Returns an NSColor whose RGB value is 1.0, 1.0, 0.0 and whose alpha value is 1.0.

Working with Pattern Images

`colorWithPatternImage` (page 360)
: Creates and returns an NSColor that uses the pattern in *`image`*.

`patternImage` (page 375)
: Returns the image that the receiver is using as a pattern.

Creating a System Color—an NSColor Whose Value Is Specified by User Preferences

`alternateSelectedControlColor` (page 356)
: Returns the system color used for the face of a selected control—a control being clicked or dragged.

`alternateSelectedControlTextColor` (page 356)
: Returns the system color used for text in a selected control—a control being clicked or dragged.

`colorForControlTint` (page 357)
: Returns the NSColor specified by *`controlTint`*, which is one of the tint settings.

`controlBackgroundColor` (page 360)
: Returns the system color used for the background of large controls such as browsers, table views, and clip views.

`controlColor` (page 361)
: Returns the system color used for the flat surfaces of a control.

`controlAlternatingRowBackgroundColors` (page 360)
: Returns an array containing the system specified background colors for alternating rows in tables and lists.

`controlHighlightColor` (page 361)
: Returns the system color used for the highlighted bezels of controls.

`controlLightHighlightColor` (page 361)
: Returns the system color used for light highlights in controls.

`controlShadowColor` (page 362)
: Returns the system color used for the shadows dropped from controls.

`controlDarkShadowColor` (page 361)
: Returns the system color used for the dark edge of the shadow dropped from controls.

`controlTextColor` (page 362)
: Returns the system color used for text on controls that aren't disabled.

`currentControlTint` (page 362)
: Returns the current system control tint.

`disabledControlTextColor` (page 363)
: Returns the system color used for text on disabled controls.

`gridColor` (page 363)
: Returns the system color used for the optional gridlines in, for example, a table view.

`highlightColor` (page 364)
: Returns the system color that represents the virtual light source on the screen.

`keyboardFocusIndicatorColor` (page 364)
: Returns the system color that represents the keyboard focus ring around controls.

`knobColor` (page 364)
: Returns the system color used for the flat surface of a slider knob that hasn't been selected.

`scrollBarColor` (page 365)
: Returns the system color used for scroll "bars"—that is, for the groove in which a scroller's knob moves

`secondarySelectedControlColor` (page 366)
: Returns the system color used in nonkey views.

`selectedControlColor` (page 366)
: Returns the system color used for the face of a selected control—a control being clicked or dragged.

`selectedControlTextColor` (page 366)
: Returns the system color used for text in a selected control—a control being clicked or dragged.

`selectedMenuItemColor` (page 367)
: Returns the system color used for the face of selected menu items.

`selectedMenuItemTextColor` (page 367)
: Returns the system color used for the text in menu items.

`selectedTextBackgroundColor` (page 367)
: Returns the system color used for the background of selected text.

`selectedTextColor` (page 367)
: Returns the system color used for selected text.

`selectedKnobColor` (page 366)
: Returns the system color used for the slider knob when it is selected—that is, dragged.

`shadowColor` (page 368)
: Returns the system color that represents the virtual shadows cast by raised objects on the screen.

`textBackgroundColor` (page 368)
: Returns the system color used for the text background.

`textColor` (page 368)
: Returns the system color used for text.

`windowBackgroundColor` (page 369)
: Returns a pattern color that will draw the ruled lines for the window background.

`windowFrameColor` (page 369)
: Returns the system color used for window frames, except for their text.

`windowFrameTextColor` (page 369)
: Returns the system color used for the text in window frames.

Ignoring Alpha Components

`ignoresAlpha` (page 364)
: Returns `true` if the application doesn't support alpha.

`setIgnoresAlpha` (page 367)

Copying and Pasting

`colorFromPasteboard` (page 357)
: Returns the NSColor currently on *`pasteBoard`*, or `null` if *`pasteBoard`* doesn't contain color data.

`writeToPasteboard` (page 377)
: Writes the receiver's data to *`pasteBoard`*, unless *`pasteBoard`* doesn't support color data (in which case the method does nothing).

Retrieving a Set of Components

`components` (page 373)
Returns the components of the receiver as an array of `float` values.

`numberOfComponents` (page 375)
Returns the number of components in the receiver.

Retrieving Individual Components

`alphaComponent` (page 370)
Returns the receiver's alpha (opacity) component.

`blackComponent` (page 370)
Returns the receiver's black component.

`blueComponent` (page 370)
Returns the receiver's blue component.

`brightnessComponent` (page 370)
Returns the brightness component of the HSB color equivalent to the receiver.

`catalogNameComponent` (page 371)
Returns the name of the catalog containing the receiver's name.

`colorNameComponent` (page 371)
Returns the receiver's name.

`cyanComponent` (page 373)
Returns the receiver's cyan component.

`greenComponent` (page 374)
Returns the receiver's green component.

`hueComponent` (page 374)
Returns the hue component of the HSB color equivalent to the receiver.

`localizedCatalogNameComponent` (page 374)

`localizedColorNameComponent` (page 374)

`magentaComponent` (page 375)
Returns the receiver's magenta component.

`redComponent` (page 375)
Returns the receiver's red component.

`saturationComponent` (page 376)
Returns the saturation component of the HSB color equivalent to the receiver.

`whiteComponent` (page 377)
Returns the receiver's white component.

`yellowComponent` (page 378)
Returns the receiver's yellow component.

Working with the Color Space

`colorSpaceName` (page 371)
: Returns the name of the receiver's color space.

`colorUsingColorSpaceName` (page 372)
: Creates and returns an NSColor whose color is the same as the receiver's, except that the new NSColor is in the color space named *colorSpace*.

`colorUsingColorSpaceNameAndDevice` (page 372)
: Creates and returns an 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*.

`colorSpace` (page 371)
: Returns an object representing the color space of the receiver.

`colorUsingColorSpace` (page 372)
: Returns a new color object representing the color of the receiver in the specified color space *space*.

Changing the Color

`blendedColorWithFractionOfColor` (page 370)
: Creates and returns an NSColor whose component values are a weighted sum of the receiver's and *color*'s.

`colorWithAlphaComponent` (page 373)
: Creates and returns an NSColor that has the same color space and component values as the receiver, except its alpha component is *alpha*.

`highlightWithLevel` (page 374)
: Returns an NSColor that represents a blend between the receiver and the highlight color returned by `highlightColor` (page 364).

`shadowWithLevel` (page 377)
: Returns an NSColor that represents a blend between the receiver and the shadow color returned by `shadowColor` (page 368).

Drawing

`drawSwatchInRect` (page 373)
: Draws the current color in the rectangle *rect*.

`set` (page 376)
: Sets the color of subsequent drawing to the color that the receiver represents.

`setFill` (page 376)
: Sets the fill color of subsequent drawing to the receiver's color.

`setStroke` (page 376)
: Sets the stroke color of subsequent drawing to the receiver's color.

Constructors

NSColor

Creates an empty NSColor.

```
public NSColor()
```

Static Methods

alternateSelectedControlColor

Returns the system color used for the face of a selected control—a control being clicked or dragged.

```
public static NSColor alternateSelectedControlColor()
```

Discussion
For use where iApp-like highlighting is desired. For general information about system colors, see "Accessing System Colors".

Availability
Available in Mac OS X v10.2 and later.

See Also
`alternateSelectedControlTextColor` (page 356)
`selectedControlColor` (page 366)

alternateSelectedControlTextColor

Returns the system color used for text in a selected control—a control being clicked or dragged.

```
public static NSColor alternateSelectedControlTextColor()
```

Discussion
or use where iApp-like highlighting is desired. For general information about system colors, see "Accessing System Colors".

Availability
Available in Mac OS X v10.2 and later.

See Also
`alternateSelectedControlColor` (page 356)
`selectedControlTextColor` (page 366)

blackColor

Returns an NSColor whose grayscale value is 0.0 and whose alpha value is 1.0.

```
public static NSColor blackColor()
```

See Also

`blackComponent` (page 370)

blueColor

Returns an NSColor whose RGB value is 0.0, 0.0, 1.0 and whose alpha value is 1.0.

```
public static NSColor blueColor()
```

See Also

`blueComponent` (page 370)

brownColor

Returns an NSColor whose RGB value is 0.6, 0.4, 0.2 and whose alpha value is 1.0.

```
public static NSColor brownColor()
```

clearColor

Returns an NSColor whose grayscale and alpha values are both 0.0.

```
public static NSColor clearColor()
```

colorForControlTint

Returns the NSColor specified by *`controlTint`*, which is one of the tint settings.

```
public static NSColor colorForControlTint(int controlTint)
```

colorFromPasteboard

Returns the NSColor currently on *`pasteBoard`*, or `null` if *`pasteBoard`* doesn't contain color data.

```
public static NSColor colorFromPasteboard(NSPasteboard pasteBoard)
```

Discussion

The returned color's alpha component is set to 1.0 if `ignoresAlpha` (page 364) returns `true`.

See Also

`writeToPasteboard` (page 377)

colorWithCalibratedHSB

Creates and returns an NSColor whose opacity value is *`alpha`* and whose components in HSB space would be *`hue`*, *`saturation`*, and *`brightness`*.

```
public static NSColor colorWithCalibratedHSB(float hue, float saturation, float
    brightness, float alpha)
```

Discussion
(Values below 0.0 are interpreted as 0.0, and values above 1.0 are interpreted as 1.0.)

See Also
`colorWithCalibratedRGB` (page 358)
`colorWithDeviceHSB` (page 359)

colorWithCalibratedRGB

Creates and returns an NSColor whose opacity value is *alpha* and whose RGB components are *red*, *green*, and *blue*.

```
public static NSColor colorWithCalibratedRGB(float red, float green, float blue,
    float alpha)
```

Discussion
(Values below 0.0 are interpreted as 0.0, and values above 1.0 are interpreted as 1.0.)

See Also
`colorWithCalibratedHSB` (page 357)
`colorWithDeviceRGB` (page 359)

colorWithCalibratedWhite

Creates and returns an NSColor whose opacity value is *alpha* and whose grayscale value is *white*.

```
public static NSColor colorWithCalibratedWhite(float white, float alpha)
```

Discussion
(Values below 0.0 are interpreted as 0.0, and values above 1.0 are interpreted as 1.0.)

See Also
`colorWithDeviceWhite` (page 360)

colorWithCatalogName

Creates and returns an NSColor by finding the color named *colorName* in the catalog named *listName*, which may be a standard catalog.

```
public static NSColor colorWithCatalogName(String listName, String colorName)
```

Discussion
Note that the color must be defined in the named color space to retrieve it with this method.

See Also
`catalogNameComponent` (page 371)
`colorNameComponent` (page 371)
`localizedCatalogNameComponent` (page 374)

colorWithColorSpace

Returns an NSColor object created from the specified *components* of color space *space*

```
public static NSColor colorWithColorSpace(NSColorSpace space, float[] components)
```

Discussion
. *space* must be a NSColorSpace object representing a color space. The number of components in the *components* array should match the number dictated by the specified color space plus one for alpha. Throws an exception if they do not match. The order of the components is determined by the color-space profile, with the alpha component always last. (If you want the created color to be opaque, specify 1.0 for the alpha component.) If *space* represents a color space that cannot cannot be used with NSColor objects—for example, a "pattern" color space—the method returns `null`.

Availability
Available in Mac OS X v10.4.

See Also
`colorUsingColorSpace` (page 372)

colorWithDeviceCMYK

Creates and returns an NSColor whose opacity value is *alpha* and whose CMYK components are *cyan*, *magenta*, *yellow*, and *black*.

```
public static NSColor colorWithDeviceCMYK(float cyan, float magenta, float yellow,
     float black, float alpha)
```

Discussion
(Values below 0.0 are interpreted as 0.0, and values above 1.0 are interpreted as 1.0.) In PostScript, this color space corresponds directly to the device-dependent operator `setcmykcolor`.

colorWithDeviceHSB

Creates and returns an NSColor whose opacity value is *alpha* and whose components in HSB space would be *hue*, *saturation*, and *brightness*.

```
public static NSColor colorWithDeviceHSB(float hue, float saturation, float
    brightness, float alpha)
```

Discussion
(Values below 0.0 are interpreted as 0.0, and values above 1.0 are interpreted as 1.0.) In PostScript, this color space corresponds directly to the device-dependent operator `setrgbcolor`.

See Also
`colorWithCalibratedHSB` (page 357)
`colorWithDeviceRGB` (page 359)

colorWithDeviceRGB

Creates and returns an NSColor whose opacity value is *alpha* and whose RGB components are *red*, *green*, and *blue*.

```
public static NSColor colorWithDeviceRGB(float red, float green, float blue, float
     alpha)
```

Discussion
(Values below 0.0 are interpreted as 0.0, and values above 1.0 are interpreted as 1.0.) In PostScript, this color space corresponds directly to the device-dependent operator `setrgbcolor`.

See Also
`colorWithCalibratedRGB` (page 358)
`colorWithDeviceHSB` (page 359)

colorWithDeviceWhite

Creates and returns an NSColor whose opacity value is *alpha* and whose grayscale value is *white*.

```
public static NSColor colorWithDeviceWhite(float white, float alpha)
```

Discussion
(Values below 0.0 are interpreted as 0.0, and values above 1.0 are interpreted as 1.0.) In PostScript, this color space corresponds directly to the device-dependent operator `setgray`.

See Also
`colorWithCalibratedWhite` (page 358)

colorWithPatternImage

Creates and returns an NSColor that uses the pattern in *image*.

```
public static NSColor colorWithPatternImage(NSImage image)
```

Discussion
The image is tiled starting at the bottom of the window. The image is not scaled.

controlAlternatingRowBackgroundColors

Returns an array containing the system specified background colors for alternating rows in tables and lists.

```
public static NSArray controlAlternatingRowBackgroundColors()
```

Discussion
You should not assume the array will contain only two colors. For general information on system colors, see "Accessing System Colors".

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

controlBackgroundColor

Returns the system color used for the background of large controls such as browsers, table views, and clip views.

```
public static NSColor controlBackgroundColor()
```

Discussion

For general information on system colors, see "Accessing System Colors".

controlColor

Returns the system color used for the flat surfaces of a control.

```
public static NSColor controlColor()
```

Discussion

By default, the control color is a pattern color that will draw the ruled lines for the window background.

If you use `controlColor` assuming that it is a solid, you may have an incorrect appearance. You should use `lightGrayColor` (page 365) in its place.

controlDarkShadowColor

Returns the system color used for the dark edge of the shadow dropped from controls.

```
public static NSColor controlDarkShadowColor()
```

Discussion

Controls are displayed as though they were lit from the upper left. Two dark borders, representing shadows, run along the bottom and right. `controlDarkShadowColor` (page 361) returns the color of the outer, darker border. For general information about system colors, see "Accessing System Colors".

See Also

`controlShadowColor` (page 362)

controlHighlightColor

Returns the system color used for the highlighted bezels of controls.

```
public static NSColor controlHighlightColor()
```

Discussion

Controls are displayed as though they were lit from the upper left. Two light borders, representing reflections from the light source, run along the top and left. `controlHighlightColor` (page 361) returns the color of the inner, duller border. For general information about system colors, see "Accessing System Colors".

See Also

`controlLightHighlightColor` (page 361)

controlLightHighlightColor

Returns the system color used for light highlights in controls.

```
public static NSColor controlLightHighlightColor()
```

Discussion

Controls are displayed as though they were lit from the upper left. Two light borders, representing reflections from the light source, run along the top and left. `controlLightHighlightColor` (page 361) returns the color of the outer, brighter border. For general information about system colors, see "Accessing System Colors".

See Also

`controlHighlightColor` (page 361)

controlShadowColor

Returns the system color used for the shadows dropped from controls.

```
public static NSColor controlShadowColor()
```

Discussion

Controls are displayed as though they were lit from the upper left. Two dark borders, representing shadows, run along the bottom and right. `controlShadowColor` (page 362) returns the color of the inner, lighter border. For general information about system colors, see "Accessing System Colors".

See Also

`controlDarkShadowColor` (page 361)

controlTextColor

Returns the system color used for text on controls that aren't disabled.

```
public static NSColor controlTextColor()
```

Discussion

For general information about system colors, see "Accessing System Colors".

See Also

`disabledControlTextColor` (page 363)

currentControlTint

Returns the current system control tint.

```
public static int currentControlTint()
```

Availability

Available in Mac OS X v10.3 and later.

See Also

`colorForControlTint` (page 357)

cyanColor

Returns an NSColor whose RGB value is 0.0, 1.0, 1.0 and whose alpha value is 1.0.

```
public static NSColor cyanColor()
```

See Also
`cyanComponent` (page 373)

darkGrayColor

Returns an NSColor whose grayscale value is 1/3 and whose alpha value is 1.0.

```
public static NSColor darkGrayColor()
```

See Also
`lightGrayColor` (page 365)
`grayColor` (page 363)

disabledControlTextColor

Returns the system color used for text on disabled controls.

```
public static NSColor disabledControlTextColor()
```

Discussion
For general information about system colors, see "Accessing System Colors".

See Also
`controlTextColor` (page 362)

grayColor

Returns an NSColor whose grayscale value is 0.5 and whose alpha value is 1.0.

```
public static NSColor grayColor()
```

See Also
`lightGrayColor` (page 365)
`darkGrayColor` (page 363)

greenColor

Returns an NSColor whose RGB value is 0.0, 1.0, 0.0 and whose alpha value is 1.0.

```
public static NSColor greenColor()
```

See Also
`greenComponent` (page 374)

gridColor

Returns the system color used for the optional gridlines in, for example, a table view.

```
public static NSColor gridColor()
```

Discussion

For general information about system colors, see "Accessing System Colors".

highlightColor

Returns the system color that represents the virtual light source on the screen.

```
public static NSColor highlightColor()
```

Discussion

This method is invoked by the `highlightWithLevel` (page 374) method. For general information about system colors, see "Accessing System Colors".

See Also

`highlightWithLevel` (page 374)

ignoresAlpha

Returns `true` if the application doesn't support alpha.

```
public static boolean ignoresAlpha()
```

Discussion

This value is consulted when an application imports alpha (through color dragging, for instance). The value determines whether the color panel has an opacity slider. This value is `true` by default, indicating that the opacity components of imported colors will be set to 1.0. If an application wants alpha, it can invoke the `setIgnoresAlpha` (page 367) method with a parameter of `false`.

See Also

`setIgnoresAlpha` (page 367)

`alphaComponent` (page 370)

keyboardFocusIndicatorColor

Returns the system color that represents the keyboard focus ring around controls.

```
public static NSColor keyboardFocusIndicatorColor()
```

knobColor

Returns the system color used for the flat surface of a slider knob that hasn't been selected.

```
public static NSColor knobColor()
```

Discussion

The knob's beveled edges, which set it in relief, are drawn in highlighted and shadowed versions of the face color. When a knob is selected, its color changes to `selectedKnobColor` (page 366). For general information about system colors, see "Accessing System Colors".

lightGrayColor

Returns an NSColor whose grayscale value is 2/3 and whose alpha value is 1.0.

```
public static NSColor lightGrayColor()
```

See Also

grayColor (page 363)

darkGrayColor (page 363)

magentaColor

Returns an NSColor whose RGB value is 1.0, 0.0, 1.0 and whose alpha value is 1.0.

```
public static NSColor magentaColor()
```

See Also

magentaComponent (page 375)

orangeColor

Returns an NSColor whose RGB value is 1.0, 0.5, 0.0 and whose alpha value is 1.0.

```
public static NSColor orangeColor()
```

purpleColor

Returns an NSColor whose RGB value is 0.5, 0.0, 0.5 and whose alpha value is 1.0.

```
public static NSColor purpleColor()
```

redColor

Returns an NSColor whose RGB value is 1.0, 0.0, 0.0 and whose alpha value is 1.0.

```
public static NSColor redColor()
```

See Also

redComponent (page 375)

scrollBarColor

Returns the system color used for scroll “bars”—that is, for the groove in which a scroller’s knob moves

```
public static NSColor scrollBarColor()
```

Discussion

. For general information about system colors, see “Accessing System Colors”.

secondarySelectedControlColor

Returns the system color used in nonkey views.

```
public static NSColor secondarySelectedControlColor()
```

Discussion
For general information about system colors, see "Accessing System Colors".

See Also
`selectedControlColor` (page 366)

selectedControlColor

Returns the system color used for the face of a selected control—a control being clicked or dragged.

```
public static NSColor selectedControlColor()
```

Discussion
For general information about system colors, see "Accessing System Colors".

See Also
`selectedControlTextColor` (page 366)
`secondarySelectedControlColor` (page 366)
`alternateSelectedControlColor` (page 356)

selectedControlTextColor

Returns the system color used for text in a selected control—a control being clicked or dragged.

```
public static NSColor selectedControlTextColor()
```

Discussion
For general information about system colors, see "Accessing System Colors".

See Also
`selectedControlColor` (page 366)
`alternateSelectedControlTextColor` (page 356)

selectedKnobColor

Returns the system color used for the slider knob when it is selected—that is, dragged.

```
public static NSColor selectedKnobColor()
```

Discussion
For general information about system colors, see "Accessing System Colors".

See Also
`knobColor` (page 364)

selectedMenuItemColor

Returns the system color used for the face of selected menu items.

```
public static NSColor selectedMenuItemColor()
```

Discussion

For general information about system colors, see "Accessing System Colors".

See Also

`selectedMenuItemTextColor` (page 367)

selectedMenuItemTextColor

Returns the system color used for the text in menu items.

```
public static NSColor selectedMenuItemTextColor()
```

Discussion

For general information about system colors, see "Accessing System Colors".

See Also

`selectedMenuItemColor` (page 367)

selectedTextBackgroundColor

Returns the system color used for the background of selected text.

```
public static NSColor selectedTextBackgroundColor()
```

Discussion

For general information about system colors, see "Accessing System Colors".

See Also

`selectedTextColor` (page 367)

selectedTextColor

Returns the system color used for selected text.

```
public static NSColor selectedTextColor()
```

Discussion

For general information about system colors, see "Accessing System Colors".

See Also

`selectedTextBackgroundColor` (page 367)

setIgnoresAlpha

```
public static void setIgnoresAlpha(boolean flag)
```

Discussion

If *flag* is true, the application won't support alpha. In this case, no opacity slider is displayed in the color panel, and colors dragged in or pasted have their alpha values set to 1.0. By default, applications ignore alpha. Applications that need to import alpha can invoke this method with *flag* set to false and explicitly make colors opaque in cases where it matters to them. Note that calling this with a value of true overrides any value set with the NSColorPanel method setShowsAlpha (page 392).

See Also

ignoresAlpha (page 364)

alphaComponent (page 370)

shadowColor

Returns the system color that represents the virtual shadows cast by raised objects on the screen.

```
public static NSColor shadowColor()
```

Discussion

This method is invoked by shadowWithLevel (page 377). For general information about system colors, see "Accessing System Colors".

See Also

shadowWithLevel (page 377)

textBackgroundColor

Returns the system color used for the text background.

```
public static NSColor textBackgroundColor()
```

Discussion

When text is selected, its background color changes to the return value of selectedTextBackgroundColor (page 367). For general information about system colors, see "Accessing System Colors".

See Also

textColor (page 368)

textColor

Returns the system color used for text.

```
public static NSColor textColor()
```

Discussion

When text is selected, its background color changes to the return value of selectedTextColor (page 367). For general information about system colors, see "Accessing System Colors".

See Also

textBackgroundColor (page 368)

whiteColor

Returns an NSColor whose grayscale and alpha values are both 1.0.

```
public static NSColor whiteColor()
```

See Also

whiteComponent (page 377)

windowBackgroundColor

Returns a pattern color that will draw the ruled lines for the window background.

```
public static NSColor windowBackgroundColor()
```

Discussion

For general information about system colors, see "Accessing System Colors".

windowFrameColor

Returns the system color used for window frames, except for their text.

```
public static NSColor windowFrameColor()
```

Discussion

For general information about system colors, see "Accessing System Colors".

See Also

windowFrameTextColor (page 369)

windowFrameTextColor

Returns the system color used for the text in window frames.

```
public static NSColor windowFrameTextColor()
```

Discussion

For general information about system colors, see "Accessing System Colors".

See Also

windowFrameColor (page 369)

yellowColor

Returns an NSColor whose RGB value is 1.0, 1.0, 0.0 and whose alpha value is 1.0.

```
public static NSColor yellowColor()
```

See Also

yellowComponent (page 378)

Instance Methods

alphaComponent

Returns the receiver's alpha (opacity) component.

```
public float alphaComponent()
```

Discussion
Returns 1.0 (opaque) if the receiver has no alpha component.

blackComponent

Returns the receiver's black component.

```
public float blackComponent()
```

Discussion
Throws an exception if the receiver isn't a CMYK color.

blendedColorWithFractionOfColor

Creates and returns an NSColor whose component values are a weighted sum of the receiver's and *color*'s.

```
public NSColor blendedColorWithFractionOfColor(float fraction, NSColor color)
```

Discussion
The method converts *color* and a copy of the receiver to RGB, and then sets each component of the returned color to *fraction* of *color*'s value plus 1 – *fraction* of the receiver's. Returns `null` if the colors can't be converted.

blueComponent

Returns the receiver's blue component.

```
public float blueComponent()
```

Discussion
Throws an exception if the receiver isn't an RGB color.

brightnessComponent

Returns the brightness component of the HSB color equivalent to the receiver.

```
public float brightnessComponent()
```

Discussion
Throws an exception if the receiver isn't an RGB color.

catalogNameComponent

Returns the name of the catalog containing the receiver's name.

```
public String catalogNameComponent()
```

Discussion
Throws an exception if the receiver's color space isn't `NSGraphics.NamedColorSpace`.

See Also
`colorWithCatalogName` (page 358)
`colorNameComponent` (page 371)
`localizedCatalogNameComponent` (page 374)

colorNameComponent

Returns the receiver's name.

```
public String colorNameComponent()
```

Discussion
Throws an exception if the receiver's color space isn't `NSGraphics.NamedColorSpace`.

See Also
`colorWithCatalogName` (page 358)
`catalogNameComponent` (page 371)
`localizedCatalogNameComponent` (page 374)

colorSpace

Returns an object representing the color space of the receiver.

```
public NSColorSpace colorSpace()
```

Discussion
The returned NSColorSpace object may represent a custom color space.

Availability
Available in Mac OS X v10.4.

colorSpaceName

Returns the name of the receiver's color space.

```
public String colorSpaceName()
```

Discussion
This method should be implemented in subclasses of NSColor.

See Also
`colorUsingColorSpaceName` (page 372)
`colorUsingColorSpaceNameAndDevice` (page 372)

colorUsingColorSpace

Returns a new color object representing the color of the receiver in the specified color space *space*.

```
public NSColor colorUsingColorSpace(NSColorSpace space)
```

Discussion

In creating the new NSColor object, this method converts the receiver's color to an equivalent one in the new color space. Although the new color might have different component values, it looks the same as the original. The method returns the same NSColor object as the receiver if its color space is the same as the one specified. Returns `nil` if conversion is not possible.

Availability

Available in Mac OS X v10.4.

colorUsingColorSpaceName

Creates and returns an NSColor whose color is the same as the receiver's, except that the new NSColor is in the color space named *colorSpace*.

```
public NSColor colorUsingColorSpaceName(String colorSpace)
```

Discussion

If *colorSpace* is `null`, the most appropriate color space is used.

Returns `null` if the specified conversion cannot be done.

See Also

`colorSpaceName` (page 371)

colorUsingColorSpaceNameAndDevice

Creates and returns an 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*.

```
public NSColor colorUsingColorSpaceNameAndDevice(String colorSpace, NSDictionary
    deviceDescription)
```

Discussion

Device descriptions can be obtained from windows, screens, and printers with the `deviceDescription` method. If *colorSpace* is `null`, the most appropriate color space is used.

If *deviceDescription* is `null`, the current device (as obtained from the currently lockFocus'ed view's window or, if printing, the current printer) is used.

Returns `null` if the specified conversion cannot be done.

See Also

`colorSpaceName` (page 371)

`colorUsingColorSpaceName` (page 372)

colorWithAlphaComponent

Creates and returns an NSColor that has the same color space and component values as the receiver, except its alpha component is *alpha*.

```
public NSColor colorWithAlphaComponent(float alpha)
```

Discussion
If the receiver's color space doesn't include an alpha component, the receiver is returned. A subclass with explicit opacity components should override this method to return a color with the specified alpha.

See Also
`alphaComponent` (page 370)
`blendedColorWithFractionOfColor` (page 370)

components

Returns the components of the receiver as an array of `float` values.

```
public float[] components()
```

Discussion
You can invoke this method on NSColor objects created from custom color spaces to get the individual floating point components, including alpha. Throws an exception if the receiver doesn't have floating-point components. To find out how many components are in the array, send the receiver a `numberOfComponents` (page 375) message.

Availability
Available in Mac OS X v10.4.

See Also
`colorSpace` (page 371)

cyanComponent

Returns the receiver's cyan component.

```
public float cyanComponent()
```

Discussion
Throws an exception if the receiver isn't a CMYK color.

drawSwatchInRect

Draws the current color in the rectangle *rect*.

```
public void drawSwatchInRect(NSRect rect)
```

Discussion
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.

greenComponent

Returns the receiver's green component.

```
public float greenComponent()
```

Discussion
Throws an exception if the receiver isn't an RGB color.

highlightWithLevel

Returns an NSColor that represents a blend between the receiver and the highlight color returned by `highlightColor` (page 364).

```
public NSColor highlightWithLevel(float highlightLevel)
```

Discussion
The highlight color's contribution to the blend depends on *`highlightLevel`*, which should be a number from 0.0 through 1.0. (A *`highlightLevel`* below 0.0 is interpreted as 0.0 [the receiver]; a *`highlightLevel`* above 1.0 is interpreted as 1.0 [*`highlightLevel`*].)

Returns `null` if the colors can't be converted. Invoke this method when you want to brighten the receiving NSColor for use in highlights.

See Also
`shadowWithLevel` (page 377)

hueComponent

Returns the hue component of the HSB color equivalent to the receiver.

```
public float hueComponent()
```

Discussion
Throws an exception if the receiver isn't an RGB color.

localizedCatalogNameComponent

```
public String localizedCatalogNameComponent()
```

Discussion
Like `catalogNameComponent` (page 371), but returns a localized string. This string may be displayed in user interface items like color pickers.

See Also
`colorWithCatalogName` (page 358)
`colorNameComponent` (page 371)

localizedColorNameComponent

```
public String localizedColorNameComponent()
```

Discussion
Like colorNameComponent (page 371), but returns a localized string. This string may be displayed in user interface items like color pickers.

See Also
colorWithCatalogName (page 358)
catalogNameComponent (page 371)
colorNameComponent (page 371)
localizedCatalogNameComponent (page 374)

magentaComponent

Returns the receiver's magenta component.

```
public float magentaComponent()
```

Discussion
Throws an exception if the receiver isn't a CMYK color.

numberOfComponents

Returns the number of components in the receiver.

```
public int numberOfComponents()
```

Discussion
The floating-point components counted include alpha. Throws an exception if the receiver doesn't have floating-point components.

Availability
Available in Mac OS X v10.4

See Also
colorSpace (page 371)
components (page 373)

patternImage

Returns the image that the receiver is using as a pattern.

```
public NSImage patternImage()
```

Discussion
Throws an exception if the receiver doesn't have an image.

redComponent

Returns the receiver's red component.

```
public float redComponent()
```

Discussion

Throws an exception if the receiver isn't an RGB color.

saturationComponent

Returns the saturation component of the HSB color equivalent to the receiver.

```
public float saturationComponent()
```

Discussion

Throws an exception if the receiver isn't an RGB color.

set

Sets the color of subsequent drawing to the color that the receiver represents.

```
public void set()
```

Discussion

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` (page 370); if the color doesn't know about alpha, it's set to 1.0. This method should be implemented in subclasses.

setFill

Sets the fill color of subsequent drawing to the receiver's color.

```
public void setFill()
```

Discussion

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` (page 370); if the color doesn't know about alpha, it's set to 1.0.

Subclasses of NSColor should implement this method.

Availability

Available in Mac OS X v10.3 and later.

See Also

`setStroke` (page 376)

setStroke

Sets the stroke color of subsequent drawing to the receiver's color.

```
public void setStroke()
```

Discussion

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` (page 370); if the color doesn't know about alpha, it's set to 1.0.

Subclasses of NSColor should implement this method.

Availability

Available in Mac OS X v10.3 and later.

See Also

`setFill` (page 376)

shadowWithLevel

Returns an NSColor that represents a blend between the receiver and the shadow color returned by `shadowColor` (page 368).

```
public NSColor shadowWithLevel(float shadowLevel)
```

Discussion

The shadow color's contribution to the blend depends on *shadowLevel*, which should be a number from 0.0 through 1.0. (A *shadowLevel* below 0.0 is interpreted as 0.0 [the receiver]; a *shadowLevel* above 1.0 is interpreted as 1.0 [*shadowLevel*].)

Returns `null` if the colors can't be converted. Invoke this method when you want to darken the receiving NSColor for use in shadows.

See Also

`highlightWithLevel` (page 374)

whiteComponent

Returns the receiver's white component.

```
public float whiteComponent()
```

Discussion

Throws an exception if the receiver isn't a grayscale color.

writeToPasteboard

Writes the receiver's data to *pasteBoard*, unless *pasteBoard* doesn't support color data (in which case the method does nothing).

```
public void writeToPasteboard(NSPasteboard pasteBoard)
```

See Also

`colorFromPasteboard` (page 357)

yellowComponent

Returns the receiver's yellow component.

```
public float yellowComponent()
```

Discussion
Throws an exception if the receiver isn't a CMYK color.

Notifications

SystemColorsDidChangeNotification

Sent when the system colors have been changed (such as through a system control panel interface).

This notification contains no notification object and no *userInfo* dictionary.

NSColorList

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.application
Companion guide	Color Programming Topics for Cocoa

Overview

An NSColorList is an ordered list of NSColors, identified by keys. Instances of NSColorList, or more simply color lists, are used to manage named lists of NSColors. 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.

Tasks

Constructors

`NSColorList` (page 380)
Creates an empty NSColorList.

Getting All Color Lists

`availableColorLists` (page 381)
Returns an array of all NSColorLists found in the standard color list directories, including color catalogs (lists of colors identified only by name).

Getting a Color List by Name

`colorListNamed` (page 381)
Searches the array that's returned by `availableColorLists` (page 381) and returns the NSColorList named *`name`*, or `null` if no such color list exists.

`name` (page 382)
Returns the name of the receiver.

Managing Colors by Key

`allKeys` (page 381)
Returns an array of String objects that contains all the keys by which the NSColors are stored in the receiver.

`colorWithKey` (page 382)
Returns the NSColor associated with *`key`*, or `null` if there is none.

`insertColorForKeyAtIndex` (page 382)
Inserts *`color`* at the specified *`location`* in the receiver (numbered starting with 0).

`removeColorWithKey` (page 382)
Removes the color associated with *`key`* from the receiver.

`setColorForKey` (page 383)
Associates the specified NSColor with *`key`*.

Editing

`isEditable` (page 382)
Returns `true` if the receiver can be modified.

Writing and Removing Files

`removeFile` (page 383)
Removes the file from which the list was created, if the file is in a standard search path and owned by the user.

`writeToFile` (page 383)
If *`path`* is a directory, saves the receiver in a file named *listname*`.clr` in that directory (where *listname* is the name with which the receiver was initialized).

Constructors

NSColorList

Creates an empty NSColorList.

```
public NSColorlist()
```

Creates a color list, registering it under *`name`* if *`name`* isn't in use already.

```
public NSColorList(String name)
```

Discussion
This constructor invokes the constructor with two arguments, using `null` as the additional argument, indicating that the color list doesn't need to be initialized from a file.

Creates an NSColorList, registering it under *`name`* if *`name`* isn't in use already.

```
public NSColorList(String name, String path)
```

Discussion

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). A null path indicates the color list should be initialized with no colors.

Static Methods

availableColorLists

Returns an array of all NSColorLists found in the standard color list directories, including color catalogs (lists of colors identified only by name).

```
public static NSArray availableColorLists()
```

Discussion

Color lists created at runtime aren't included in this list unless they're saved into one of the standard color list directories.

See Also

colorListNamed (page 381)

colorListNamed

Searches the array that's returned by availableColorLists (page 381) and returns the NSColorList named *name*, or null if no such color list exists.

```
public static NSColorList colorListNamed(String name)
```

Discussion

name must not include the ".clr" suffix.

See Also

name (page 382)

Instance Methods

allKeys

Returns an array of String objects that contains all the keys by which the NSColors are stored in the receiver.

```
public NSArray allKeys()
```

Discussion

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

Returns the NSColor associated with *key*, or `null` if there is none.

```
public NSColor colorWithKey(String key)
```

insertColorForKeyAtIndex

Inserts *color* at the specified *location* in the receiver (numbered starting with 0).

```
public void insertColorForKeyAtIndex(NSColor color, String key, int location)
```

Discussion
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 `ColorListDidChangeNotification` (page 383) to the default notification center. It throws `ColorListNotEditableException` if the color list isn't editable.

See Also
`colorWithKey` (page 382)
`removeColorWithKey` (page 382)
`setColorForKey` (page 383)

isEditable

Returns `true` if the receiver can be modified.

```
public boolean isEditable()
```

Discussion
This result depends on the source of the list: If it came from a write-protected file, this method returns `false`.

name

Returns the name of the receiver.

```
public String name()
```

removeColorWithKey

Removes the color associated with *key* from the receiver.

```
public void removeColorWithKey(String key)
```

Discussion
This method does nothing if the receiver doesn't contain the key. This method posts `ColorListDidChangeNotification` (page 383) to the default notification center. It throws `ColorListNotEditableException` if the receiver is not editable.

See Also
`insertColorForKeyAtIndex` (page 382)
`setColorForKey` (page 383)

removeFile

Removes the file from which the list was created, if the file is in a standard search path and owned by the user.

```
public void removeFile()
```

Discussion
The receiver is removed from the list of available color lists returned by `availableColorLists` (page 381).

setColorForKey

Associates the specified NSColor with *key*.

```
public void setColorForKey(NSColor color, String key)
```

Discussion
If the list already contains *key*, this method sets the corresponding color to *color*; otherwise, it inserts *color* at the end of the list by invoking `insertColorForKeyAtIndex` (page 382).

See Also
`colorWithKey` (page 382)
`insertColorForKeyAtIndex` (page 382)
`removeColorWithKey` (page 382)

writeToFile

If *path* is a directory, saves the receiver in a file named *listname*`.clr` in that directory (where *listname* is the name with which the receiver was initialized).

```
public boolean writeToFile(String path)
```

Discussion
If *path* includes a filename, this method saves the file under that name. If *path* is `null`, this method saves the file as *listname*`.clr` in the user's private colorlists directory. Returns `true` upon success and `false` if it fails to write the file.

See Also
`removeFile` (page 383)

Notifications

ColorListDidChangeNotification

Posted whenever a color list changes. The notification object is the NSColorList object that changed. This notification does not contain a `userInfo` dictionary.

NSColorPanel

Inherits from	NSPanel : NSWindow : NSResponder : NSObject
Implements	NSCoding (NSResponder)
Package:	com.apple.cocoa.application
Companion guide	Color Programming Topics for Cocoa

Overview

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 interfaces, allows an application to add its own color selection modes. It allows the user to save swatches containing frequently used colors.

Tasks

Constructors

`NSColorPanel` (page 387)
Creates an empty NSColorPanel.

Creating the NSColorPanel

`sharedColorPanel` (page 389)
Returns the shared NSColorPanel, creating it if necessary.

`sharedColorPanelExists` (page 389)
Returns `true` if the NSColorPanel has been created already.

Setting Color Picker Modes

`setPickerMask` (page 388)
Accepts as the *mask* parameter one or more logically ORed color mode masks described in "Constants" (page 393).

`setPickerMode` (page 388)

Setting the NSColorPanel

`accessoryView` (page 389)
Returns the accessory view, or `null` if there is none.

`isContinuous` (page 390)
Returns whether the receiver continuously sends the action message to the target as the user manipulates the color picker.

`mode` (page 390)
Returns the color picker mode of the receiver.

`setAccessoryView` (page 390)
Sets the accessory view displayed in the receiver to *aView*.

`setAction` (page 391)
Sets the action message to *action*.

`setContinuous` (page 391)
Sets the receiver to send the action message to its target continuously as the color of the NSColorPanel is set by the user.

`setMode` (page 392)
Sets the mode of the receiver if *mode* is one of the modes allowed by the color mask.

`setShowsAlpha` (page 392)
Tells the receiver whether or not to show alpha values and an opacity slider, depending on the Boolean value *flag*.

`setTarget` (page 392)
Sets the target of the receiver to *target*.

`showsAlpha` (page 392)
Returns whether or not the receiver shows alpha values and an opacity slider.

Attaching a Color List

`attachColorList` (page 389)
Adds the list of NSColors specified in *colorList* to all the color pickers in the receiver that display color lists by invoking `attachColorList` (page 389) on all color pickers in the application.

`detachColorList` (page 390)
Removes the list of NSColors specified in *colorList* from all the color pickers in the receiver that display color lists by invoking `detachColorList` on all color pickers in the application.

Setting Color

`dragColor` (page 388)
Drags *color* into a destination view from *sourceView*

`setColor` (page 391)
Sets the color of the receiver to *color*.

Getting Color Information

`color` (page 390)
Returns the currently selected color in the receiver.

Responding to a color change

`changeColor` (page 394) *delegate method*

Constructors

NSColorPanel

Creates an empty NSColorPanel.

```
public NSColorPanel()
```

Creates a new NSColorPanel.

```
public NSColorPanel(NSRect contentRect, int styleMask, int backingType, boolean
    defer)
```

Discussion
The *contentRect* argument specifies the location and size of the panel's content area in screen coordinates. Note that the Window Server limits window position coordinates to ±16,000 and sizes to 10,000.

The *styleMask* argument specifies the panel's style. Either it can be `NSWindow.BorderlessWindowMask`, or it can contain any of the options described in NSWindow's "Constants" (page 1875), combined using the C bitwise OR operator.

Borderless windows display none of the usual peripheral elements and are generally useful only for display or caching purposes; you should normally not need to create them. Also, note that an NSWindow's style mask should include `NSWindow.TitledWindowMask` if it includes any of the others.

The *backingType* argument specifies how the drawing done in the panel is buffered by the object's window device, and possible values are described in NSWindow's "Constants" (page 1875).

The *defer* argument determines whether the window server creates a window device for the new panel immediately. If *defer* is `true`, it defers creating the window until the panel is moved onscreen. All display messages sent are postponed until the panel is created, just before it's moved onscreen. Deferring the creation of the window improves launch time and minimizes the virtual memory load on the window server.

The new panel creates an instance of NSView to be its default content view. You can replace it with your own object by using the `setContentView` (page 1858) method.

Creates a new NSColorPanel.

```
public NSColorPanel(NSRect contentRect, int styleMask, int backingType, boolean
    defer, NSScreen aScreen)
```

Discussion

This constructor is equivalent to the one above, except *contentRect* is specified relative to the lower-left corner of *aScreen*.

If *aScreen* is `null`, *contentRect* 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.

Static Methods

dragColor

Drags *color* into a destination view from *sourceView*

```
public static boolean dragColor(NSColor color, NSEvent anEvent, NSView sourceView)
```

Discussion

in response to *anEvent*. This method is usually invoked by the `mouseDown` method of *sourceView*. The dragging mechanism handles all subsequent events.

Because it is a static method, `dragColor` can be invoked whether or not the instance of NSColorPanel exists. Returns `true`.

setPickerMask

Accepts as the *mask* parameter one or more logically ORed color mode masks described in "Constants" (page 393).

```
public static void setPickerMask(int mask)
```

Discussion

This method determines which color selection modes will be available in an application's NSColorPanel. This method has an effect only before NSColorPanel is instantiated.

If you create a class that implements the color-picking interfaces (NSColorPickingDefault and NSColorPickingCustom), 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` (page 388)

setPickerMode

```
public static void setPickerMode(int mode)
```

Discussion
Sets the color panel's initial picker to *mode*, which may be one of the symbolic constants described in "Constants" (page 393). 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` (page 388)
`setMode` (page 392)

sharedColorPanel

Returns the shared NSColorPanel, creating it if necessary.

```
public static NSColorPanel sharedColorPanel()
```

sharedColorPanelExists

Returns `true` if the NSColorPanel has been created already.

```
public static boolean sharedColorPanelExists()
```

See Also
`sharedColorPanel` (page 389)

Instance Methods

accessoryView

Returns the accessory view, or `null` if there is none.

```
public NSView accessoryView()
```

See Also
`setAccessoryView` (page 390)

attachColorList

Adds the list of NSColors specified in *colorList* to all the color pickers in the receiver that display color lists by invoking `attachColorList` (page 389) on all color pickers in the application.

```
public void attachColorList(NSColorList colorList)
```

Discussion
An application should use this method to add an NSColorList saved with a document in its file package or in a directory other than NSColorList's standard search directories.

See Also
`detachColorList` (page 390)

color

Returns the currently selected color in the receiver.

```
public NSColor color()
```

See Also

`setColor` (page 391)

detachColorList

Removes the list of NSColors specified in *`colorList`* from all the color pickers in the receiver that display color lists by invoking `detachColorList` on all color pickers in the application.

```
public void detachColorList(NSColorList colorList)
```

Discussion

Your application should use this method to remove an NSColorList saved with a document in its file package or in a directory other than NSColorList's standard search directories.

See Also

`attachColorList` (page 389)

isContinuous

Returns whether the receiver continuously sends the action message to the target as the user manipulates the color picker.

```
public boolean isContinuous()
```

See Also

`setContinuous` (page 391)

mode

Returns the color picker mode of the receiver.

```
public int mode()
```

Discussion

The mode constants for the standard color pickers are listed in "Choosing the Color Pickers in a Color Panel".

See Also

`setPickerMode` (page 388)

`setMode` (page 392)

setAccessoryView

Sets the accessory view displayed in the receiver to *`aView`*.

```
public void setAccessoryView(NSView aView)
```

Discussion

The accessory view can be any custom view you want to display with NSColorPanel, such as a view offering color blends in a drawing program. The accessory view is displayed below the color picker and above the color swatches in the NSColorPanel. The NSColorPanel automatically resizes to accommodate the accessory view.

See Also

`accessoryView` (page 389)

setAction

Sets the action message to *`action`*.

```
public void setAction(NSSelector action)
```

Discussion

When you select a color in the color panel NSColorPanel sends its action to its target, provided that neither the action nor the target is `null`. The action is `NULL` by default.

See *Action Messages* for additional information on action messages.

See Also

`setTarget` (page 392)

setColor

Sets the color of the receiver to *`color`*.

```
public void setColor(NSColor color)
```

Discussion

This method posts a `ColorPanelColorDidChangeNotification` (page 394) with the receiver to the default notification center.

See Also

`color` (page 390)

setContinuous

Sets the receiver to send the action message to its target continuously as the color of the NSColorPanel is set by the user.

```
public void setContinuous(boolean flag)
```

Discussion

Send this message with *`flag`* set to `true` if, for example, you want to continuously update the color of the target.

See Also

`isContinuous` (page 390)

setMode

Sets the mode of the receiver if *mode* is one of the modes allowed by the color mask.

```
public void setMode(int mode)
```

Discussion

The color mask is set when you first create the shared instance of NSColorPanel for an application. *mode* may be one of the symbolic constants described in "Constants" (page 393).

See Also

setPickerMode (page 388)

mode (page 390)

setShowsAlpha

Tells the receiver whether or not to show alpha values and an opacity slider, depending on the Boolean value *flag*.

```
public void setShowsAlpha(boolean flag)
```

Discussion

Note that calling the NSColor method setIgnoresAlpha (page 367) with a value of true overrides any value set with this method.

See Also

showsAlpha (page 392)

setTarget

Sets the target of the receiver to *target*.

```
public void setTarget(Object target)
```

Discussion

When you select a color in the color panel NSColorPanel sends its action to its target, provided that neither the action nor the target is null. The target is null by default.

See Also

setAction (page 391)

setContinuous (page 391)

showsAlpha

Returns whether or not the receiver shows alpha values and an opacity slider.

```
public boolean showsAlpha()
```

Discussion

Note that calling the NSColor method setIgnoresAlpha (page 367) with a value of true overrides any value set with setShowsAlpha (page 392).

See Also
setShowsAlpha (page 392)

Constants

These constants specify which of the color modes the NSColorPanel can use. For more information, see "Choosing the Color Pickers in a Color Panel".

Constant	Description
ColorPanelGrayModeMask	Grayscale-alpha
ColorPanelRGBModeMask	Red-green-blue
ColorPanelCMYKModeMask	Cyan-yellow-magenta-black
ColorPanelHSBModeMask	Hue-saturation-brightness
ColorPanelCustomPaletteModeMask	Custom palette
ColorPanelColorListModeMask	Custom color list
ColorPanelWheelModeMask	Color wheel
ColorPanelCrayonModeMask	Crayons
ColorPanelAllModesMask	All of the above

These constants specify the active color mode used when an application's instance of NSColorPanel is masked for more than one color mode. For more information, see "Choosing the Color Pickers in a Color Panel".

Constant	Description
GrayModeColorPanel	Grayscale-alpha
RGBModeColorPanel	Red-green-blue
CMYKModeColorPanel	Cyan-yellow-magenta-black
HSBModeColorPanel	Hue-saturation-brightness
CustomPaletteModeColorPanel	Custom palette
ColorListModeColorPanel	Custom color list
WheelModeColorPanel	Color wheel
CrayonModeColorPanel	Crayons

Delegate Methods

changeColor

```
public void changeColor(Object sender)
```

Discussion
When the user selects a color in an NSColorPanel, the NSColorPanel sends a `changeColor` action message to the first responder. You can override this method in any responder that needs to respond to a color change. *`sender`* is the color panel.

Notifications

ColorPanelColorDidChangeNotification

Posted when the NSColorPanel's color is set, as when `setColor` (page 391) is invoked. The notification object is the notifying NSColorPanel. This notification does not contain a *`userInfo`* dictionary.

NSColorPicker

Inherits from NSObject
Implements NSColorPickingDefault

Package: com.apple.cocoa.application
Companion guide Color Programming Topics for Cocoa

Overview

NSColorPicker is an abstract superclass that implements the NSColorPickingDefault interface. The NSColorPickingDefault and NSColorPickingCustom interfaces define a way to add color pickers (custom user interfaces for color selection) to the NSColorPanel.

Interfaces Implemented

NSColorPickingDefault

- `alphaControlAddedOrRemoved` (page 1948)
- `attachColorList` (page 1948)
- `detachColorList` (page 397)
- `insertNewButtonImage` (page 1949)
- `provideNewButtonImage` (page 1949)
- `setMode` (page 1949)
- `viewSizeChanged` (page 399)

Tasks

Constructors

`NSColorPicker` (page 396)
Creates an empty NSColorPicker.

Getting the Color Panel

`colorPanel` (page 397)
Returns the NSColorPanel that owns the receiver.

Adding Button Images

`insertNewButtonImage` (page 397)
Sets *newButtonImage* as *buttonCell*'s image by invoking NSButtonCell's `setImage` (page 50) method.

`provideNewButtonImage` (page 398)
Returns the button image for the receiver.

Setting the Mode

`setMode` (page 398)
Does nothing. Override to set the color picker's mode.

Using Color Lists

`attachColorList` (page 397)
Does nothing. Override to attach a color list to a color picker.

`detachColorList` (page 397)
Does nothing. Override to detach a color list from a color picker.

Responding to View Changes

`viewSizeChanged` (page 399)
Does nothing. Override to respond to a size change.

`alphaControlAddedOrRemoved` (page 397)

Constructors

NSColorPicker

Creates an empty NSColorPicker.

```
public NSColorPicker()
```

Creates a color picker, setting its color panel to *owningColorPanel*, caching the *owningColorPanel* value so it can later be returned by the `colorPanel` (page 397) method.

```
public NSColorPicker(int mask, NSColorPanel owningColorPanel)
```

Discussion
Write your own version of this constructor in your superclass to respond to the values in *mask* or do other custom initialization. If you do write your own version, forward the message to `super` as part of the implementation.

Instance Methods

alphaControlAddedOrRemoved

```
public void alphaControlAddedOrRemoved(Object sender)
```

Discussion
Sent by the color panel when the opacity controls have been hidden or displayed. Invoked automatically when the NSColorPanel's opacity slider is added or removed; you never invoke this method directly.

If the receiver has its own opacity controls, it should hide or display them, depending on whether *sender*'s `showsAlpha` (page 392) method returns `false` or `true`.

attachColorList

Does nothing. Override to attach a color list to a color picker.

```
public void attachColorList(NSColorList colorList)
```

See Also
`detachColorList` (page 397)

colorPanel

Returns the NSColorPanel that owns the receiver.

```
public NSColorPanel colorPanel()
```

detachColorList

Does nothing. Override to detach a color list from a color picker.

```
public void detachColorList(NSColorList colorList)
```

See Also
`attachColorList` (page 397)

insertNewButtonImage

Sets *newButtonImage* as *buttonCell*'s image by invoking NSButtonCell's `setImage` (page 50) method.

```
public void insertNewButtonImage(NSImage newButtonImage, NSButtonCell buttonCell)
```

Discussion
Called by the color panel to insert a new image into the specified cell. Override this method to customize *newButtonImage* before insertion in *buttonCell*.

See Also
`provideNewButtonImage` (page 398)

provideNewButtonImage

Returns the button image for the receiver.

```
public NSImage provideNewButtonImage()
```

Discussion
The color panel will place this image in the mode button the user uses to select this picker. (This is the same image the color panel uses as an argument when sending the `insertNewButtonImage` (page 397) 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`".

See Also
`insertNewButtonImage` (page 397)

setMode

Does nothing. Override to set the color picker's mode.

```
public void setMode(int mode)
```

Discussion
Here are the standard color picking modes and mode constants:

Mode	Color Mode Constant
Grayscale-alpha	`NSColorPanel.GrayModeColorPanel`
Red-green-blue	`NSColorPanel.RGBModeColorPanel`
Cyan-yellow-magenta-black	`NSColorPanel.CMYKModeColorPanel`
Hue-saturation-brightness	`NSColorPanel.HSBModeColorPanel`
Custom palette	`NSColorPanel.CustomPaletteModeColorPanel`
Custom color list	`NSColorPanel.ColorListModeColorPanel`
Color wheel	`NSColorPanel.WheelModeColorPanel`

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 an NSImage file (TIFF or EPS) into the NSColorPanel, then select colors from the 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.

viewSizeChanged

Does nothing. Override to respond to a size change.

```
public void viewSizeChanged(Object sender)
```

NSColorSpace

Inherits from	NSColorSpace : NSObject
Implements	NSCoding
Package:	com.apple.cocoa.application
Availability	Available in Mac OS X v10.4 and later.
Companion guide	Color Programming Topics for Cocoa

Overview

The NSColorSpace class enables the creation of objects representing custom color spaces. You can make custom color spaces from ColorSync profiles or from ICC profiles. NSColorSpace also has factory methods that return objects representing the system color spaces.

You can send the `colorUsingColorSpace` (page 372) message to an NSColor object to convert it between two color spaces, either of which may be a custom color space.

Tasks

Constructors

`NSColorSpace` (page 402)
Creates an NSColorSpace object initialized with the ICC profile in *iccData*.

Getting a Named NSColorSpace Object

`deviceRGBColorSpace` (page 403)
Returns an NSColorSpace object representing a calibrated or device-dependent RGB color space.

`genericRGBColorSpace` (page 404)
Returns an NSColorSpace object representing a device-independent RGB grayscale.

`deviceCMYKColorSpace` (page 402)
Returns an NSColorSpace object representing a calibrated or device-dependent CMYK color space.

`genericCMYKColorSpace` (page 403)
Returns an NSColorSpace object representing a device-independent CMYK color space.

`deviceGrayColorSpace` (page 403)
Returns an NSColorSpace object representing a calibrated or device-dependent gray-scale color space.

`genericGrayColorSpace` (page 404)
Returns an NSColorSpace object representing a device-independent grayscale color space.

Accessing Color-space Data and Attributes

`colorSpaceModel` (page 404)
Returns the model on which the color space of the receiver is based.

`ICCProfileData` (page 405)
Returns the ICC profile data from which the receiver was created.

`localizedName` (page 405)
Returns the localized name of the receiver.

`numberOfColorComponents` (page 405)
Returns the number of components (excluding alpha) supported by the receiver.

Constructors

NSColorSpace

```
public NSColorSpace()
```

Discussion
Creates an NSColorSpace object.

Creates an NSColorSpace object initialized with the ICC profile in *iccData*.

```
NSColorSpace(NSData iccData)
```

Discussion
For information on ICC profiles, see the latest ICC specification at the International Color Consortium website (http://www.color.org/icc_specs2.html)

Static Methods

deviceCMYKColorSpace

Returns an NSColorSpace object representing a calibrated or device-dependent CMYK color space.

```
public static NSColorSpace deviceCMYKColorSpace()
```

Discussion
This color space has cyan, magenta, yellow, black, and alpha components. Typical devices that use the color-subtractive CMYK color space are color printers. This object corresponds to the Cocoa color space name `DeviceCMYKColorSpace`.

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

See Also
`genericCMYKColorSpace` (page 403)

deviceGrayColorSpace

Returns an NSColorSpace object representing a calibrated or device-dependent gray-scale color space.

```
public static NSColorSpace deviceGrayColorSpace()
```

Discussion
The color space also includes an alpha component. Typical devices that use this color space are grayscale printers and displays. This object corresponds to the Cocoa color space name `DeviceWhiteColorSpace`.

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

See Also
`genericGrayColorSpace` (page 404)

deviceRGBColorSpace

Returns an NSColorSpace object representing a calibrated or device-dependent RGB color space.

```
public static NSColorSpace deviceRGBColorSpace()
```

Discussion
This color space has red, green, blue, and alpha components. Typical devices that use the color-additive RGB color space are displays and scanners. This object corresponds to the Cocoa color space name `DeviceRGBColorSpace`.

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

See Also
`genericRGBColorSpace` (page 404)

genericCMYKColorSpace

Returns an NSColorSpace object representing a device-independent CMYK color space.

```
public static NSColorSpace genericCMYKColorSpace()
```

Discussion
This color space has cyan, magenta, yellow, black and alpha component.

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

See Also

deviceCMYKColorSpace (page 402)

genericGrayColorSpace

Returns an NSColorSpace object representing a device-independent grayscale color space.

```
public static NSColorSpace genericGrayColorSpace()
```

Discussion

The color space also includes an alpha component. This object corresponds to the Cocoa color space name `CalibratedWhiteColorSpace`.

Availability

Available in Mac OS X v10.4 and later.

See Also

deviceGrayColorSpace (page 403)

genericRGBColorSpace

Returns an NSColorSpace object representing a device-independent RGB grayscale.

```
public static NSColorSpace genericRGBColorSpace()
```

Discussion

This color-additive color space has red, green, blue, and alpha components. This object corresponds to the Cocoa color space name `CalibratedRGBColorSpace`.

Availability

Available in Mac OS X v10.4 and later.

See Also

deviceRGBColorSpace (page 403)

Instance Methods

colorSpaceModel

Returns the model on which the color space of the receiver is based.

```
public int colorSpaceModel()
```

Discussion

See "Constants" (page 405) for a list of valid `NSColorSpaceModel` constants.

Availability

Available in Mac OS X v10.4 and later.

ICCProfileData

Returns the ICC profile data from which the receiver was created.

```
public native NSData ICCProfileData()
```

Discussion
This method attempts to compute the profile data from a CMProfileRef object and returns null if it is unable to. For information on ICC profiles, see the latest ICC specification at the International Color Consortium website (http://www.color.org/icc_specs2.html).

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

localizedName

Returns the localized name of the receiver.

```
public String localizedName()
```

Discussion
Returns null if no localized name exists.

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

numberOfColorComponents

Returns the number of components (excluding alpha) supported by the receiver.

```
public int numberOfColorComponents()
```

Discussion
Returns zero if the receiver is not based on float components.

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

Constants

The following constants indentify the abstract model on which an NSColorSpace object is based. This constant is returned from colorSpaceModel (page 404) and is derived from the profile data encapsulated by the object.

Constant	Description
UnknownColorSpaceModel	This model is not known to NSColorSpace.
GrayColorSpaceModel	The grayscale color-space model. Can refer to both device-dependent and generic color space variants.

Constant	Description
RGBColorSpaceModel	The RGB (red green blue) color-space model. Can refer to both device-dependent and generic color space variants.
CMYKColorSpaceModel	The CYMK (cyan, yellow, magenta, black) color-space model. Can refer to both device-dependent and generic color space variants.
LABColorSpaceModel	The L*a*b* device-independent color-space model, which represents colors relative to a reference white point.
DeviceNColorSpaceModel	DeviceN is a color-space model from Adobe Systems, Inc. used in PostScript and PDF color specification.

NSColorWell

Inherits from NSControl : NSView : NSResponder : NSObject

Implements NSCoding (NSResponder)

Package: com.apple.cocoa.application

Companion guide Color Programming Topics for Cocoa

Overview

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. A color well is available from the Palettes panel of Interface Builder.

Tasks

Constructors

`NSColorWell` (page 408)
Creates an NSColorWell with a zero-sized frame rectangle.

Drawing

`drawWellInside` (page 409)
Draws the colored area inside the receiver at the location specified by *`insideRect`* without drawing borders.

Activating

`activate` (page 408)
Activates the receiver, displays the color panel, and makes the NSColorPanel's current color the same as its own.

`deactivate` (page 409)
Deactivates the receiver and redraws it.

`isActive` (page 409)
Returns `true` if the receiver is active, `false` otherwise.

Managing Color

`color` (page 409)
Returns the color of the receiver.

`setColor` (page 410)
Sets the color of the receiver to *`color`* and redraws the receiver.

`takeColorFrom` (page 410)
Changes the color of the receiver to that of *`sender`*.

Managing Borders

`isBordered` (page 409)
Returns `true` if the receiver is bordered, `false` otherwise.

`setBordered` (page 410)
Places or removes a border on the receiver, depending on *`bordered`*, and redraws the receiver.

Constructors

NSColorWell

Creates an NSColorWell with a zero-sized frame rectangle.

```
public NSColorWell()
```

Creates an NSColor with *`frameRect`* as its frame rectangle.

```
public NSColorWell(NSRect frameRect)
```

Instance Methods

activate

Activates the receiver, displays the color panel, and makes the NSColorPanel's current color the same as its own.

```
public void activate(boolean exclusive)
```

Discussion
Redraws the receiver. An active color well will have its color updated when the NSColorPanel's current color changes. Any color well that shows its border highlights the border when it's active.

If *`exclusive`* is `true`, deactivates any other color wells; if `false`, keeps them active. If a color panel is active with *`exclusive`* set to `true` and another is subsequently activated with *`exclusive`* set to `false`, the exclusive setting of the first panel is ignored.

See Also
`deactivate` (page 409)
`isActive` (page 409)

color

Returns the color of the receiver.

```
public NSColor color()
```

See Also
`setColor` (page 410)
`takeColorFrom` (page 410)

deactivate

Deactivates the receiver and redraws it.

```
public void deactivate()
```

See Also
`activate` (page 408)
`isActive` (page 409)

drawWellInside

Draws the colored area inside the receiver at the location specified by *insideRect* without drawing borders.

```
public void drawWellInside(NSRect insideRect)
```

isActive

Returns `true` if the receiver is active, `false` otherwise.

```
public boolean isActive()
```

isBordered

Returns `true` if the receiver is bordered, `false` otherwise.

```
public boolean isBordered()
```

See Also
`setBordered` (page 410)

setBordered

Places or removes a border on the receiver, depending on *bordered*, and redraws the receiver.

```
public void setBordered(boolean bordered)
```

See Also

isBordered (page 409)

setColor

Sets the color of the receiver to *color* and redraws the receiver.

```
public void setColor(NSColor color)
```

See Also

color (page 409)

takeColorFrom (page 410)

takeColorFrom

Changes the color of the receiver to that of *sender*.

```
public void takeColorFrom(Object sender)
```

See Also

color (page 409)

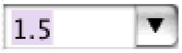
setColor (page 410)

NSComboBox

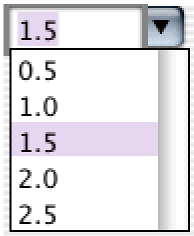
Inherits from	NSTextField : NSControl : NSView : NSResponder : NSObject
Implements	NSCoding (NSResponder)
Package:	com.apple.cocoa.application
Companion guide	Combo Box Programming Topics

Overview

An NSComboBox is a kind of NSControl that allows you to either enter text directly (as you would with an NSTextField) or click the attached arrow at the right of the combo box and select from a displayed ("pop-up") list of items. It normally looks like this:



When you click the downward-pointing arrow at the right side of the text field, the pop-up list appears, like this:



NSComboBox uses NSComboBoxCell (page 427) to implement its user interface.

Also see the NSComboBox.DataSource (page 1951) interface, which declares the methods that an NSComboBox uses to access the contents of its data source object.

Tasks

Constructors

`NSComboBox` (page 414)
  Creates an NSComboBox with a zero-sized frame rectangle.

Setting Display Attributes

`hasVerticalScroller` (page 416)
  Returns `true` if the receiver will display a vertical scroller.

`intercellSpacing` (page 417)
  Returns the horizontal and vertical spacing between cells in the receiver's pop-up list.

`isButtonBordered` (page 417)
  Returns whether the combo box button is set to display a border.

`itemHeight` (page 417)
  Returns the height of each item in the receiver's pop-up list.

`numberOfVisibleItems` (page 418)
  Returns the maximum number of items visible at any one time in the pop-up list.

`setButtonBordered` (page 421)
  Determines whether the button in the combo box is displayed with a border.

`setHasVerticalScroller` (page 422)
  Determines according to *`flag`* whether the receiver displays a vertical scroller.

`setIntercellSpacing` (page 422)
  Sets the width and height between pop-up list items to the values in *`aSize`*.

`setItemHeight` (page 423)
  Sets the height for items to *`itemHeight`*.

`setNumberOfVisibleItems` (page 423)
  Sets the maximum number of items that will be visible at one time in the receiver's pop-up list to *`visibleItems`*.

Setting a Data Source

`dataSource` (page 415)
  Returns the object that provides the data displayed in the receiver's pop-up list.

`setDataSource` (page 422)
  Sets the receiver's data source to *`aSource`*.

`setUsesDataSource` (page 423)
  Sets according to *`flag`* whether the receiver uses an external data source (specified by `setDataSource` (page 422)) to populate the receiver's pop-up list.

`usesDataSource` (page 423)
: Returns `true` if the receiver uses an external data source to populate the receiver's pop-up list, `false` if it uses an internal item list.

Working with an Internal List

`addItemsWithObjectValues` (page 415)
: Adds multiple objects to the end of the receiver's internal item list.

`addItemWithObjectValue` (page 415)
: Adds *`anObject`* to the end of the receiver's internal item list.

`insertItemWithObjectValueAtIndex` (page 417)
: Inserts *`anObject`* at *`index`* in the receiver's internal item list, shifting the previous item at *`index`*—along with all following items—down one slot to make room.

`objectValues` (page 419)
: Returns as an array the receiver's internal item list.

`removeAllItems` (page 419)
: Removes all items from the receiver's internal item list.

`removeItemAtIndex` (page 420)
: Removes the object at *`index`* from the receiver's internal item list and moves all items beyond *`index`* up one slot to fill the gap.

`removeItemWithObjectValue` (page 420)
: Removes all occurrences of *`anObject`* from the receiver's internal item list.

`numberOfItems` (page 418)
: Returns the total number of items in the pop-up list.

Manipulating the Displayed List

`indexOfItemWithObjectValue` (page 416)
: Searches the receiver's internal item list for *`anObject`* and returns the lowest index whose corresponding value is equal to *`anObject`*.

`itemObjectValueAtIndex` (page 418)
: Returns the object located at *`index`* within the receiver's internal item list.

`noteNumberOfItemsChanged` (page 418)
: Informs the receiver that the number of items in its data source has changed, allowing the receiver to update the scrollers in its displayed pop-up list without actually reloading data into the receiver.

`reloadData` (page 419)
: Marks the receiver as needing redisplay, so that it will reload the data for visible pop-up items and draw the new values.

`scrollItemAtIndexToTop` (page 420)
: Scrolls the receiver's pop-up list vertically so that the item at *`index`* is as close to the top as possible.

`scrollItemAtIndexToVisible` (page 420)
: Scrolls the receiver's pop-up list vertically so that the item at *`index`* is visible.

Manipulating the Selection

`deselectItemAtIndex` (page 416)
Deselects the pop-up list item at *index* if it's selected.

`indexOfSelectedItem` (page 416)
Returns the index of the last item selected from the receiver's pop-up list, or –1 if no item is selected.

`objectValueOfSelectedItem` (page 419)
Returns the object from the receiver's internal item list corresponding to the last item selected from the pop-up list, or `null` if no item is selected.

`selectItemAtIndex` (page 420)
Selects the pop-up list row at *index*.

`selectItemWithObjectValue` (page 421)
Selects the first pop-up list item that corresponds to *anObject*.

Completing the Text Field

`completes` (page 415)
Returns `true` if the receiver tries to complete what the user types in the text field.

`setCompletes` (page 421)
Sets whether the receiver tries to complete what the user types in the text field.

Displaying and dismissing a combo box

`comboBoxWillDismiss` (page 424) *delegate method*
Informs the delegate that the pop-up list is about to be dismissed.

`comboBoxWillPopUp` (page 424) *delegate method*
Informs the delegate that the pop-up list is about to be displayed.

Changing selection

`comboBoxSelectionDidChange` (page 424) *delegate method*
Informs the delegate that the pop-up list selection has finished changing.

`comboBoxSelectionIsChanging` (page 424) *delegate method*
Informs the delegate that the pop-up list selection is changing.

Constructors

NSComboBox

Creates an NSComboBox with a zero-sized frame rectangle.

```
public NSComboBox()
```

Creates an NSComboBox with *frameRect* as its frame rectangle.

```
public NSComboBox(NSRect frameRect)
```

Instance Methods

addItemsWithObjectValues

Adds multiple objects to the end of the receiver's internal item list.

```
public void addItemsWithObjectValues(NSArray objects)
```

Discussion
This method logs a warning if usesDataSource (page 423) returns true.

addItemWithObjectValue

Adds *anObject* to the end of the receiver's internal item list.

```
public void addItemWithObjectValue(Object anObject)
```

Discussion
This method logs a warning if usesDataSource (page 423) returns true.

completes

Returns true if the receiver tries to complete what the user types in the text field.

```
public boolean completes()
```

Discussion
It returns false otherwise.

See Also
setCompletes (page 421)

dataSource

Returns the object that provides the data displayed in the receiver's pop-up list.

```
public Object dataSource()
```

Discussion
This method logs a warning if usesDataSource (page 423) returns false. See the class description and the NSComboBox.DataSource (page 1951) interface specification for more information on combo box data source objects.

deselectItemAtIndex

Deselects the pop-up list item at `index` if it's selected.

```
public void deselectItemAtIndex(int index)
```

Discussion

If the selection does in fact change, this method posts a `ComboBoxSelectionDidChangeNotification` (page 424) to the default notification center.

See Also

`indexOfSelectedItem` (page 416)

`numberOfItems` (page 418)

`selectItemAtIndex` (page 420)

hasVerticalScroller

Returns `true` if the receiver will display a vertical scroller.

```
public boolean hasVerticalScroller()
```

Discussion

Note that the scroller will be displayed even if the pop-up list contains fewer items than will fit in the area specified for display. Returns `false` if the receiver won't display a vertical scroller.

See Also

`numberOfItems` (page 418)

`numberOfVisibleItems` (page 418)

indexOfItemWithObjectValue

Searches the receiver's internal item list for `anObject` and returns the lowest index whose corresponding value is equal to `anObject`.

```
public int indexOfItemWithObjectValue(Object anObject)
```

Discussion

Objects are considered equal if `equals` returns `true`. If none of the objects in the receiver's internal item list are equal to `anObject`, `indexOfItemWithObjectValue` returns `NSArray.NotFound`. This method logs a warning if `usesDataSource` (page 423) returns `true`.

See Also

`selectItemWithObjectValue` (page 421)

indexOfSelectedItem

Returns the index of the last item selected from the receiver's pop-up list, or –1 if no item is selected.

```
public int indexOfSelectedItem()
```

Discussion

Note that nothing is initially selected in a newly initialized combo box.

See Also

objectValueOfSelectedItem (page 419)

insertItemWithObjectValueAtIndex

Inserts *anObject* at *index* in the receiver's internal item list, shifting the previous item at *index*—along with all following items—down one slot to make room.

```
public void insertItemWithObjectValueAtIndex(Object anObject, int index)
```

Discussion

This method logs a warning if usesDataSource (page 423) returns true.

See Also

addItemWithObjectValue (page 415)

numberOfItems (page 418)

intercellSpacing

Returns the horizontal and vertical spacing between cells in the receiver's pop-up list.

```
public NSSize intercellSpacing()
```

Discussion

The default spacing is (3.0, 2.0).

See Also

itemHeight (page 417)

numberOfVisibleItems (page 418)

isButtonBordered

Returns whether the combo box button is set to display a border.

```
public boolean isButtonBordered()
```

Discussion

The return value is true if the button has a border.

Availability

Available in Mac OS X v10.3 and later.

See Also

setButtonBordered (page 421)

itemHeight

Returns the height of each item in the receiver's pop-up list.

```
public float itemHeight()
```

Discussion
The default item height is 16.0.

See Also
`intercellSpacing` (page 417)
`numberOfVisibleItems` (page 418)

itemObjectValueAtIndex

Returns the object located at *`index`* within the receiver's internal item list.

```
public Object itemObjectValueAtIndex(int index)
```

Discussion
If *`index`* is beyond the end of the list, a `RangeException` is thrown. This method logs a warning if `usesDataSource` (page 423) returns `true`.

See Also
`objectValueOfSelectedItem` (page 419)

noteNumberOfItemsChanged

Informs the receiver that the number of items in its data source has changed, allowing the receiver to update the scrollers in its displayed pop-up list without actually reloading data into the receiver.

```
public void noteNumberOfItemsChanged()
```

Discussion
This method is particularly useful for a data source that continually receives data in the background over a period of time, in which case the NSComboBox can remain responsive to the user while the data is received.

See the NSComboBox.DataSource (page 1951) interface specification for information on the messages an NSComboBox sends to its data source.

See Also
`reloadData` (page 419)

numberOfItems

Returns the total number of items in the pop-up list.

```
public int numberOfItems()
```

See Also
`numberOfVisibleItems` (page 418)
`numberOfItemsInComboBox` (page 1952) (NSComboBoxDataSource interface)

numberOfVisibleItems

Returns the maximum number of items visible at any one time in the pop-up list.

```
public int numberOfVisibleItems()
```

See Also
`numberOfItems` (page 418)

objectValueOfSelectedItem

Returns the object from the receiver’s internal item list corresponding to the last item selected from the pop-up list, or `null` if no item is selected.

```
public Object objectValueOfSelectedItem()
```

Discussion
Note that nothing is initially selected in a newly initialized combo box. This method logs a warning if `usesDataSource` (page 423) returns `true`.

See Also
`indexOfSelectedItem` (page 416)
`comboBoxValueForItemAtIndex` (page 1952) (NSComboBoxDataSource interface)

objectValues

Returns as an array the receiver’s internal item list.

```
public NSArray objectValues()
```

Discussion
This method logs a warning if `usesDataSource` (page 423) returns `true`.

reloadData

Marks the receiver as needing redisplay, so that it will reload the data for visible pop-up items and draw the new values.

```
public void reloadData()
```

See Also
`noteNumberOfItemsChanged` (page 418)

removeAllItems

Removes all items from the receiver’s internal item list.

```
public void removeAllItems()
```

Discussion
This method logs a warning if `usesDataSource` (page 423) returns `true`.

See Also
`objectValues` (page 419)

removeItemAtIndex

Removes the object at *index* from the receiver's internal item list and moves all items beyond *index* up one slot to fill the gap.

```
public void removeItemAtIndex(int index)
```

Discussion

This method throws a `RangeException` if *index* is beyond the end of the list and logs a warning if `usesDataSource` (page 423) returns `true`.

removeItemWithObjectValue

Removes all occurrences of *anObject* from the receiver's internal item list.

```
public void removeItemWithObjectValue(Object anObject)
```

Discussion

Objects are considered equal if `equals` returns `true`. This method logs a warning if `usesDataSource` (page 423) returns `true`.

See Also

`indexOfItemWithObjectValue` (page 416)

scrollItemAtIndexToTop

Scrolls the receiver's pop-up list vertically so that the item at *index* is as close to the top as possible.

```
public void scrollItemAtIndexToTop(int index)
```

Discussion

The pop-up list need not be displayed at the time this method is invoked.

scrollItemAtIndexToVisible

Scrolls the receiver's pop-up list vertically so that the item at *index* is visible.

```
public void scrollItemAtIndexToVisible(int index)
```

Discussion

The pop-up list need not be displayed at the time this method is invoked.

selectItemAtIndex

Selects the pop-up list row at *index*.

```
public void scrollItemAtIndexToVisible(int index)
```

Discussion

Posts a `ComboBoxSelectionDidChangeNotification` (page 424) to the default notification center if the selection does in fact change. Note that this method does not alter the contents of the combo box's text field—see "Setting the Combo Box's Value" for more information.

See Also

`setObjectValue` (page 459) (NSControl)

selectItemWithObjectValue

Selects the first pop-up list item that corresponds to *`anObject`*.

```
public void selectItemWithObjectValue(Object anObject)
```

Discussion

Objects are considered equal if `equals` returns `true`. This method logs a warning if `usesDataSource` (page 423) returns `true`. Posts a `ComboBoxSelectionDidChangeNotification` (page 424) to the default notification center if the selection does in fact change. Note that this method doesn't alter the contents of the combo box's text field—see "Setting the Combo Box's Value" for more information.

See Also

`setObjectValue` (page 459) (NSControl)

setButtonBordered

Determines whether the button in the combo box is displayed with a border.

```
public void setButtonBordered(boolean flag)
```

Discussion

For example, it is often useful when using a combo box in an NSTableView to display the button without the border. Set *`flag`* to `true` to display a border.

Availability

Available in Mac OS X v10.3 and later.

See Also

`isButtonBordered` (page 417)

setCompletes

Sets whether the receiver tries to complete what the user types in the text field.

```
public void setCompletes(boolean completes)
```

Discussion

By default, the combo box does not try to.

If *completes* is true, every time the user adds characters to the end of the text field, the combo box calls the NSComboBoxCell method completedString (page 431). If completedString (page 431) returns a string that's longer than the existing string, the combo box replaces the existing string with the returned string and selects the additional characters. If the user is deleting characters or adds characters somewhere besides the end of the string, the combo box does not try to complete it.

See Also

completes (page 415)

setDataSource

Sets the receiver's data source to *aSource*.

```
public void setDataSource(Object aSource)
```

Discussion

aSource should implement the appropriate methods of the NSComboBoxDataSource interface. This method doesn't automatically set usesDataSource (page 423) to false and in fact logs a warning if usesDataSource (page 423) returns false.

This method logs a warning if *aSource* doesn't respond to either numberOfItemsInComboBox (page 1952) or comboBoxValueForItemAtIndex (page 1952).

See Also

setUsesDataSource (page 423)

setHasVerticalScroller

Determines according to *flag* whether the receiver displays a vertical scroller.

```
public void setHasVerticalScroller(boolean flag)
```

Discussion

By default, *flag* is true. If *flag* is false and the combo box has more list items (either in its internal item list or from its data source) than are allowed by numberOfVisibleItems (page 418), only a subset will be displayed. NSComboBox's scroll... methods can be used to position this subset within the pop-up list.

Note that if flag is true, a scroller will be displayed even if the combo box has fewer list items than are allowed by numberOfVisibleItems (page 418).

See Also

numberOfItems (page 418)

scrollItemAtIndexToTop (page 420)

scrollItemAtIndexToVisible (page 420)

setIntercellSpacing

Sets the width and height between pop-up list items to the values in *aSize*.

```
public void setIntercellSpacing(NSSize aSize)
```

Discussion

The default intercell spacing is (3.0, 2.0).

See Also

`setItemHeight` (page 423)

`setNumberOfVisibleItems` (page 423)

setItemHeight

Sets the height for items to *`itemHeight`*.

```
public void setItemHeight(float itemHeight)
```

See Also

`setIntercellSpacing` (page 422)

`setNumberOfVisibleItems` (page 423)

setNumberOfVisibleItems

Sets the maximum number of items that will be visible at one time in the receiver's pop-up list to *`visibleItems`*.

```
public void setNumberOfVisibleItems(int visibleItems)
```

See Also

`numberOfItems` (page 418)

`setItemHeight` (page 423)

`setIntercellSpacing` (page 422)

setUsesDataSource

Sets according to *`flag`* whether the receiver uses an external data source (specified by `setDataSource` (page 422)) to populate the receiver's pop-up list.

```
public void setUsesDataSource(boolean flag)
```

usesDataSource

Returns `true` if the receiver uses an external data source to populate the receiver's pop-up list, `false` if it uses an internal item list.

```
public boolean usesDataSource()
```

See Also

`dataSource` (page 415)

Delegate Methods

comboBoxSelectionDidChange

Informs the delegate that the pop-up list selection has finished changing.

```
public abstract void comboBoxSelectionDidChange(NSNotification notification)
```

Discussion
The name of *notification* is `ComboBoxSelectionDidChangeNotification` (page 424).

comboBoxSelectionIsChanging

Informs the delegate that the pop-up list selection is changing.

```
public abstract void comboBoxSelectionIsChanging(NSNotification notification)
```

Discussion
The name of *notification* is `ComboBoxSelectionIsChangingNotification` (page 425).

comboBoxWillDismiss

Informs the delegate that the pop-up list is about to be dismissed.

```
public abstract void comboBoxWillDismiss(NSNotification notification)
```

Discussion
The name of *notification* is `ComboBoxWillDismissNotification` (page 425).

comboBoxWillPopUp

Informs the delegate that the pop-up list is about to be displayed.

```
public abstract void comboBoxWillPopUp(NSNotification notification)
```

Discussion
The name of *notification* is `ComboBoxWillPopUpNotification` (page 425).

Notifications

ComboBoxSelectionDidChangeNotification

Posted after the NSComboBox's pop-up list selection changes. The notification object is the NSComboBox whose selection changed. This notification does not contain a *userInfo* dictionary.

ComboBoxSelectionIsChangingNotification

Posted whenever the NSComboBox's pop-up list selection is changing. The notification object is the NSComboBox whose selection is changing. This notification does not contain a `userInfo` dictionary.

ComboBoxWillDismissNotification

Posted whenever the NSComboBox's pop-up list is about to be dismissed. The notification object is the NSComboBox whose pop-up list will be dismissed. This notification does not contain a `userInfo` dictionary.

ComboBoxWillPopUpNotification

Posted whenever the NSComboBox's pop-up list is going to be displayed. The notification object is the NSComboBox whose pop-up window will be displayed. This notification does not contain a `userInfo` dictionary.

NSComboBoxCell

Inherits from	NSTextFieldCell : NSActionCell : NSCell : NSObject
Implements	NSCoding (NSCell)
Package:	com.apple.cocoa.application
Companion guide	Combo Box Programming Topics

Overview

NSComboBoxCell is a subclass of NSTextFieldCell used to implement the user interface of "combo boxes" (see NSComboBox (page 411) for information on how combo boxes look and work). The NSComboBox subclass of NSTextField uses a single NSComboBoxCell, and essentially all of NSComboBox's methods simply invoke the corresponding NSComboBoxCell method.

Also see the NSComboBoxCell.DataSource (page 1953) interface, which declares the methods that an NSComboBoxCell uses to access the contents of its data source object.

Tasks

Constructors

NSComboBoxCell (page 430)
Creates an empty NSComboBoxCell.

Setting Display Attributes

hasVerticalScroller (page 432)
Returns true if the receiver will display a vertical scroller.

isButtonBordered (page 433)
Returns whether the combo box button is set to display a border.

intercellSpacing (page 433)
Returns the horizontal and vertical spacing between cells in the receiver's pop-up list.

itemHeight (page 433)
Returns the height of each item in the receiver's pop-up list.

`numberOfVisibleItems` (page 434)
: Returns the maximum number of items visible at any one time in the pop-up list.

`setButtonBordered` (page 437)
: Determines whether the button in the combo box is displayed with a border.

`setHasVerticalScroller` (page 438)
: Determines according to *flag* whether the receiver displays a vertical scroller.

`setIntercellSpacing` (page 438)
: Sets the width and height between pop-up list items to the values in *aSize*.

`setItemHeight` (page 438)
: Sets the height for items to *itemHeight*.

`setNumberOfVisibleItems` (page 439)
: Sets the maximum number of items that will be visible at one time in the receiver's pop-up list to *visibleItems*.

Setting a Data Source

`dataSource` (page 431)
: Returns the object that provides the data displayed in the receiver's pop-up list.

`setDataSource` (page 437)
: Sets the receiver's data source to *aSource*.

`setUsesDataSource` (page 439)
: Sets according to *flag* whether the receiver uses an external data source (specified by `setDataSource` (page 437)) to populate the receiver's pop-up list.

`usesDataSource` (page 439)
: Returns `true` if the receiver uses an external data source to populate the receiver's pop-up list, `false` if it uses an internal item list.

Working with an Internal List

`addItemsWithObjectValues` (page 430)

`addItemWithObjectValue` (page 430)
: Adds *anObject* to the end of the receiver's internal item list.

`insertItemWithObjectValueAtIndex` (page 432)
: Inserts *anObject* at *index* in the receiver's internal item list, shifting the previous item at *index*—along with all following items—down one slot to make room.

`objectValues` (page 435)
: Returns as an array the receiver's internal item list.

`removeAllItems` (page 435)
: Removes all items from the receiver's internal item list.

`removeItemAtIndex` (page 435)
: Removes the object at *index* from the receiver's internal item list and moves all items beyond *index* up one slot to fill the gap.

`removeItemWithObjectValue` (page 436)
: Removes all occurrences of *anObject* from the receiver's internal item list.

`numberOfItems` (page 434)
: Returns the total number of items in the pop-up list.

Manipulating the Displayed List

`indexOfItemWithObjectValue` (page 432)
: Searches the receiver's internal item list for *anObject* and returns the lowest index whose corresponding value is equal to *anObject*.

`itemObjectValueAtIndex` (page 434)
: Returns the object located at *index* within the receiver's internal item list.

`noteNumberOfItemsChanged` (page 434)
: Informs the receiver that the number of items in its data source has changed, allowing the receiver to update the scrollers in its displayed pop-up list without actually reloading data into the receiver.

`reloadData` (page 435)
: Marks the receiver as needing redisplay, so that it will reload the data for visible pop-up items and draw the new values.

`scrollItemAtIndexToTop` (page 436)
: Scrolls the receiver's pop-up list vertically so that the item specified by *index* is as close to the top as possible.

`scrollItemAtIndexToVisible` (page 436)
: Scrolls the receiver's pop-up list vertically so that the item specified by *index* is visible.

Manipulating the Selection

`deselectItemAtIndex` (page 431)
: Deselects the pop-up list item at *index* if it's selected.

`indexOfSelectedItem` (page 432)
: Returns the index of the last item selected from the receiver's pop-up list, or –1 if no item is selected.

`objectValueOfSelectedItem` (page 435)
: Returns the object from the receiver's internal item list corresponding to the last item selected from the pop-up list, or `null` if no item is selected.

`selectItemAtIndex` (page 436)
: Selects the pop-up list row at *index*.

`selectItemWithObjectValue` (page 437)
: Selects the first pop-up list item that corresponds to *anObject*.

Completing the Text Field

`completedString` (page 431)
: Returns a string from the receiver's pop-up list that starts with the *substring*, or `null` if there is no such string.

`completes` (page 431)
    Returns `true` if the receiver tries to complete what the user types in the text field.

`setCompletes` (page 437)
    Sets whether the receiver tries to complete what the user types in the text field.

Constructors

NSComboBoxCell

Creates an empty NSComboBoxCell.

```
public NSComboBoxCell()
```

Creates an NSCell initialized with *aString* and set to have the cell's default menu.

```
public NSComboBoxCell(String aString)
```

Creates an NSCell initialized with *anImage* and set to have the cell's default menu.

```
public NSComboBoxCell(NSImage anImage)
```

Discussion
If *anImage* is `null`, no image is set.

Instance Methods

addItemsWithObjectValues

```
public void addItemsWithObjectValues(NSArray objects)
```

Discussion
Adds multiple *objects* to the end of the receiver's internal item list. This method logs a warning if `usesDataSource` (page 439) returns `true`.

addItemWithObjectValue

Adds *anObject* to the end of the receiver's internal item list.

```
public void addItemWithObjectValue(Object anObject)
```

Discussion
This method logs a warning if `usesDataSource` (page 439) returns `true`.

completedString

Returns a string from the receiver's pop-up list that starts with the *substring*, or null if there is no such string.

```
public String completedString(String substring)
```

Discussion

The argument *substring* is what the user entered in the combo box's text field. The default implementation of this method first checks whether the combo box uses a data source and whether the data source responds to comboBoxCompletedString (page 1951) or comboBoxCellCompletedString (page 1953). If so, the combo box cell returns that method's return value. Otherwise, this method goes through the combo box's items one by one and returns an item that starts with *substring*.

Override this method only if your subclass completes strings differently. The overriding method does not need to call the superclass's method. Generally, you do not need to call this method directly.

completes

Returns true if the receiver tries to complete what the user types in the text field.

```
public boolean completes()
```

Discussion

It returns false otherwise.

See Also

setCompletes (page 437)

dataSource

Returns the object that provides the data displayed in the receiver's pop-up list.

```
public Object dataSource()
```

Discussion

This method logs a warning if usesDataSource (page 439) returns false. See the class description and the NSComboBoxCell.DataSource (page 1953) interface specification for more information on combo box cell data source objects.

deselectItemAtIndex

Deselects the pop-up list item at *index* if it's selected.

```
public void deselectItemAtIndex(int index)
```

Discussion

If the selection does in fact change, this method posts a ComboBoxSelectionDidChangeNotification (page 424) to the default notification center.

See Also

indexOfSelectedItem (page 432)

`numberOfItems` (page 434)
`selectItemAtIndex` (page 436)

hasVerticalScroller

Returns `true` if the receiver will display a vertical scroller.

```
public boolean hasVerticalScroller()
```

Discussion

Note that the scroller will be displayed even if the pop-up list contains fewer items than will fit in the area specified for display. Returns `false` if the receiver won't display a vertical scroller.

See Also

`numberOfItems` (page 434)
`numberOfVisibleItems` (page 434)

indexOfItemWithObjectValue

Searches the receiver's internal item list for *`anObject`* and returns the lowest index whose corresponding value is equal to *`anObject`*.

```
public int indexOfItemWithObjectValue(Object anObject)
```

Discussion

Objects are considered equal if `equals` returns `true`. If none of the objects in the receiver's internal item list is equal to *`anObject`*, `indexOfItemWithObjectValue` returns `NSArray.NotFound`. This method logs a warning if `usesDataSource` (page 439) returns `true`.

See Also

`selectItemWithObjectValue` (page 437)

indexOfSelectedItem

Returns the index of the last item selected from the receiver's pop-up list, or –1 if no item is selected.

```
public int indexOfSelectedItem()
```

Discussion

Note that nothing is initially selected in a newly initialized combo box cell.

See Also

`objectValueOfSelectedItem` (page 435)

insertItemWithObjectValueAtIndex

Inserts *`anObject`* at *`index`* in the receiver's internal item list, shifting the previous item at *`index`*—along with all following items—down one slot to make room.

```
public void insertItemWithObjectValueAtIndex(Object anObject, int index)
```

Discussion

This method logs a warning if usesDataSource (page 439) returns true.

See Also

addItemWithObjectValue (page 430)

numberOfItems (page 434)

intercellSpacing

Returns the horizontal and vertical spacing between cells in the receiver's pop-up list.

```
public NSSize intercellSpacing()
```

Discussion

The default spacing is (3.0, 2.0).

See Also

itemHeight (page 433)

numberOfVisibleItems (page 434)

isButtonBordered

Returns whether the combo box button is set to display a border.

```
public boolean isButtonBordered()
```

Discussion

The return value is true if the button has a border.

Availability

Available in Mac OS X v10.3 and later.

See Also

setButtonBordered (page 437)

itemHeight

Returns the height of each item in the receiver's pop-up list.

```
public float itemHeight()
```

Discussion

The default item height is 16.0.

See Also

intercellSpacing (page 433)

numberOfVisibleItems (page 434)

itemObjectValueAtIndex

Returns the object located at *index* within the receiver's internal item list.

```
public Object itemObjectValueAtIndex(int index)
```

Discussion

If *index* is beyond the end of the list, a `RangeException` is thrown. This method logs a warning if `usesDataSource` (page 439) returns `true`.

See Also

`objectValueOfSelectedItem` (page 435)

noteNumberOfItemsChanged

Informs the receiver that the number of items in its data source has changed, allowing the receiver to update the scrollers in its displayed pop-up list without actually reloading data into the receiver.

```
public void noteNumberOfItemsChanged()
```

Discussion

This method is particularly useful for a data source that continually receives data in the background over a period of time, in which case the NSComboBoxCell can remain responsive to the user while the data is received.

See the NSComboBoxCell.DataSource (page 1953) interface specification for information on the messages an NSComboBoxCell sends to its data source.

See Also

`reloadData` (page 435)

numberOfItems

Returns the total number of items in the pop-up list.

```
public int numberOfItems()
```

See Also

`numberOfVisibleItems` (page 434)

`numberOfItemsInComboBoxCell` (page 1954) (NSComboBoxCellDataSource interface)

numberOfVisibleItems

Returns the maximum number of items visible at any one time in the pop-up list.

```
public int numberOfVisibleItems()
```

See Also

`numberOfItems` (page 434)

objectValueOfSelectedItem

Returns the object from the receiver's internal item list corresponding to the last item selected from the pop-up list, or `null` if no item is selected.

```
public Object objectValueOfSelectedItem()
```

Discussion

Note that nothing is initially selected in a newly initialized combo box cell. This method logs a warning if `usesDataSource` (page 439) returns `true`.

See Also

`indexOfSelectedItem` (page 432)

`comboBoxCellObjectValueForItemAtIndex` (page 1954) (NSComboBoxCellDataSource interface)

objectValues

Returns as an array the receiver's internal item list.

```
public NSArray objectValues()
```

Discussion

This method logs a warning if `usesDataSource` (page 439) returns `true`.

reloadData

Marks the receiver as needing redisplay, so that it will reload the data for visible pop-up items and draw the new values.

```
public void reloadData()
```

See Also

`noteNumberOfItemsChanged` (page 434)

removeAllItems

Removes all items from the receiver's internal item list.

```
public void removeAllItems()
```

Discussion

This method logs a warning if `usesDataSource` (page 439) returns `true`.

See Also

`objectValues` (page 435)

removeItemAtIndex

Removes the object at *index* from the receiver's internal item list and moves all items beyond *index* up one slot to fill the gap.

```
public void removeItemAtIndex(int index)
```

Discussion

This method throws a `RangeException` if *`index`* is beyond the end of the list and logs a warning if `usesDataSource` (page 439) returns `true`.

removeItemWithObjectValue

Removes all occurrences of *`anObject`* from the receiver's internal item list.

```
public void removeItemWithObjectValue(Object anObject)
```

Discussion

Objects are considered equal if `equals` returns `true`. This method logs a warning if `usesDataSource` (page 439) returns `true`.

See Also

`indexOfItemWithObjectValue` (page 432)

scrollItemAtIndexToTop

Scrolls the receiver's pop-up list vertically so that the item specified by *`index`* is as close to the top as possible.

```
public void scrollItemAtIndexToTop(int index)
```

Discussion

The pop-up list need not be displayed at the time this method is invoked.

scrollItemAtIndexToVisible

Scrolls the receiver's pop-up list vertically so that the item specified by *`index`* is visible.

```
public void scrollItemAtIndexToVisible(int index)
```

Discussion

The pop-up list need not be displayed at the time this method is invoked.

selectItemAtIndex

Selects the pop-up list row at *`index`*.

```
public void selectItemAtIndex(int index)
```

Discussion

Posts a `ComboBoxSelectionDidChangeNotification` (page 424) to the default notification center if the selection does in fact change. Note that this method does not alter the contents of the combo box cell's text field—see "Setting the Combo Box's Value" for more information.

See Also

`setObjectValue` (page 459) (NSControl)

selectItemWithObjectValue

Selects the first pop-up list item that corresponds to *anObject*.

```
public void selectItemWithObjectValue(Object anObject)
```

Discussion

Objects are considered equal if `equals` returns `true`. This method logs a warning if `usesDataSource` (page 439) returns `true`. Posts a `ComboBoxSelectionDidChangeNotification` (page 424) to the default notification center if the selection does in fact change. Note that this method doesn't alter the contents of the combo box cell's text field—see "Setting the Combo Box's Value" for more information.

See Also

`setObjectValue` (page 459) (NSControl)

setButtonBordered

Determines whether the button in the combo box is displayed with a border.

```
public void setButtonBordered(boolean flag)
```

Discussion

For example, it is often useful when using a combo box in an NSTableView to display the button without the border. Set *flag* to `true` to display a border.

Availability

Available in Mac OS X v10.3 and later.

See Also

`isButtonBordered` (page 433)

setCompletes

Sets whether the receiver tries to complete what the user types in the text field.

```
public void setCompletes(boolean completes)
```

Discussion

By default, the combo box does not try to.

If *completes* is `true`, every time the user adds characters to the end of the text field, the combo box calls the NSComboBoxCell method `completedString` (page 431). If `completedString` (page 431) returns a string that's longer than the existing string, the combo box replaces the existing string with the returned string and selects the additional characters. If the user is deleting characters or adds characters somewhere besides the end of the string, the combo box does not try to complete it.

See Also

`completes` (page 431)

setDataSource

Sets the receiver's data source to *aSource*.

```
public void setDataSource(Object aSource)
```

Discussion

aSource should implement the appropriate methods of the NSComboBoxCell.DataSource (page 1953) interface. This method doesn't automatically set `usesDataSource` (page 439) to `false` and in fact logs a warning if `usesDataSource` returns `false`.

This method logs a warning if *aSource* doesn't respond to either `numberOfItemsInComboBoxCell` (page 1954) or `comboBoxCellObjectValueForItemAtIndex` (page 1954).

See Also

`setUsesDataSource` (page 439)

setHasVerticalScroller

Determines according to *flag* whether the receiver displays a vertical scroller.

```
public void setHasVerticalScroller(boolean flag)
```

Discussion

By default, *flag* is `true`. If *flag* is `false` and the combo box cell has more list items (either in its internal item list or from its data source) than are allowed by `numberOfVisibleItems` (page 434), only a subset will be displayed. NSComboBoxCell's `scroll...` methods can be used to position this subset within the pop-up list.

Note that if *flag* is `true`, a scroller will be displayed even if the combo box cell has fewer list items than are allowed by `numberOfVisibleItems`.

See Also

`numberOfItems` (page 434)

`scrollItemAtIndexToTop` (page 436)

`scrollItemAtIndexToVisible` (page 436)

setIntercellSpacing

Sets the width and height between pop-up list items to the values in *aSize*.

```
public void setIntercellSpacing(NSSize aSize)
```

Discussion

The default intercell spacing is (3.0, 2.0).

See Also

`setItemHeight` (page 438)

`setNumberOfVisibleItems` (page 439)

setItemHeight

Sets the height for items to *itemHeight*.

```
public void setItemHeight(float itemHeight)
```

See Also

`setIntercellSpacing` (page 438)

`setNumberOfVisibleItems` (page 439)

setNumberOfVisibleItems

Sets the maximum number of items that will be visible at one time in the receiver's pop-up list to *visibleItems*.

```
public void setNumberOfVisibleItems(int visibleItems)
```

See Also

`numberOfItems` (page 434)

`numberOfVisibleItems` (page 434)

`setIntercellSpacing` (page 438)

`setItemHeight` (page 438)

setUsesDataSource

Sets according to *flag* whether the receiver uses an external data source (specified by `setDataSource` (page 437)) to populate the receiver's pop-up list.

```
public void setUsesDataSource(boolean flag)
```

usesDataSource

Returns `true` if the receiver uses an external data source to populate the receiver's pop-up list, `false` if it uses an internal item list.

```
public boolean usesDataSource()
```

See Also

`dataSource` (page 431)

NSControl

Inherits from	NSView : NSResponder : NSObject
Implements	NSCoding (NSResponder)
Package:	com.apple.cocoa.application
Companion guide	Control and Cell Programming Topics for Cocoa

Overview

NSControl is an abstract superclass that provides three fundamental features for implementing user interface devices: drawing devices on the screen, responding to user events, and sending action messages. It works closely with NSCell.

NSControl provides several delegate methods for its subclasses that allow text editing, such as NSTextField and NSMatrix. Note that although NSControl defines delegate methods, it does not itself have a delegate. Any subclass that uses these methods must have a delegate and the methods to get and set it.

Note that although NSControl defines delegate methods, it does not itself have a delegate. Any subclass that uses these methods must have a delegate and the methods to get and set it.

Tasks

Constructors

`NSControl` (page 447)
Creates an NSControl object with a zero-sized frame rectangle and creates a cell for it if the cell's class has been specified for controls of this type with `setCellClass` (page 447).

Setting the Control's Cell

`cellClass` (page 447)
Returns the class of cells used by the receiving class.

`setCellClass` (page 447)
Sets the class of cells used by instances of the receiver.

`cell` (page 449)
Returns the receiver's cell.

`setCell` (page 455)
: Sets the receiver's cell to *`aCell`*.

Enabling and Disabling the Control

`isEnabled` (page 452)
: Returns whether the receiver reacts to mouse events.

`setEnabled` (page 456)
: Sets whether the receiver's cell—or if there is no associated cell, the NSControl itself—is active (that is, whether it tracks the mouse and sends its action to its target).

Identifying the Selected Cell

`selectedCell` (page 454)
: Returns the receiver's selected cell.

`selectedTag` (page 454)
: Returns the tag integer of the receiver's selected cell (see `selectedCell` (page 454)) or –1 if there is no selected cell.

Setting the Control's Value

`doubleValue` (page 450)
: Returns the value of the receiver's cell as a double-precision floating-point number.

`setDoubleValue` (page 456)
: Sets the value of the receiver's cell (or selected cell) to *`aDouble`* (a double-precision floating-point number).

`floatValue` (page 451)
: Returns the value of the receiver's cell (or selected cell, if a multiple-cell NSControl) as a single-precision floating-point number.

`setFloatValue` (page 457)
: Sets the value of the receiver's cell (or selected cell) to *`aFloat`* (a single-precision floating-point number).

`intValue` (page 452)
: Returns the value of the receiver's cell (or selected cell, if a multiple-cell NSControl) as an integer.

`setIntValue` (page 458)
: Sets the value of the receiver's cell (or selected cell) to the integer *`anInt`*.

`objectValue` (page 453)
: Returns the value of the receiver's cell (or selected cell, if a multiple-cell NSControl) as an object.

`setObjectValue` (page 459)
: Sets the value of the receiver's cell (or selected cell) to *`object`*.

`stringValue` (page 461)
: Returns the value of the receiver's cell (or selected cell, if a multiple-cell NSControl) as a String.

`setStringValue` (page 460)
: Sets the value of the receiver's cell (or selected cell) to the string *`aString`*.

`setNeedsDisplay` (page 459)
: Marks the receiver as needing redisplay (assuming automatic display is enabled) after recalculation of its dimensions.

`attributedStringValue` (page 448)
: Returns the object value of the receiver's cell (or selected cell) as an attributed string after validating any editing currently being done.

`setAttributedStringValue` (page 455)
: Sets the value of the receiver's cell (or selected cell) as an attributed string *`object`*.

Interacting with Other Controls

`takeDoubleValue` (page 461)
: Sets the double-precision floating-point value of the receiver's cell (or selected cell) to the value obtained by sending a `doubleValue` (page 450) message to *`sender`*.

`takeFloatValue` (page 462)
: Sets the receiver's selected cell to the value obtained by sending a `floatValue` (page 451) message to *`sender`*.

`takeIntValue` (page 462)
: Sets the receiver's selected cell to the value obtained by sending an `intValue` (page 452) message to *`sender`*.

`takeObjectValue` (page 462)
: Sets the receiver's selected cell to the value obtained by sending an `objectValue` (page 453) message to *`sender`*.

`takeStringValue` (page 462)
: Sets the receiver's selected cell to the value obtained by sending a `stringValue` (page 461) message to *`sender`*.

Formatting Text

`alignment` (page 448)
: Returns the alignment mode of the text in the receiver's cell.

`setAlignment` (page 455)
: Sets the alignment of text in the receiver's cell and, if the cell is being edited, aborts editing and updates the cell.

`font` (page 451)
: Returns the NSFont used to draw text in the receiver's cell.

`setFont` (page 458)
: Sets the font used to draw text in the receiver's cell to *`fontObject`*.

`setFloatingPointFormat` (page 457)
: Sets the autoranging and floating point number format of the receiver's cell, so that at most *`leftDigits`* are displayed to the left of the decimal point, and *`rightDigits`* to the right.

`formatter` (page 451)
: Returns the receiver's formatter.

`setFormatter` (page 458)
: Sets the receiver's formatter to *`newFormatter`*.

Managing the Field Editor

abortEditing (page 448)

Terminates and discards any editing of text displayed by the receiver and removes the field editor's delegate.

currentEditor (page 449)

validateEditing (page 463)

Validates the user's changes to text in a cell of the receiving control.

Resizing the Control

calcSize (page 449)

Recomputes any internal sizing information for the receiver, if necessary, by invoking its NSCell's calcDrawInfo (page 306) method.

sizeToFit (page 461)

Changes the width and the height of the receiver's frame so they are the minimum needed to contain its cell.

Displaying a Cell

selectCell (page 453)

Selects *aCell* (by setting its state to true) and redraws the NSControl if *aCell* is a cell of the receiver and is unselected.

drawCell (page 450)

If *aCell* is the cell used to implement the receiver, then the receiver is displayed.

drawCellInside (page 450)

Draws the inside of the receiver's cell (the area within a bezel or border) specified by *aCell*.

updateCell (page 463)

Redisplays *aCell* or marks it for redisplay.

updateCellInside (page 463)

Redisplays the inside of *aCell* or marks it for redisplay.

Implementing the Target/action Mechanism

action (page 448)

Returns the action-message selector of the receiver's cell (the default NSControl behavior), or the default action-message selector for a control with multiple cells (such as an NSMatrix or an NSForm).

setAction (page 455)

Sets the receiver's action method to *aSelector*.

target (page 463)

Returns the target object of the receiver's cell.

`setTarget` (page 460)
Sets the target object for the action message of the receiver's cell; NSCell's `setTarget` (page 331) is used instead of any subclass override of this method.

`isContinuous` (page 452)
Returns whether the receiver's NSCell continuously sends its action message to its target during mouse tracking.

`setContinuous` (page 456)
Sets whether the receiver's cell continuously sends its action message to its target as it tracks the mouse, depending on the Boolean value *`flag`*.

`sendActionToTarget` (page 454)
Sends `sendActionToTargetFromSender` (page 121) to `NSApplication.sharedApplication()`, which in turn sends a message to *`theTarget`* to perform *`theAction`*, adding the receiver as the last parameter.

`setEventMaskForSendingAction` (page 457)
Sets the conditions on which the receiver sends action messages to its target and returns a bit mask with which to detect the previous settings.

Getting and Setting Tags

`tag` (page 461)
Returns the tag identifying the receiver (not the tag of the receiver's cell).

`setTag` (page 460)
Sets the tag of the receiver to *`anInt`*.

Activating from the Keyboard

`performClick` (page 453)
Can be used to simulate a single mouse click on the receiver.

`refusesFirstResponder` (page 453)
Returns whether the receiver refuses first responder status.

`setRefusesFirstResponder` (page 459)
Sets whether the receiver refuses first responder status, depending on the Boolean value *`flag`*.

Tracking the Mouse

`mouseDown` (page 452)

`ignoresMultiClick` (page 451)
Returns whether the receiver ignores multiple clicks made in rapid succession.

`setIgnoresMultiClick` (page 458)
Sets whether the receiver ignores multiple clicks made in rapid succession, depending on the Boolean value *`flag`*.

Validating the contents of a control

`controlIsValidObject` (page 464) *delegate method*
: Invoked when the insertion point leaves a cell belonging to *`control`*, but before the string value of the cell's object is displayed.

Editing text in a control

`controlTextShouldBeginEditing` (page 465) *delegate method*
: Sent directly by *`control`* to the delegate when the user tries to enter a character in a cell of a control that allows editing of text (such as a text field or form field).

`controlTextShouldEndEditing` (page 465) *delegate method*
: Sent directly by *`control`* to the delegate when the insertion point tries to leave a cell of the control that has been edited.

`controlTextDidBeginEditing` (page 464) *delegate method*
: Sent by the default notification center to the delegate and all observers of the notification when a control with editable cells (such as a text field, a form field, or an NSMatrix) begins editing text.

`controlTextDidChange` (page 465) *delegate method*
: Sent by the default notification center to the delegate when the text in the receiving control (usually a text field, a form field, or NSMatrix with editable cells) changes.

`controlTextDidEndEditing` (page 465) *delegate method*
: Sent by the default notification center to the delegate and all observers of the notification when a control with editable cells (such as a text field, a form field, or an NSMatrix) ends editing text.

Getting error information from a formatter

`controlDidFailToFormatStringErrorDescription` (page 463) *delegate method*
: Invoked when the formatter for the cell belonging to *`control`* (or selected cell) cannot convert a String to an underlying object.

`controlDidFailToValidatePartialString` (page 464) *delegate method*
: Invoked when the formatter for the cell belonging to *`control`* (or selected cell) rejects a partial string a user is typing into the cell.

Working with key bindings

`controlTextViewDoCommandBySelector` (page 466) *delegate method*
: Invoked when users press keys with predefined bindings in a cell of the *`control`* or selected cell, as communicated to the control by the cell's field editor (*`textView`*).

Working with text completion

`controlTextViewCompletionsForPartialWordRange` (page 465) *delegate method*
: Sent to the delegate to allow you to control the list of proposed text completions generated by text fields and other controls.

Constructors

NSControl

Creates an NSControl object with a zero-sized frame rectangle and creates a cell for it if the cell's class has been specified for controls of this type with `setCellClass` (page 447).

```
public NSControl()
```

Discussion
Because NSControl is an abstract class, you should call this constructor only in the constructors of subclasses; that is, there should always be a more specific constructor for the subclass, as this is the constructor for NSControl.

Creates an NSControl object in *frameRect* and creates a cell for it if the cell's class has been specified for controls of this type with `setCellClass` (page 447).

```
public NSControl(NSRect frameRect)
```

Discussion
Because NSControl is an abstract class, you should call this constructor only in the constructors of subclasses; that is, there should always be a more specific constructor for the subclass, as this is the constructor for NSControl.

Static Methods

cellClass

Returns the class of cells used by the receiving class.

```
public static Class cellClass()
```

Discussion
Returns `null` if no cell class has been specified for the receiving class or any of its superclasses (up to NSControl).

See Also
`cell` (page 449)
`setCell` (page 455)

setCellClass

Sets the class of cells used by instances of the receiver.

```
public static void setCellClass(Class class)
```

See Also
`cell` (page 449)
`setCell` (page 455)

Instance Methods

abortEditing

Terminates and discards any editing of text displayed by the receiver and removes the field editor's delegate.

```
public boolean abortEditing()
```

Discussion

Returns `true` if there was a field editor associated with the control, `false` otherwise.

See Also

`currentEditor` (page 449)

`validateEditing` (page 463)

action

Returns the action-message selector of the receiver's cell (the default NSControl behavior), or the default action-message selector for a control with multiple cells (such as an NSMatrix or an NSForm).

```
public NSSelector action()
```

Discussion

For controls with multiple cells, it's better to get the action-message selector for a particular cell.

See Also

`setAction` (page 455)

`setTarget` (page 460)

`target` (page 463)

alignment

Returns the alignment mode of the text in the receiver's cell.

```
public int alignment()
```

Discussion

The return value can be one of these constants: `NSText.LeftTextAlignment`, `NSText.RightTextAlignment`, `NSText.CenterTextAlignment`, `NSText.JustifiedTextAlignment`, or `NSText.NaturalTextAlignment` (the default alignment).

See Also

`setAlignment` (page 455)

attributedStringValue

Returns the object value of the receiver's cell (or selected cell) as an attributed string after validating any editing currently being done.

```
public NSAttributedString attributedStringValue()
```

Discussion

If no cell is associated with the receiver, returns an empty attributed string.

See Also

`setAttributedStringValue` (page 455)

calcSize

Recomputes any internal sizing information for the receiver, if necessary, by invoking its NSCell's `calcDrawInfo` (page 306) method.

```
public void calcSize()
```

Discussion

Most NSControls maintain a flag that informs them if any of their cells have been modified in such a way that the location or size of the cell should be recomputed. If such a modification happens, `calcSize` is automatically invoked whenever the NSControl is displayed; you never need to invoke it yourself.

See Also

`sizeToFit` (page 461)

cell

Returns the receiver's cell.

```
public NSCell cell()
```

Discussion

In NSControls with multiple cells (such as NSMatrix or NSForm), use `selectedCell` (page 454) or a similar method for finding a particular cell.

See Also

`cellClass` (page 447)

`setCellClass` (page 447)

`setCell` (page 455)

currentEditor

```
public NSText currentEditor()
```

Discussion

If the receiver is being edited—that is, it has a field editor and is the first responder of its NSWindow—this method returns the field editor; otherwise, it returns `null`.

The field editor is a single NSTextView object that is shared among all the controls in a window for light text-editing needs. It is automatically instantiated when needed.

See Also

`abortEditing` (page 448)

validateEditing (page 463)

doubleValue

Returns the value of the receiver's cell as a double-precision floating-point number.

```
public double doubleValue()
```

Discussion

If the NSControl contains many cells (for example, NSMatrix), then the value of the currently selected cell is returned. If the NSControl is in the process of editing the affected cell, then it invokes validateEditing (page 463) before extracting and returning the value.

See Also

doubleValue (page 450)

floatValue (page 451)

intValue (page 452)

objectValue (page 453)

stringValue (page 461)

setDoubleValue (page 456)

drawCell

If *aCell* is the cell used to implement the receiver, then the receiver is displayed.

```
public void drawCell(NSCell aCell)
```

Discussion

This method is provided primarily to support a consistent set of methods between NSControls with single and multiple cells, because an NSControl with multiple cells needs to be able to draw a single cell at a time.

See Also

selectCell (page 453)

updateCell (page 463)

updateCellInside (page 463)

drawCellInside

Draws the inside of the receiver's cell (the area within a bezel or border) specified by *aCell*.

```
public void drawCellInside(NSCell aCell)
```

Discussion

If the receiver is transparent, the method causes the superview to draw itself. This method invokes NSCell's drawInteriorWithFrameInView (page 309) method. This method has no effect on NSControls (such as NSMatrix and NSForm) that have multiple cells.

See Also

selectCell (page 453)

updateCell (page 463)

updateCellInside (page 463)

floatValue

Returns the value of the receiver's cell (or selected cell, if a multiple-cell NSControl) as a single-precision floating-point number.

```
public float floatValue()
```

Discussion

See doubleValue (page 450) for more details.

See Also

doubleValue (page 450)

intValue (page 452)

objectValue (page 453)

stringValue (page 461)

setFloatValue (page 457)

font

Returns the NSFont used to draw text in the receiver's cell.

```
public NSFont font()
```

See Also

setFont (page 458)

formatter

Returns the receiver's formatter.

```
public Object formatter()
```

See Also

setFormatter (page 458)

ignoresMultiClick

Returns whether the receiver ignores multiple clicks made in rapid succession.

```
public boolean ignoresMultiClick()
```

Discussion

See setIgnoresMultiClick (page 458) for details.

intValue

Returns the value of the receiver's cell (or selected cell, if a multiple-cell NSControl) as an integer.

```
public int intValue()
```

Discussion

See `doubleValue` (page 450) for more details.

See Also

`doubleValue` (page 450)

`floatValue` (page 451)

`objectValue` (page 453)

`stringValue` (page 461)

`setIntValue` (page 458)

isContinuous

Returns whether the receiver's NSCell continuously sends its action message to its target during mouse tracking.

```
public boolean isContinuous()
```

See Also

`setContinuous` (page 456)

isEnabled

Returns whether the receiver reacts to mouse events.

```
public boolean isEnabled()
```

See Also

`setEnabled` (page 456)

mouseDown

```
public void mouseDown(NSEvent theEvent)
```

Discussion

Invoked when the mouse button is pressed while the cursor is within the bounds of the receiver, generating *theEvent*. This method highlights the receiver's NSCell and sends it a `trackMouse` (page 336) message. Whenever the NSCell 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` (page 336) message is sent. This behavior repeats until the mouse button goes up. If it goes up with the cursor in the control, the state of the control is changed, and the action message is sent to the target. If the mouse button goes up when the cursor is outside the control, no action message is sent.

See Also

`ignoresMultiClick` (page 451)

trackMouse (page 336) (NSCell)

objectValue

Returns the value of the receiver's cell (or selected cell, if a multiple-cell NSControl) as an object.

```
public Object objectValue()
```

Discussion
See doubleValue (page 450) for more details.

See Also
doubleValue (page 450)
floatValue (page 451)
intValue (page 452)
stringValue (page 461)
setObjectValue (page 459)

performClick

Can be used to simulate a single mouse click on the receiver.

```
public void performClick(Object sender)
```

Discussion
sender is ignored. This method calls performClick (page 318) on the receiver's cell with the sender being the control itself. Throws an exception if the action message cannot be successfully sent.

refusesFirstResponder

Returns whether the receiver refuses first responder status.

```
public boolean refusesFirstResponder()
```

See Also
setRefusesFirstResponder (page 459)

selectCell

Selects *aCell* (by setting its state to true) and redraws the NSControl if *aCell* is a cell of the receiver and is unselected.

```
public void selectCell(NSCell aCell)
```

See Also
selectedCell (page 454)

selectedCell

Returns the receiver's selected cell.

```
public NSCell selectedCell()
```

Discussion

The default implementation for NSControl simply returns the associated cell (or `null` if no cell has been set). Subclasses of NSControl that manage multiple cells (such as NSMatrix and NSForm) override this method to return the cell selected by users.

See Also

`cell` (page 449)

`setCell` (page 455)

selectedTag

Returns the tag integer of the receiver's selected cell (see `selectedCell` (page 454)) or –1 if there is no selected cell.

```
public int selectedTag()
```

Discussion

When you set the tag of a control with a single cell in Interface Builder, it sets the tags of both the control and the cell with the same value as a convenience.

See Also

`setTag` (page 460)

`tag` (page 461)

sendActionToTarget

Sends `sendActionToTargetFromSender` (page 121) to `NSApplication.sharedApplication()`, which in turn sends a message to *`theTarget`* to perform *`theAction`*, adding the receiver as the last parameter.

```
public boolean sendActionToTarget(NSSelector theAction, Object theTarget)
```

Discussion

`sendActionToTarget` is invoked primarily by NSCell's `trackMouse` (page 336).

If *`theAction`* is `NULL`, no message is sent. If *`theTarget`* is `null`, `NSApplication.sharedApplication()` looks for an object that can respond to the message by following the responder chain (see the class description for NSActionCell (page 45)). This method returns `false` if no object that responds to *`theAction`* could be found.

See Also

`action` (page 448)

`target` (page 463)

setAction

Sets the receiver's action method to *aSelector*.

```
public void setAction(NSSelector aSelector)
```

Discussion

If *aSelector* is `NULL`, then no action messages will be sent from the receiver.

See *Action Messages* for additional information on action messages.

See Also

`action` (page 448)

`setTarget` (page 460)

`target` (page 463)

setAlignment

Sets the alignment of text in the receiver's cell and, if the cell is being edited, aborts editing and updates the cell.

```
public void setAlignment(int mode)
```

Discussion

mode is one of five constants: `NSText.LeftTextAlignment`, `NSText.RightTextAlignment`, `NSText.CenterTextAlignment`, `NSText.JustifiedTextAlignment`, `NSText.NaturalTextAlignment` (the default alignment for the text).

See Also

`alignment` (page 448)

setAttributedStringValue

Sets the value of the receiver's cell (or selected cell) as an attributed string *object*.

```
public void setAttributedStringValue(NSAttributedString object)
```

Discussion

If the cell is being edited, it aborts all editing before setting the value; if the cell doesn't inherit from NSActionCell, it marks it for automatic redisplay (NSActionCell performs its own updating of cells).

See Also

`attributedStringValue` (page 448)

setCell

Sets the receiver's cell to *aCell*.

```
public void setCell(NSCell aCell)
```

Discussion

Use this method with great care as it can irrevocably damage the affected control; specifically, you should only use this method in initializers for subclasses of NSControl.

See Also

`cell` (page 449)

`selectedCell` (page 454)

setContinuous

Sets whether the receiver's cell continuously sends its action message to its target as it tracks the mouse, depending on the Boolean value *`flag`*.

```
public void setContinuous(boolean flag)
```

See Also

`isContinuous` (page 452)

setDoubleValue

Sets the value of the receiver's cell (or selected cell) to *`aDouble`* (a double-precision floating-point number).

```
public void setDoubleValue(double aDouble)
```

Discussion

If the cell is being edited, it aborts all editing before setting the value; if the cell doesn't inherit from NSActionCell, it marks the cell's interior as needing to be redisplayed (NSActionCell performs its own updating of cells).

See Also

`doubleValue` (page 450)

`setFloatValue` (page 457)

`setIntValue` (page 458)

`setObjectValue` (page 459)

`setStringValue` (page 460)

setEnabled

Sets whether the receiver's cell—or if there is no associated cell, the NSControl itself—is active (that is, whether it tracks the mouse and sends its action to its target).

```
public void setEnabled(boolean flag)
```

Discussion

If *`flag`* is `false`, any editing is aborted. Redraws the entire control if it is marked as needing redisplay. Subclasses may want to override this method to redraw only a portion of the control when the enabled state changes, as do NSButton and NSSlider.

See Also

`isEnabled` (page 452)

setEventMaskForSendingAction

Sets the conditions on which the receiver sends action messages to its target and returns a bit mask with which to detect the previous settings.

```
public int setEventMaskForSendingAction(int mask)
```

Discussion

You use this method during mouse tracking when the mouse button changes state, the mouse moves, or—if the cell is marked to send its action continuously while tracking—periodically. Because of this, the only bits checked in *mask* are `LeftMouseDownMask`, `LeftMouseUpMask`, `LeftMouseDraggedMask`, and `PeriodicMask`, which are declared in `NSEvent.h`. NSControl's default implementation simply invokes the `setEventMaskForSendingAction` (page 324) method of its associated cell.

See Also

`sendActionToTarget` (page 454)

`setEventMaskForSendingAction` (page 324) (NSCell)

setFloatingPointFormat

Sets the autoranging and floating point number format of the receiver's cell, so that at most *leftDigits* are displayed to the left of the decimal point, and *rightDigits* to the right.

```
public void setFloatingPointFormat(boolean autoRange, int leftDigits, int
    rightDigits)
```

Discussion

The *autoRange* argument specifies whether floating-point numbers are autoranged in the receiver. See the description of this method in the NSCell (page 295) class specification for details. If the cell is being edited, what's typed is discarded, and the cell's interior is redrawn.

See Also

`setFloatingPointFormat` (page 325) (NSCell)

setFloatValue

Sets the value of the receiver's cell (or selected cell) to *aFloat* (a single-precision floating-point number).

```
public void setFloatValue(float aFloat)
```

Discussion

If the cell is being edited, it aborts all editing before setting the value; if the cell doesn't inherit from NSActionCell, it marks the cell's interior as needing to be redisplayed (NSActionCell performs its own updating of cells).

See Also

`floatValue` (page 451)

`setDoubleValue` (page 456)

`setIntValue` (page 458)

`setObjectValue` (page 459)

`setStringValue` (page 460)

setFont

Sets the font used to draw text in the receiver's cell to *fontObject*.

```
public void setFont(NSFont fontObject)
```

Discussion

If the cell is being edited, the text in the cell is redrawn in the new font, and the cell's editor (the NSText object used globally for editing) is updated with the new NSFont.

See Also

setFont (page 458)

setFormatter

Sets the receiver's formatter to *newFormatter*.

```
public void setFormatter(NSFormatter newFormatter)
```

See Also

formatter (page 451)

setIgnoresMultiClick

Sets whether the receiver ignores multiple clicks made in rapid succession, depending on the Boolean value *flag*.

```
public void setIgnoresMultiClick(boolean flag)
```

Discussion

By default, controls treat double clicks as two distinct clicks, triple clicks as three distinct clicks, and so on. However, when an NSControl returning `true` to this method receives multiple clicks (within a predetermined interval), each `mouseDown` event after the first is passed on to `super`.

See Also

ignoresMultiClick (page 451)

setIntValue

Sets the value of the receiver's cell (or selected cell) to the integer *anInt*.

```
public void setIntValue(int anInt)
```

Discussion

If the cell is being edited, it aborts all editing before setting the value; if the cell doesn't inherit from NSActionCell, it marks the cell's interior as needing to be redisplayed (NSActionCell performs its own updating of cells).

See Also

intValue (page 452)

setDoubleValue (page 456)

setFloatValue (page 457)

setObjectValue (page 459)
setStringValue (page 460)

setNeedsDisplay

Marks the receiver as needing redisplay (assuming automatic display is enabled) after recalculation of its dimensions.

```
public void setNeedsDisplay()
```

See Also
setNeedsDisplay (page 1779) (NSView)

setObjectValue

Sets the value of the receiver's cell (or selected cell) to *object*.

```
public void setObjectValue(Object object)
```

Discussion
If the cell is being edited, it aborts all editing before setting the value; if the cell doesn't inherit from NSActionCell, it marks the cell's interior as needing to be redisplayed (NSActionCell performs its own updating of cells).

See Also
objectValue (page 453)
setDoubleValue (page 456)
setFloatValue (page 457)
setIntValue (page 458)
setStringValue (page 460)

setRefusesFirstResponder

Sets whether the receiver refuses first responder status, depending on the Boolean value *flag*.

```
public void setRefusesFirstResponder(boolean flag)
```

Discussion
By default, the user can advance the focus of keyboard events between controls by pressing the Tab key; when this focus—or first responder status—is indicated for a control (by the insertion point or, for nontext controls, a faint rectangle), the user can activate the control by pressing the Space bar.

See Also
refusesFirstResponder (page 453)
objectValue (page 453)
setDoubleValue (page 456)
setFloatValue (page 457)

setStringValue

Sets the value of the receiver's cell (or selected cell) to the string *aString*.

```
public void setStringValue(String aString)
```

Discussion

If the cell is being edited, it aborts all editing before setting the value; if the cell doesn't inherit from NSActionCell, it marks the cell's interior as needing to be redisplayed (NSActionCell performs its own updating of cells).

See Also

`setDoubleValue` (page 456)

`setFloatValue` (page 457)

`setIntValue` (page 458)

`setObjectValue` (page 459)

`stringValue` (page 461)

setTag

Sets the tag of the receiver to *anInt*.

```
public void setTag(int anInt)
```

Discussion

It doesn't affect the tag of the receiver's cell. Tags allow you to identify particular cells. Tag values are not used internally; they are only changed by external invocations of `setTag` (page 460). You typically set tag values in Interface Builder. When you set the tag of a control with a single cell in Interface Builder, it sets the tags of both the control and the cell to the same value as a convenience.

See Also

`tag` (page 461)

`selectedTag` (page 454)

setTarget

Sets the target object for the action message of the receiver's cell; NSCell's `setTarget` (page 331) is used instead of any subclass override of this method.

```
public void setTarget(Object anObject)
```

Discussion

If *anObject* is `null` and the control sends an action message, the application looks for an object that can respond to the message by following the responder chain (see description of the NSActionCell (page 45) class for details).

See Also

`action` (page 448)

`setAction` (page 455)

`target` (page 463)

`setTarget` (page 331) (NSCell)

sizeToFit

Changes the width and the height of the receiver's frame so they are the minimum needed to contain its cell.

```
public void sizeToFit()
```

Discussion

If you want a multiple-cell custom subclass of NSControl to size itself to fit its cells, you must override this method. This method neither redisplays the receiver nor marks it as needing display. You must do this yourself with either `display` (page 1747) or `setNeedsDisplay` (page 459).

See Also

`calcSize` (page 449)

stringValue

Returns the value of the receiver's cell (or selected cell, if a multiple-cell NSControl) as a String.

```
public String stringValue()
```

Discussion

See `doubleValue` (page 450) for details.

See Also

`doubleValue` (page 450)

`floatValue` (page 451)

`intValue` (page 452)

`objectValue` (page 453)

`setStringValue` (page 460)

tag

Returns the tag identifying the receiver (not the tag of the receiver's cell).

```
public int tag()
```

Discussion

Tags allow you to identify particular controls. Tag values are not used internally; they are only changed by external invocations of `setTag`. You typically set tag values in Interface Builder. When you set the tag of a control with a single cell in Interface Builder, it sets the tags of both the control and the cell to the same value as a convenience.

See Also

`setTag` (page 460)

`selectedTag` (page 454)

takeDoubleValue

Sets the double-precision floating-point value of the receiver's cell (or selected cell) to the value obtained by sending a `doubleValue` (page 450) message to *`sender`*.

```
public void takeDoubleValue(Object sender)
```

Discussion
You can use this method to link action messages between controls. It permits one control or cell (*sender*) to affect the value of another control (the receiver) by invoking this method in an action message to the receiver. For example, a text field can be made the target of a slider. Whenever the slider is moved, it will send a `takeDoubleValue` message to the text field. The text field 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.

takeFloatValue

Sets the receiver's selected cell to the value obtained by sending a `floatValue` (page 451) message to *sender*.

```
public void takeFloatValue(Object sender)
```

Discussion
See `takeDoubleValue` (page 461) for more information.

takeIntValue

Sets the receiver's selected cell to the value obtained by sending an `intValue` (page 452) message to *sender*.

```
public void takeIntValue(Object sender)
```

Discussion
See `takeDoubleValue` (page 461) for more information.

takeObjectValue

Sets the receiver's selected cell to the value obtained by sending an `objectValue` (page 453) message to *sender*.

```
public void takeObjectValue(Object sender)
```

Discussion
See `takeDoubleValue` (page 461) for more information.

takeStringValue

Sets the receiver's selected cell to the value obtained by sending a `stringValue` (page 461) message to *sender*.

```
public void takeStringValue(Object sender)
```

Discussion
See `takeDoubleValue` (page 461) for more information.

target

Returns the target object of the receiver's cell.

```
public Object target()
```

See Also

`action` (page 448)

`setAction` (page 455)

`setTarget` (page 460)

updateCell

Redisplays *aCell* or marks it for redisplay.

```
public void updateCell(NSCell aCell)
```

updateCellInside

Redisplays the inside of *aCell* or marks it for redisplay.

```
public void updateCellInside(NSCell aCell)
```

validateEditing

Validates the user's changes to text in a cell of the receiving control.

```
public void validateEditing()
```

Discussion

Validation sets the object value of the cell to the current contents of the cell's editor (the NSText object used for editing), storing it as a simple String or an attributed string object based on the attributes of the editor.

See Also

`abortEditing` (page 448)

`currentEditor` (page 449)

Delegate Methods

controlDidFailToFormatStringErrorDescription

Invoked when the formatter for the cell belonging to *control* (or selected cell) cannot convert a String to an underlying object.

```
public abstract boolean controlDidFailToFormatStringErrorDescription(NSControl
    control, String string, String error)
```

Discussion

error is a localized user-presentable String that explains why the conversion failed. Evaluate the error or query the user and return `true` if *string* should be accepted as is, or `false` if *string* should be rejected.

See Also

`objectValueForString` (NSFormatter)

controlDidFailToValidatePartialString

Invoked when the formatter for the cell belonging to *control* (or selected cell) rejects a partial string a user is typing into the cell.

```
public abstract void controlDidFailToValidatePartialString(NSControl control, String
    string, String error)
```

Discussion

This String (*string*) includes the character that caused the rejection. *error* is a localized user-presentable String that explains why the validation failed. You can implement this method to display a warning message or perform a similar action when the user enters improperly formatted text.

See Also

`isPartialStringValid` (NSFormatter)

controlIsValidObject

Invoked when the insertion point leaves a cell belonging to *control*, but before the string value of the cell's object is displayed.

```
public abstract boolean controlIsValidObject(NSControl control, Object object)
```

Discussion

Return `true` to allow display of the string and `false` to reject display and return the cursor to the cell. This method gives the delegate the opportunity to validate the contents of the control's cell (or selected cell). In validating, the delegate checks *object* to determine if it falls within a permissible range, has required attributes, accords with a given context, and so on. Examples of objects subject to such evaluations are an NSDate object that should not represent a future date or a monetary amount (represented by an NSNumber) that exceeds a predetermined limit.

controlTextDidBeginEditing

Sent by the default notification center to the delegate and all observers of the notification when a control with editable cells (such as a text field, a form field, or an NSMatrix) begins editing text.

```
public abstract void controlTextDidBeginEditing(NSNotification aNotification)
```

Discussion

The name of notification *aNotification* is always `ControlTextDidBeginEditingNotification` (page 466). Use the key `"NSFieldEditor"` to obtain the field editor from *aNotification*'s `userInfo` dictionary. If the delegate implements this method, it's automatically registered to receive this notification.

controlTextDidChange

Sent by the default notification center to the delegate when the text in the receiving control (usually a text field, a form field, or NSMatrix with editable cells) changes.

```
public abstract void controlTextDidChange(NSNotification aNotification)
```

Discussion

The name of notification `aNotification` is always `ControlTextDidChangeNotification` (page 467). Use the key `"NSFieldEditor"` to obtain the field editor from `aNotification`'s `userInfo` dictionary. If the delegate implements this method, it's automatically registered to receive this notification.

controlTextDidEndEditing

Sent by the default notification center to the delegate and all observers of the notification when a control with editable cells (such as a text field, a form field, or an NSMatrix) ends editing text.

```
public abstract void controlTextDidEndEditing(NSNotification aNotification)
```

Discussion

The name of notification `aNotification` is always `ControlTextDidEndEditingNotification` (page 467). Use the key `"NSFieldEditor"` to obtain the field editor from `aNotification`'s `userInfo` dictionary. If the delegate implements this method, it's automatically registered to receive this notification.

controlTextShouldBeginEditing

Sent directly by `control` to the delegate when the user tries to enter a character in a cell of a control that allows editing of text (such as a text field or form field).

```
public abstract boolean controlTextShouldBeginEditing(NSControl control, NSText
    fieldEditor)
```

Discussion

Return `true` if the NSControl's `fieldEditor` should be allowed to start editing the text, `false` otherwise.

controlTextShouldEndEditing

Sent directly by `control` to the delegate when the insertion point tries to leave a cell of the control that has been edited.

```
public abstract boolean controlTextShouldEndEditing(NSControl control, NSText
    fieldEditor)
```

Discussion

It's sent only by controls that allow editing of text (such as a text field or a form field). Return `true` if the control's `fieldEditor` should be allowed to end its edit session, `false` otherwise.

controlTextViewCompletionsForPartialWordRange

Sent to the delegate to allow you to control the list of proposed text completions generated by text fields and other controls.

```
public abstract NSArray controlTextViewCompletionsForPartialWordRange(NSControl
    control, NSTextView textView, NSArray words, NSRange charRange)
```

Discussion
Arguments include the text view for which the list was generated, the list of potential completions, and the partial word range (that is, the number of characters the user has already typed). The array you return is used for the completions list in place of the potential list passed to this method.

The completions list is an array of strings, in the order in which they should be presented. Each string should be a complete word that the user might be trying to type. The strings must be complete words rather than just the remainder of the word, in case completion requires some slight modification of what the user has already typed—for example, the addition of an accent, or a change in capitalization. You can also use this method to support abbreviations that complete into words that don't start with the characters of the abbreviation.

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

controlTextViewDoCommandBySelector

Invoked when users press keys with predefined bindings in a cell of the `control` or selected cell, as communicated to the control by the cell's field editor (`textView`).

```
public abstract boolean controlTextViewDoCommandBySelector(NSControl control,
    NSTextView textView, NSSelector command)
```

Discussion
The delegate returns `true` if it handles the key binding, and `false` otherwise. These bindings are usually implemented as methods (`command`) defined in NSResponder; examples of such key bindings are arrow keys (for directional movement) and the Escape key (for name completion). By implementing this method, the delegate can override the default implementation of `command` and supply its own behavior.

Notifications

NSControl posts the following notifications to interested observers and its delegate.

ControlTextDidBeginEditingNotification

The field editor of the edited cell originally sends a `TextDidBeginEditingNotification` (page 1532) to the control, which passes it on in this form to its delegate. The notification object is the NSControl posting the notification. The `userInfo` dictionary contains the following information:

Key	Value
"NSFieldEditor"	The edited cell's field editor

See the description of `controlTextDidBeginEditing` (page 464) for details.

ControlTextDidChangeNotification

The field editor of the edited cell originally sends a TextDidChangeNotification (page 1532) to the control, which passes it on in this form to its delegate. The notification object is the NSControl posting the notification. The *userInfo* dictionary contains the following information:

Key	Value
"NSFieldEditor"	The edited cell's field editor

See the description of controlTextDidChange (page 465) for details.

ControlTextDidEndEditingNotification

The field editor of the edited cell originally sends a ControlTextDidEndEditingNotification (page 467) to the control, which passes it on in this form to its delegate. The notification object is the NSControl posting the notification. The *userInfo* dictionary contains the following information:

Key	Value
"NSFieldEditor"	The edited cell's field editor

See the description of controlTextDidEndEditing (page 465) for details.

NSController

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.application
Availability	Available in Mac OS X v10.3 and later.
Companion guide	Cocoa Bindings Programming Topics

Overview

NSController is an abstract class that implements the base functionality required by the Controller Layer.

Tasks

Constructors

`NSController` (page 470)
Creates and returns an empty NSController.

Managing Editing

`objectDidBeginEditing` (page 471)
Invoked to inform the receiver that *`editor`* has uncommitted changes that can affect the receiver.

`objectDidEndEditing` (page 471)
Invoked to inform the receiver that *`editor`* has committed or discarded its changes.

`commitEditing` (page 470)
Causes the receiver to attempt to commit any pending edits, returning `true` if successful or no edits where pending.

`discardEditing` (page 471)
Discards any pending changes by registered editors.

`isEditing` (page 471)
Returns `true` if there are any editors currently registered with the receiver, `false` otherwise.

Binding

`bind` (page 470)
Creates a relationship between the receiver's *binding* and the property of *observableController* specified by *keyPath*.

Constructors

NSController

Creates and returns an empty NSController.

```
public NSController()
```

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

Instance Methods

bind

Creates a relationship between the receiver's *binding* and the property of *observableController* specified by *keyPath*.

```
public void bind(String binding, Object observableController, String keyPath,
    NSDictionary options)
```

Discussion
The *binding* is the key path for a property of the receiver previously exposed. The *options* dictionary is optional. If present, it contains placeholder objects or an NSValueTransformer identifier as described in "Constants" (page 2005).

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

commitEditing

Causes the receiver to attempt to commit any pending edits, returning `true` if successful or no edits where pending.

```
public boolean commitEditing()
```

Discussion
A commit is denied if the receiver fails to apply the changes to the model object, perhaps due to a validation error.

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

See Also
`discardEditing` (page 471)

discardEditing

Discards any pending changes by registered editors.

```
public void discardEditing()
```

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

See Also
`commitEditing` (page 470)

isEditing

```
public boolean isEditing()
```

Returns `true` if there are any editors currently registered with the receiver, `false` otherwise.

```
- (BOOL)isEditing
```

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

objectDidBeginEditing

Invoked to inform the receiver that *editor* has uncommitted changes that can affect the receiver.

```
public void objectDidBeginEditing(Object editor)
```

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

See Also
`objectDidEndEditing` (page 471)

objectDidEndEditing

Invoked to inform the receiver that *editor* has committed or discarded its changes.

```
public void objectDidEndEditing(Object editor)
```

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

See Also

objectDidBeginEditing (page 471)

NSControllerPlaceholders

Inherits from	NSObject
Package:	com.apple.cocoa.application
Availability	Available in Mac OS X v10.3 and later.
Companion guide	Cocoa Bindings Programming Topics

Overview

The NSControllerPlaceholders class provides an means for an object to register default placeholders that will be displayed for a binding when no other placeholder has been specified. Individual placeholder values can be specified for each of the marker objects, as well as when the property is `null`.

Placeholders are used when a property of an instance of the receiving class is accessed through a key value coding compliant method, and returns `null` or a specialized marker.

Tasks

Constructors

`NSControllerPlaceholders` (page 474)
Creates a new NSControllerPlaceholders instance.

Managing Default Placeholders

`setDefaultPlaceholderForMarker` (page 476)
Sets *`placeholder`* as the default placeholder for the *`binding`*, when a key value coding compliant property of an instance of *`classObject`* returns the value specified by *`marker`*, and no other placeholder has been specified.

`defaultPlaceholderForMarker` (page 474)
Returns an object that will be used as the placeholder for the *`binding`*, when a key value coding compliant property of an instance of *`classObject`* returns the value specified by *`marker`*, and no other placeholder has been specified.

Obtaining Controller Markers

`multipleValuesMarker` (page 475)
Returns a marker which indicates that a controller's selection contains multiple items.

`noSelectionMarker` (page 475)
Returns a marker which indicates that a controller's selection is currently empty.

`notApplicableMarker` (page 475)
Returns a marker which indicates that a controller does not support selection.

Testing Markers

`isControllerMarker` (page 475)
Returns whether *marker* is a valid controller marker.

Constructors

NSControllerPlaceholders

Creates a new NSControllerPlaceholders instance.

```
public NSControllerPlaceholders()
```

Discussion
All NSControllerPlaceholders methods are static, so there is no need create individual instances.

Static Methods

defaultPlaceholderForMarker

Returns an object that will be used as the placeholder for the *binding*, when a key value coding compliant property of an instance of *classObject* returns the value specified by *marker*, and no other placeholder has been specified.

```
public static Object defaultPlaceholderForMarker(Class classObject, Object marker,
      String binding)
```

Discussion
marker can be `null` or one of the controller markers returned by `multipleValuesMarker` (page 475), `noSelectionMarker` (page 475), and `notApplicableMarker` (page 475).

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

See Also
`setDefaultPlaceholderForMarker` (page 476)

isControllerMarker

Returns whether *marker* is a valid controller marker.

```
public static boolean isControllerMarker(Object marker)
```

Discussion
Valid controller markers are returned by multipleValuesMarker (page 475), noSelectionMarker (page 475), and notApplicableMarker (page 475).

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

multipleValuesMarker

Returns a marker which indicates that a controller's selection contains multiple items.

```
public static Object multipleValuesMarker()
```

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

See Also
noSelectionMarker (page 475)
notApplicableMarker (page 475)

noSelectionMarker

Returns a marker which indicates that a controller's selection is currently empty.

```
public static Object noSelectionMarker()
```

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

See Also
multipleValuesMarker (page 475)
notApplicableMarker (page 475)

notApplicableMarker

Returns a marker which indicates that a controller does not support selection.

```
public static Object notApplicableMarker()
```

Discussion
NSUserDefaultsController is an example of a controller that does not support the concept of selection.

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

See Also

`multipleValuesMarker` (page 475)

`noSelectionMarker` (page 475)

setDefaultPlaceholderForMarker

Sets *placeholder* as the default placeholder for the *binding*, when a key value coding compliant property of an instance of *classObject* returns the value specified by *marker*, and no other placeholder has been specified.

```
public static void setDefaultPlaceholderForMarker(Class classObject, Object
    placeholder, Object marker, String binding)
```

Discussion

marker can be `null` or one of the controller markers returned by `multipleValuesMarker` (page 475), `noSelectionMarker` (page 475), and `notApplicableMarker` (page 475).

Availability

Available in Mac OS X v10.3 and later.

See Also

`defaultPlaceholderForMarker` (page 474)

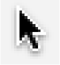
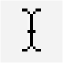
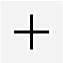
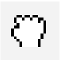
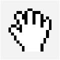
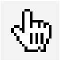
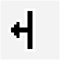
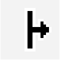
NSCursor

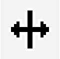
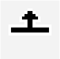
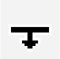
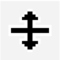
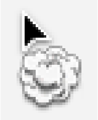
Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.application
Companion guide	Cursor Management

Overview

Instances of the NSCursor class manage the appearance of the cursor.

The following table shows and describes the system cursors, and indicates the static method for obtaining them:

Cursor	Description
	The arrow cursor (`arrowCursor` (page 480))
	The I-beam cursor for indicating insertion points (`IBeamCursor` (page 482))
	The cross-hair cursor (`crosshairCursor` (page 481))
	The closed-hand cursor (`closedHandCursor` (page 481))
	The open-hand cursor (`openHandCursor` (page 482))
	The pointing-hand cursor (`pointingHandCursor` (page 482))
	The resize-left cursor (`resizeLeftCursor` (page 483))
	The resize-right cursor (`resizeRightCursor` (page 483))

Cursor	Description
	The resize-left-and-right cursor (resizeLeftRightCursor (page 483))
	The resize-up cursor (resizeUpCursor (page 483))
	The resize-down cursor (resizeDownCursor (page 483))
	The resize-up-and-down cursor (resizeUpDownCursor (page 484))
	The disappearing item cursor (disappearingItemCursor (page 481))

In Mac OS X version 10.3 and later, cursor size is no longer limited to 16 by 16 pixels.

Tasks

Constructors

NSCursor (page 480)
Creates a new cursor, assigning it a null image and a hot spot of NSPoint.ZeroPoint.

Setting Cursor Attributes

image (page 485)
Returns the image for the receiver, or null if none exists.

hotSpot (page 484)
Returns the position of the hot spot, specified according to the cursor's flipped coordinate system.

hide (page 481)
Makes the current cursor invisible.

unhide (page 484)
Negates an earlier call to hide (page 481) by showing the current cursor.

setHiddenUntilMouseMoves (page 484)

Controlling Which Cursor Is Current

`popCursor` (page 482)
Pops the current cursor off the top of the stack.

`pop` (page 486)
Sends a `popCursor` (page 482) message to the receiver's class.

`push` (page 486)
Puts the receiver on top of the cursor stack and makes it the current cursor.

`set` (page 486)
Makes the receiver the current cursor.

`mouseEntered` (page 485)

`setOnMouseEntered` (page 487)

`isSetOnMouseEntered` (page 485)

`mouseExited` (page 486)

`setOnMouseExited` (page 487)

`isSetOnMouseExited` (page 485)

Retrieving Cursor Instances

`currentCursor` (page 481)
Returns the cursor currently displayed on the screen.

`arrowCursor` (page 480)
Returns the default cursor, a slanted arrow with its hot spot at the tip.

`closedHandCursor` (page 481)
Returns the closed-hand system cursor.

`crosshairCursor` (page 481)
Returns the cross-hair system cursor.

`disappearingItemCursor` (page 481)
Returns a cursor indicating that the current operation will result in a disappearing item (for example, when dragging an item from the dock or a toolbar).

`IBeamCursor` (page 482)
Returns a cursor that looks like a capital I with a tiny crossbeam at its middle.

`openHandCursor` (page 482)
Returns the open-hand system cursor.

`pointingHandCursor` (page 482)
Returns the pointing-hand system cursor.

`resizeDownCursor` (page 483)

`resizeLeftCursor` (page 483)

`resizeLeftRightCursor` (page 483)
Returns the resize-left-and-right system cursor.

`resizeRightCursor` (page 483)
Returns the resize-right system cursor.

`resizeUpCursor` (page 483)
Returns the resize-up system cursor.

`resizeUpDownCursor` (page 484)
Returns the resize-up-and-down system cursor.

Constructors

NSCursor

Creates a new cursor, assigning it a `null` image and a hot spot of `NSPoint.ZeroPoint`.

```
public NSCursor()
```

Creates a cursor, assigns it *newImage*, and sets its hot spot to *aPoint*.

```
public NSCursor(NSImage newImage, NSPoint aPoint)
```

Creates a new cursor, assigns it *newImage*, and sets its hot spot to *aPoint*.

```
public NSCursor(NSImage newImage, NSColor fg, NSColor bg, NSPoint aPoint)
```

Discussion
The foreground and background colors (*fg* and *bg*, respectively) are currently ignored.

Static Methods

arrowCursor

Returns the default cursor, a slanted arrow with its hot spot at the tip.

```
public static NSCursor arrowCursor()
```

Discussion
The arrow cursor is the one you're used to seeing over buttons, scrollers, and many other objects in the window system.

See Also
`IBeamCursor` (page 482)
`currentCursor` (page 481)
`hotSpot` (page 484)

closedHandCursor

Returns the closed-hand system cursor.

```
public static NSCursor closedHandCursor()
```

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

crosshairCursor

Returns the cross-hair system cursor.

```
public static NSCursor crosshairCursor()
```

Discussion
This cursor is used for situations when precise location is required (where the lines cross is the hot spot).

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

currentCursor

Returns the cursor currently displayed on the screen.

```
public static NSCursor currentCursor()
```

See Also
`set` (page 486)
`push` (page 486)
`pop` (page 486)
`mouseEntered` (page 485)
`mouseExited` (page 486)

disappearingItemCursor

Returns a cursor indicating that the current operation will result in a disappearing item (for example, when dragging an item from the dock or a toolbar).

```
public static NSCursor disappearingItemCursor()
```

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

hide

Makes the current cursor invisible.

```
public static void hide()
```

Discussion

If another cursor becomes current, that cursor will be invisible, too. It will remain invisible until you invoke the `unhide` (page 484) method.

`hide` (page 481) overrides `setHiddenUntilMouseMoves` (page 484).

IBeamCursor

Returns a cursor that looks like a capital I with a tiny crossbeam at its middle.

```
public static NSCursor IBeamCursor()
```

Discussion

This is the cursor that you're used to seeing over editable or selectable text. The I-beam cursor's default hot spot is where the crossbeam intersects the I.

See Also

`arrowCursor` (page 480)
`currentCursor` (page 481)

openHandCursor

Returns the open-hand system cursor.

```
public static NSCursor openHandCursor()
```

Availability

Available in Mac OS X v10.3 and later.

pointingHandCursor

Returns the pointing-hand system cursor.

```
public static NSCursor pointingHandCursor()
```

Discussion

The tip of the pointing finger is the hot spot.

Availability

Available in Mac OS X v10.3 and later.

popCursor

Pops the current cursor off the top of the stack.

```
public static void popCursor()
```

Discussion

The new object on the top of the stack becomes the current cursor. If the current cursor is the only cursor on the stack, this method does nothing.

See Also
`push` (page 486)

resizeDownCursor

```
public static NSCursor resizeDownCursor()
```

Discussion
Returns the resize-down system cursor.

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

resizeLeftCursor

```
public static NSCursor resizeLeftCursor()
```

Discussion
Returns the resize-left system cursor.

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

resizeLeftRightCursor

Returns the resize-left-and-right system cursor.

```
public static NSCursor resizeLeftRightCursor()
```

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

resizeRightCursor

Returns the resize-right system cursor.

```
public static NSCursor resizeRightCursor()
```

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

resizeUpCursor

Returns the resize-up system cursor.

```
public static NSCursor resizeUpCursor()
```

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

resizeUpDownCursor

Returns the resize-up-and-down system cursor.

```
public static NSCursor resizeUpDownCursor()
```

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

setHiddenUntilMouseMoves

```
public static void setHiddenUntilMouseMoves(boolean flag)
```

Discussion
If *flag* is true, hides the cursor. The cursor remains invisible until one of the following occurs:

- The mouse moves.
- You invoke the method again, with *flag* set to false.

Do not try to counter this method by invoking unhide (page 484). The results are undefined.

See Also
hide (page 481)

unhide

Negates an earlier call to hide (page 481) by showing the current cursor.

```
public static void unhide()
```

See Also
setHiddenUntilMouseMoves (page 484)
hide (page 481)

Instance Methods

hotSpot

Returns the position of the hot spot, specified according to the cursor's flipped coordinate system.

```
public NSPoint hotSpot()
```

Discussion
For a more complete explanation, see the class description.

Note that an NSCursor object is immutable: you cannot change its hot spot after it's created. Instead, use a constructor to create a new cursor with the new settings.

image

Returns the image for the receiver, or `null` if none exists.

```
public NSImage image()
```

Discussion
Note that an NSCursor object is immutable: you cannot change its image after it's created. Instead, use a constructor to create a new cursor with the new settings.

isSetOnMouseEntered

```
public boolean isSetOnMouseEntered()
```

Discussion
Returns `true` if the receiver will become current when it receives a `mouseEntered` (page 485) message; otherwise, returns `false`.

To receive such a message, the receiver must first be assigned a cursor rectangle. This assignment can be made using NSView's `addCursorRect` (page 1739) method. For a more complete explanation, see the class description.

See Also
`setOnMouseEntered` (page 487)
`isSetOnMouseExited` (page 485)

isSetOnMouseExited

```
public boolean isSetOnMouseExited()
```

Discussion
Returns `true` if the receiver becomes current when it receives a `mouseExited` (page 486) message; otherwise, returns `false`.

To receive such a message, the receiver must first be assigned a cursor rectangle. This assignment can be made using NSView's `addCursorRect` (page 1739) method. For a more complete explanation, see the class description.

See Also
`setOnMouseExited` (page 487)

mouseEntered

```
public void mouseEntered(NSEvent anEvent)
```

Discussion
This message is automatically sent to the receiver when the cursor enters a tracking rectangle owned by the receiver, generating *`anEvent`*. If used after `setOnMouseEntered` (page 487) has been called with an argument of `true`, `mouseEntered` can make the receiver the current cursor.

In your programs, you won't invoke `mouseEntered` explicitly. It's only included in the class interface so you can override it.

For a more complete explanation, see "Handling Tracking-Rectangle and Cursor-Update Events in Views".

See Also
`isSetOnMouseEntered` (page 485)
`mouseExited` (page 486)

mouseExited

```
public void mouseExited(NSEvent anEvent)
```

Discussion
This message is automatically sent to the receiver when the cursor exits a tracking rectangle owned by the receiver, generating *anEvent*. Like `mouseEntered` (page 485), it is part of the class interface only so you can override it.

For a more complete explanation, see "Handling Tracking-Rectangle and Cursor-Update Events in Views".

See Also
`setOnMouseExited` (page 487)
`isSetOnMouseExited` (page 485)

pop

Sends a `popCursor` (page 482) message to the receiver's class.

```
public void pop()
```

See Also
`push` (page 486)
`pop` (page 486)

push

Puts the receiver on top of the cursor stack and makes it the current cursor.

```
public void push()
```

See Also
`pop` (page 486)
`pop` (page 486)

set

Makes the receiver the current cursor.

```
public void set()
```

See Also

`currentCursor` (page 481)

setOnMouseEntered

```
public void setOnMouseEntered(boolean flag)
```

Discussion

If *flag* is `true`, the receiver accepts future `mouseEntered` (page 485) event messages; otherwise it ignores them. Accepting `mouseEntered` (page 485) event messages allows the cursor to be made the current cursor when the cursor enters a view's cursor rectangle.

See Also

`mouseEntered` (page 485)

setOnMouseExited

```
public void setOnMouseExited(boolean flag)
```

Discussion

If *flag* is `true`, the receiver accepts future `mouseExited` (page 486) event messages; otherwise it ignores them. Accepting `mouseExited` (page 486) event messages allows the cursor to be made the current cursor when the cursor exits a view's cursor rectangle.

NSCustomImageRep

Inherits from	NSImageRep : NSObject
Implements	NSCoding (NSImageRep)
Package:	com.apple.cocoa.application
Companion guide	Drawing and Images

Overview

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.

Tasks

Constructors

`NSCustomImageRep` (page 489)
Throws an exception. Use the other constructor instead.

Identifying the Object

`delegate` (page 490)
Returns the delegate object that renders the image for the receiver.

`drawMethod` (page 490)
Returns the associated draw method selector.

Constructors

NSCustomImageRep

Throws an exception. Use the other constructor instead.

```
public NSCustomImageRep()
```

Creates a new NSCustomImageRep instance, so that it delegates responsibility for rendering the image to *anObject*.

```
public NSCustomImageRep(NSSelector aMethod, Object anObject)
```

Discussion

When the NSCustomImageRep receives a `draw` message, it will in turn send a message to *anObject* to perform *aMethod*. *aMethod* should take only one argument, the NSCustomImageRep. It should draw the image at location (0.0, 0.0) in the current coordinate system.

Instance Methods

delegate

Returns the delegate object that renders the image for the receiver.

```
public Object delegate()
```

drawMethod

Returns the associated draw method selector.

```
public NSSelector drawMethod()
```

NSDatePicker

Inherits from	NSControl : NSView : NSResponder : NSObject
Implements	NSCoding (NSResponder)
Package:	com.apple.cocoa.application
Availability	Available in Mac OS X v10.4 and later.

Overview

NSDatePicker is a subclass of NSControl that provides for visually displaying and editing an NSDate instance.

NSDatePicker uses an NSDatePickerCell to implement much of the control's functionality. NSDatePicker provides cover methods for most of NSDatePickerCell methods, which invoke the corresponding cell method.

Tasks

Constructors

`NSDatePicker` (page 493)

Appearance

`isBezeled` (page 495)

`setBezeled` (page 496)
  Specifies whether the receiver has a bezeled border.

`isBordered` (page 495)

`setBordered` (page 497)
  Specifies whether the receiver draws a plain border.

`backgroundColor` (page 493)
  Returns the background color of the receiver.

`setBackgroundColor` (page 496)
  Sets the receiver's background color to *`color`*.

drawsBackground (page 495)

setDrawsBackground (page 498)

textColor (page 500)
Returns the text color of the receiver.

setTextColor (page 499)
Sets the text color of the receiver to *color*.

datePickerStyle (page 494)
Returns the receiver's date picker style.

setDatePickerStyle (page 497)
Sets the receiver's date picker style to *newStyle*.

Range Mode Control

datePickerMode (page 494)
Returns the receiver's date picker mode.

setDatePickerMode (page 497)
Sets the receiver's date picker mode to *newMode*.

timeZone (page 500)
Returns the receiver's time zone.

setTimeZone (page 500)
Sets the receiver's time zone to *newTimeZone*.

Object Value Access

dateValue (page 494)
Returns the receiver's date.

setDateValue (page 498)
Sets the date of the receiver to *newStartDate*.

timeInterval (page 500)
Returns the time interval that represents the date range.

setTimeInterval (page 499)
Sets the time interval of the date range to *newTimeInterval*.

datePickerElements (page 494)
Returns the receiver's date picker elements flags.

setDatePickerElements (page 497)
Sets the date picker elements displayed by the receiver to those specified by *elementFlags*.

Constraints on Displayable/selectable Range

minDate (page 496)
Returns the minimum date value that the receiver allows as input.

`setMinDate` (page 499)
Sets the minimum date allowed as input by the receiver to *date*.

`maxDate` (page 496)
Returns the maximum date value that the receiver allows as input.

`setMaxDate` (page 498)
Sets the maximum date allowed as input by the receiver to *date*.

Getting and Setting the Delegate

`delegate` (page 495)
Returns the receiver's delegate.

`setDelegate` (page 498)
Sets the receiver's delegate to *anObject*.

Constructors

NSDatePicker

```
NSDatePicker()
```

Discussion
Creates an NSDatePicker object with a zero-sized frame rectangle.

```
public NSDatePicker(NSRect frameRect)
```

Discussion
Creates an NSDatePicker object with *frameRect* as its frame rectangle.

Instance Methods

backgroundColor

Returns the background color of the receiver.

```
public native NSColor backgroundColor()
```

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

See Also
`setBackgroundColor` (page 496)

datePickerElements

Returns the receiver's date picker elements flags.

```
public int datePickerElements()
```

Discussion

See "Constants" (page 511) for a description of the possible values.

Availability

Available in Mac OS X v10.4 and later.

See Also

`setDatePickerElements` (page 497)

datePickerMode

Returns the receiver's date picker mode.

```
public int datePickerMode()
```

Discussion

See "Constants" (page 511) for a description of the possible values.

Availability

Available in Mac OS X v10.4 and later.

See Also

`setDatePickerMode` (page 497)

datePickerStyle

Returns the receiver's date picker style.

```
public int datePickerStyle()
```

Discussion

See "Constants" (page 511) for a description of the possible values.

Availability

Available in Mac OS X v10.4 and later.

See Also

`datePickerStyle` (page 494)

dateValue

Returns the receiver's date.

```
public NSDate dateValue()
```

Availability

Available in Mac OS X v10.4 and later.

See Also

`setDateValue` (page 498)

delegate

Returns the receiver's delegate.

```
public Object delegate()
```

Availability

Available in Mac OS X v10.4 and later.

See Also

`setDelegate` (page 498)

drawsBackground

```
public boolean drawsBackground()
```

Discussion

Returns whether the receiver draws the background.

Availability

Available in Mac OS X v10.4 and later.

See Also

`setDrawsBackground` (page 498)

isBezeled

```
public boolean isBezeled()
```

Discussion

Returns whether the receiver has a bezeled border.

Availability

Available in Mac OS X v10.4 and later.

See Also

`setBezeled` (page 496)

isBordered

```
public boolean isBordered()
```

Discussion

Returns whether the receiver has a plain border.

Availability

Available in Mac OS X v10.4 and later.

See Also
`setBordered` (page 497)

maxDate

Returns the maximum date value that the receiver allows as input.

```
public NSDate maxDate()
```

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

See Also
`setMaxDate` (page 498)

minDate

Returns the minimum date value that the receiver allows as input.

```
public NSDate minDate()
```

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

See Also
`setMinDate` (page 499)

setBackgroundColor

Sets the receiver's background color to *`color`*.

```
public void setBackgroundColor(NSColor color)
```

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

setBezeled

Specifies whether the receiver has a bezeled border.

```
public void setBezeled(boolean flag)
```

Discussion
The *`flag`* parameter specifies the Boolean value.

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

See Also
`isBezeled` (page 495)

setBordered

Specifies whether the receiver draws a plain border.

```
public void setBordered(boolean flag)
```

Discussion
The *flag* parameter specifies the Boolean value.

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

See Also
`isBordered` (page 495)

setDatePickerElements

Sets the date picker elements displayed by the receiver to those specified by *elementFlags*.

```
public void setDatePickerElements(int elementFlags)
```

Discussion
See "Constants" (page 511) for a description of the possible values.

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

See Also
`datePickerElements` (page 494)

setDatePickerMode

Sets the receiver's date picker mode to *newMode*.

```
public void setDatePickerMode(int newMode)
```

Discussion
See "Constants" (page 511) for a description of the possible values.

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

See Also
`datePickerMode` (page 494)

setDatePickerStyle

Sets the receiver's date picker style to *newStyle*.

```
public void setDatePickerStyle(int newStyle)
```

Discussion

See "Constants" (page 511) for a description of the possible values.

Availability

Available in Mac OS X v10.4 and later.

See Also

`datePickerStyle` (page 494)

setDateValue

Sets the date of the receiver to *`newStartDate`*.

```
public void setDateValue(NSDate newStartDate)
```

Availability

Available in Mac OS X v10.4 and later.

See Also

`dateValue` (page 494)

setDelegate

Sets the receiver's delegate to *`anObject`*.

```
public void setDelegate(Object anObject)
```

Availability

Available in Mac OS X v10.4 and later.

See Also

`delegate` (page 495)

setDrawsBackground

```
public void setDrawsBackground(boolean flag)
```

Discussion

Sets whether the receiver draws the background.

Availability

Available in Mac OS X v10.4 and later.

See Also

`drawsBackground` (page 495)

setMaxDate

Sets the maximum date allowed as input by the receiver to *`date`*.

```
public void setMaxDate(NSDate date)
```

Discussion

Passing `null` for *`date`* allows any date as the maximum value.

Availability

Available in Mac OS X v10.4 and later.

See Also

`maxDate` (page 496)

setMinDate

Sets the minimum date allowed as input by the receiver to *`date`*.

```
public void setMinDate(NSDate date)
```

Discussion

Passing `null` for *`date`* allows any date as the minimum value.

Availability

Available in Mac OS X v10.4 and later.

See Also

`minDate` (page 496)

setTextColor

Sets the text color of the receiver to *`color`*.

```
public void setTextColor(NSColor color)
```

Availability

Available in Mac OS X v10.4 and later.

See Also

`textColor` (page 500)

setTimeInterval

Sets the time interval of the date range to *`newTimeInterval`*.

```
public void setTimeInterval(double newTimeInterval)
```

Discussion

The time interval is only applicable when the receiver is in `NSDateRangeMode`.

Availability

Available in Mac OS X v10.4 and later.

See Also

`timeInterval` (page 500)

setTimeZone

Sets the receiver's time zone to *newTimeZone*.

```
public void setTimeZone(NSTimeZone newTimeZone)
```

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

See Also
timeZone (page 500)

textColor

Returns the text color of the receiver.

```
public NSColor textColor()
```

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

See Also
setTextColor (page 499)

timeInterval

Returns the time interval that represents the date range.

```
public double timeInterval()
```

Discussion
The date range begins at the date returned by dateValue (page 494). This method returns 0 when the receiver is not in the NSDateRangeMode mode.

Currently this method always returns 0.

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

See Also
setTimeInterval (page 499)

timeZone

Returns the receiver's time zone.

```
public NSTimeZone timeZone()
```

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

See Also

setTimeZone (page 500)

NSDatePickerCell

Inherits from	NSActionCell : NSCell : NSObject
Implements	NSCoding (NSCell)
Package:	com.apple.cocoa.application
Availability	Available in Mac OS X v10.4 and later.

Overview

An NSDatePickerCell controls the behavior of an NSDatePicker control, or of a single date picker cell in a matrix.

Tasks

Constructors

`NSDatePickerCell` (page 505)
    Creates an empty NSDatePickerCell.

Appearance

`backgroundColor` (page 505)
    Returns the background color of the receiver.

`setBackgroundColor` (page 507)
    Sets the receiver's background color to *color*.

`textColor` (page 510)
    Returns the text color of the receiver.

`setTextColor` (page 509)
    Sets the text color of the receiver to *color*.

`datePickerStyle` (page 506)
    Returns the receiver's date picker style.

`setDatePickerStyle` (page 508)
    Sets the receiver's date picker style to *newTyle*.

Range Mode Control

`datePickerMode` (page 506)
    Returns the receiver's date picker mode.

`setDatePickerMode` (page 508)
    Sets the receiver's date picker mode to *newMode*.

Object Value Access

`dateValue` (page 506)
    Returns the receiver's date.

`setDateValue` (page 508)
    Sets the date of the receiver to *newStartDate*.

`timeInterval` (page 510)
    Returns the time interval that represents the date range.

`setTimeInterval` (page 510)
    Sets the time interval of the date range to *newTimeInterval*.

`timeZone` (page 511)
    Returns the receiver's time zone.

`setTimeZone` (page 510)
    Sets the receiver's time zone to *newTimeZone*.

`datePickerElements` (page 505)
    Returns the receiver's date picker elements flags.

`setDatePickerElements` (page 507)
    Sets the date picker elements displayed by the receiver to those specified by *elementFlags*.

Constraints on the Minimum and Maximum Date Range

`minDate` (page 507)
    Returns the minimum date value that the receiver allows as input.

`setMinDate` (page 509)
    Sets the minimum date allowed as input by the receiver to *date*.

`maxDate` (page 507)
    Returns the maximum date value that the receiver allows as input.

`setMaxDate` (page 509)
    Sets the maximum date allowed as input by the receiver to *date*.

Getting and Setting the Delegate

`delegate` (page 507)
    Returns the receiver's delegate.

`setDelegate` (page 509)
    Sets the receivers delegate to *anObject*.

Validation

`datePickerCellValidateProposedDateValue` (page 512) *delegate method*
The delegate receives this message each time the user attempts to change the receiver's value, allowing the delegate the opportunity to override the change.

Constructors

NSDatePickerCell

Creates an empty NSDatePickerCell.

```
public NSDatePickerCell()
```

Creates an NSDatePickerCell initialized with *aString*.

```
public NSDatePickerCell(String aString)
```

Creates an NSDatePickerCell initialized with *anImage*.

```
public NSDatePickerCell(NSImage anImage)
```

Discussion
If *anImage* is `null`, no image is set.

Instance Methods

backgroundColor

Returns the background color of the receiver.

```
public NSColor backgroundColor()
```

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

See Also
`setBackgroundColor` (page 507)

datePickerElements

Returns the receiver's date picker elements flags.

```
public int datePickerElements()
```

Discussion
See "Constants" (page 511) for a description of the possible values.

Availability

Available in Mac OS X v10.4 and later.

See Also

`setDatePickerElements` (page 507)

datePickerMode

Returns the receiver's date picker mode.

```
public int datePickerMode()
```

Discussion

See "Constants" (page 511) for a description of the possible values.

Availability

Available in Mac OS X v10.4 and later.

See Also

`setDatePickerMode` (page 508)

datePickerStyle

Returns the receiver's date picker style.

```
public int datePickerStyle()
```

Discussion

See "Constants" (page 511) for a description of the possible values.

Availability

Available in Mac OS X v10.4 and later.

See Also

`setDatePickerStyle` (page 508)

dateValue

Returns the receiver's date.

```
public NSDate dateValue()
```

Availability

Available in Mac OS X v10.4 and later.

See Also

`setDateValue` (page 508)

delegate

Returns the receiver's delegate.

```
public Object delegate()
```

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

See Also
`setDelegate` (page 509)

maxDate

Returns the maximum date value that the receiver allows as input.

```
public NSDate maxDate()
```

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

See Also
`setMaxDate` (page 509)

minDate

Returns the minimum date value that the receiver allows as input.

```
public NSDate minDate()
```

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

See Also
`setMinDate` (page 509)

setBackgroundColor

Sets the receiver's background color to *`color`*.

```
public void setBackgroundColor(NSColor color)
```

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

See Also
`backgroundColor` (page 505)

setDatePickerElements

Sets the date picker elements displayed by the receiver to those specified by *`elementFlags`*.

```
public void setDatePickerElements(int elementFlags)
```

Discussion
See “Constants” (page 511) for a description of the possible values.

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

See Also
`datePickerElements` (page 505)

setDatePickerMode

Sets the receiver’s date picker mode to *newMode*.

```
public void setDatePickerMode(int newMode)
```

Discussion
See “Constants” (page 511) for a description of the possible values.

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

See Also
`datePickerMode` (page 506)

setDatePickerStyle

Sets the receiver’s date picker style to *newTyle*.

```
public void setDatePickerStyle(int newStyle)
```

Discussion
See “Constants” (page 511) for a description of the possible values.

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

See Also
`datePickerStyle` (page 506)

setDateValue

Sets the date of the receiver to *newStartDate*.

```
public void setDateValue(NSDate newStartDate)
```

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

See Also
`dateValue` (page 506)

setDelegate

Sets the receivers delegate to *anObject*.

```
public void setDelegate(Object anObject)
```

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

See Also
`delegate` (page 507)

setMaxDate

Sets the maximum date allowed as input by the receiver to *date*.

```
public void setMaxDate(NSDate date)
```

Discussion
Passing `null` for *date* allows any date as the maximum value.

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

See Also
`maxDate` (page 507)

setMinDate

Sets the minimum date allowed as input by the receiver to *date*.

```
public void setMinDate(NSDate date)
```

Discussion
Passing `null` for *date* allows any date as the minimum value.

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

See Also
`minDate` (page 507)

setTextColor

Sets the text color of the receiver to *color*.

```
public void setTextColor(NSColor color)
```

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

See Also
`textColor` (page 510)

setTimeInterval

Sets the time interval of the date range to *`newTimeInterval`*.

```
public void setTimeInterval(double newTimeInterval)
```

Discussion
The time interval is only applicable when the receiver is in `NSDateRangeMode`.

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

See Also
`timeInterval` (page 510)

setTimeZone

Sets the receiver's time zone to *`newTimeZone`*.

```
public void setTimeZone(NSTimeZone newTimeZone)
```

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

See Also
`timeZone` (page 511)

textColor

Returns the text color of the receiver.

```
public NSColor textColor()
```

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

See Also
`setTextColor` (page 509)

timeInterval

Returns the time interval that represents the date range.

```
public double timeInterval()
```

Discussion

The date range begins at the date returned by `dateValue` (page 506). This method returns 0 when the receiver is not in the `NSDateRangeMode` mode.

Currently this method always returns 0.

Availability

Available in Mac OS X v10.4 and later.

See Also

`setTimeInterval` (page 510)

timeZone

Returns the receiver's time zone.

```
public NSTimeZone timeZone()
```

Availability

Available in Mac OS X v10.4 and later.

See Also

`setTimeZone` (page 510)

Constants

The NSDatePickerStyle constants define the visual appearance of the NSDatePickerCell. These values are used by `datePickerStyle` (page 506) and `setDatePickerStyle` (page 508):

Constant	Description
`TextFieldAndStepperDatePickerStyle`	Provide a text field and stepper style interface. (Available on Mac OS X v10.4 and later.)
`ClockAndCalendarDatePickerStyle`	Provide a visual clock and calendar style interface. (Available on Mac OS X v10.4 and later.)

The NSDatePickerMode constants define whether the control provides a single date, or a range of dates. These values are used by `datePickerMode` (page 506) and `setDatePickerMode` (page 508). Currently only `NSSingleDateMode` is implemented.

Constant	Description
`SingleDateMode`	Allow selection of a single date. (Available in Mac OS X v10.4 and later.)
`RangeDateMode`	Allow selection of a range of dates. (Available in Mac OS X v10.4 and later.)

The NSDatePickerElementFlags allow you to specify the date and time elements that the NSDatePickerCell can edit by combining these constants using the C bitwise OR operator. These values are used by `datePickerElements` (page 505) and `setDatePickerElements` (page 507):

Constant	Description
`HourMinuteDatePickerElementFlag`	Display and allow editing of the hour and minute elements of the date. (Available in Mac OS X v10.4 and later.)
`HourMinuteSecondDatePickerElementFlag`	Display and allow editing of the hour, minute and second elements of the date. (Available in Mac OS X v10.4 and later.)
`TimeZoneDatePickerElementFlag`	Display and allow editing of the time zone. (Available in Mac OS X v10.4 and later.)
`YearMonthDatePickerElementFlag`	Display and allow editing of the year and month elements of the date. (Available in Mac OS X v10.4 and later.)
`YearMonthDayDatePickerElementFlag`	Display and allow editing of the year, month and day elements of the date. (Available in Mac OS X v10.4 and later.)
`EraDatePickerElementFlag`	Display and allow editing of the era of the date, if applicable. (Available in Mac OS X v10.4 and later.)

Delegate Methods

datePickerCellValidateProposedDateValue

The delegate receives this message each time the user attempts to change the receiver's value, allowing the delegate the opportunity to override the change.

```
public abstract NSDate datePickerCellValidateProposedDateValue(NSDatePickerCell
    aDatePickerCell, NSDate proposedDateValue)
```

Discussion

The proposed new date is passed by-reference in *proposedDateValue*. The delegate may change the value before returning. When returning a new *proposedDateValue*, the NSDate instance should be autoreleased, and the *proposedDateValue* should not be released by the delegate.

The *proposedDateValue* is guaranteed to lie between the dates returned by `minDate` (page 507) and `maxDate` (page 507). If you modify this value, you should ensure that the new value is within the appropriate range.

Availability

Available in Mac OS X v10.4 and later.

NSDocument

Inherits from	NSObject
Package:	com.apple.cocoa.application
Companion guide	Document-Based Applications Overview

Class at a Glance

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

Principal Attributes

- Window controllers
- Filenames
- Document types
- Print information

Commonly Used Methods

`dataOfType` (page 525)
Returns the document’s data in a specified type.

`readFromDataOfType` (page 537)
Sets the contents of this document by reading from data of a specified type.

`writeToFile` (page 552)
Writes the document’s data to a file.

`writeToURLOfType` (page 553)
Writes the document’s data to a URL.

`readFromURLOfType` (page 538)
Reads the document’s data from a file.

`windowNibName` (page 551)
Returns the name of the document’s sole nib file (resulting in the creation of a window controller for the window in that file).

`makeWindowControllers` (page 533)
Creates and returns the window controllers used to manage document windows.

Overview

NSDocument is an abstract class that defines the interface for documents. In a functional sense, a document is a repeatable container for a unique body of information identified by a name under which it is stored. In the context of the Application Kit, a document is an instance of an NSDocument subclass that knows how to represent internally, in one or more formats, the persistent data displayed in windows. A document can read that data from files and write it to files. It is also the first-responder target for many menu commands related to documents, such as Save Document, Revert Document, and Print Document. (When going up the responder chain, the Application Kit queries a window's NSDocument, if it exists, just after it queries the window delegate, if that is different from the NSDocument.) A document manages its window's edited status and is set up to perform undo and redo operations. When a window is closing, the document is asked before the window delegate to approve the closing.

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

To create a useful NSDocument subclass, you must override some primitive methods and might want to override others. The NSDocument class itself knows how to handle document data as undifferentiated "lumps"; although it understands that these lumps are typed, it knows nothing about particular types. In their overrides of the data-based primitive methods, subclasses must add the knowledge of particular types and how data of the document's native type is structured internally and represented in document windows. Subclasses are also responsible for the creation of the window controllers that manage document windows and for the implementation of undo and redo. NSDocument takes care of much of the rest, including running Open and Save panels and generally managing the state of the document. See "Creating a Subclass of NSDocument" for more on creating subclasses of NSDocument, particularly the list of primitive methods that subclasses must override and those that may be overridden.

Writing of HFS Creator and File Type Codes

The `fileAttributesToWriteToFile` (page 526) method can be overridden to specify that a creator code and/or file type code should be written to a file as it is being saved. See NSPathUtilities for descriptions of the new `FileHFSCreatorCode` and `FileHFSTypeCode` file attributes. NSDocument's implementation of `fileAttributesToWriteToFile` returns zeroed-out creator and file type codes, effectively excluding creator code and file type code from the attribute preservation described in `fileAttributesToWriteToFile`.

NSDocument Saving Behavior

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

- Creation date
- Permissions/privileges
- Location of the document's icon in its parent folder's Icon View Finder window

- Value of the document's Show Extension setting

Care is also taken to save documents in a way that does not break any user-created aliases that may point to documents. As a result, some methods in any class of NSDocument may be invoked with parameters that do not have the same meaning as they did in early releases of Mac OS X. It is important that overrides of `writeToFile` (page 552) make no assumptions about the file paths passed as parameters, including:

- The location to which the file is being written. This location might be a hidden temporary directory.
- The name of the file being written. It is possible that this file has no obvious relation to the document name.
- The relation of any file being passed, including the original file, to the return value of `fileName` (page 527).

Tasks

Constructors

`NSDocument` (page 521)
    Creates an empty NSDocument object.

Initializing an NSDocument

`initWithContentsOfURLOfType` (page 531)
    Initializes a document located by a URL of a specified type.

`initForURLWithContentsOfURLOfType` (page 530)
    Initializes a document located by a URL.

`initWithType` (page 531)
    Initializes a document of a specified type, and returns it if successful.

Loading and Representing Document Data

`dataOfType` (page 525)
    Creates and returns a data object that contains the contents of the document, formatted to a specified type.

`fileWrapperOfType` (page 529)
    Creates and returns a file wrapper that contains the contents of the document.

`readFromDataOfType` (page 537)
    Sets the contents of this document by reading from data of a specified type and returns `true` if successful.

Creating and Managing Window Controllers

makeWindowControllers (page 533)
: Subclasses may override this method to create the initial window controller(s) for the document.

windowNibName (page 551)
: Overridden by subclasses to return the name of the document's sole nib file.

windowControllerDidLoadNib (page 550)
: Sent after *windowController* loads a nib file if the receiver is the nib file's owner.

windowControllerWillLoadNib (page 550)
: Sent before *windowController* loads a nib file if the receiver is the nib file's owner.

windowControllers (page 550)
: Returns the receiver's current window controllers.

addWindowController (page 523)
: Adds the window controller *aController* to the list of window controllers associated with the receiver.

removeWindowController (page 538)
: Removes *windowController* from the receiver.

Managing Document Windows

showWindows (page 548)
: Displays all of the document's windows, bringing them to the front and making them main or key as necessary.

displayName (page 526)
: Returns the name of the receiver as displayed in the title bars of the document's windows and in alert dialogs related to the document.

setWindow (page 547)
: Sets the window Interface Builder outlet of this class.

windowForSheet (page 551)
: Returns the most appropriate window, of the windows associated with the receiver, to use as the parent window of a document-modal sheet.

Reading from and Writing to Files

readFromFileWrapperOfType (page 537)
: Set the contents of this document by reading from a file wrapper of a specified type, and return true if successful.

writeToFile (page 552)
: Writes document data of type *docType* to the file *fileName*, returning whether the operation was successful.

fileModificationDate (page 527)
: Returns the last known modification date of the document's on-disk representation.

setFileModificationDate (page 544)
: Sets the last known modification date of the document's on-disk representation to *modificationDate*.

shouldRunSavePanelWithAccessoryView (page 548)
: Returns true by default; as a result, when NSDocument displays the Save panel, it includes an accessory view containing a pop-up list of supported writable document types.

keepBackupFile (page 532)
: Returns whether the receiver should keep the backup files created before document data is written to a file (false by default).

Reading from and Writing to URLs

readFromURLOfType (page 538)
: Sets the contents of this document by reading from a file or file package located by a URL, of a specified type, and returns true if successful.

writeToURLOfType (page 553)
: Writes the contents of the file to a file or file package located by a URL, formatted to a specified type, and returns true if successful.

writeSafelyToURLOfType (page 552)
: Writes the contents of the document to a file or file package.

setFileURL (page 545)
: Sets the location of the document's on-disk representation.

fileURL (page 529)
: Returns the location of the document's on-disk representation.

fileAttributesToWriteToURLOfType (page 526)
: As a file is being saved, returns the attributes that should be written to a file or file package located by a URL, formatted to a specified type, for a particular kind of save operation.

saveToURLOfType (page 543)
: Saves the contents of the document to a file or file package located by a URL, formatted to a specified type, for a particular kind of save operation, and returns true if successful.

Autosaving

hasUnautosavedChanges (page 530)
: Return true if the document has changes that have not been autosaved, false otherwise, as determined by the history of previous invocations of updateChangeCount (page 549).

autosaveDocument (page 523)
: Autosaves the document’s contents at an appropriate location.

autosavingFileType (page 524)
: Returns the document type that should be used for an autosave operation.

setAutosavedContentsFileURL (page 544)
: Sets the location of the most recently autosaved document contents.

autosavedContentsFileURL (page 523)
: Returns the location of the most recently autosaved document contents.

Managing Document Status

isDocumentEdited (page 531)
: Returns true if the receiver has been edited since it was last saved or if the document is new; otherwise, returns false.

updateChangeCount (page 549)
: Updates the receiver's change count according to *changeType*.

fileNameExtensionWasHiddenInLastRunSavePanel (page 528)
: Returns true if a Save panel has been presented by this document, and the user chose to hide the name extension of the file that was selected in that Save panel.

Responding to User Actions

prepareSavePanel (page 533)
: Invoked by runModalSavePanel (page 541) to do any customization of the Save panel *savePanel*.

printDocument (page 535)
: Prints the receiver in response to the user choosing the Print menu command.

runPageLayout (page 541)
: The action method invoked in the receiver as first responder when the user chooses the Page Setup menu command.

revertDocumentToSaved (page 539)
: The action of the File menu item Revert to Saved in a document-based application.

saveDocument (page 542)
: The action method invoked in the receiver as first responder when the user chooses the Save menu command.

saveDocumentAs (page 542)
: The action method invoked in the receiver as first responder when the user chooses the Save As menu command.

saveDocumentTo (page 543)
: The action method invoked in the receiver as first responder when the user chooses the Save To menu command.

Closing Documents

close (page 525)
: Closes all windows owned by the receiver and removes the receiver from the list of documents maintained by the document controller, which consequently releases it.

Reverting Documents

revertToContentsOfURLOfType (page 539)
: Discards all unsaved document modifications and replaces the document's contents by reading a file or file package located by a URL of a specified type and returns true if successful.

Printing Documents

printInfo (page 536)
Returns the receiver's customized NSPrintInfo object or the default NSPrintInfo instance.

setPrintInfo (page 546)
Sets the receiver's NSPrintInfo object to *printInfo*; this object is used in laying out the document for printing.

preparePageLayout (page 533)
Invoked by runModalPageLayoutWithPrintInfo (page 540) and runModalPageLayout (page 540) to do any customization of the Page Layout panel *pageLayout*, such as adding an accessory view.

runModalPageLayout (page 540)
Runs the modal page layout panel with the receiver's printing information object (*printInfo*).

runModalPrintOperation (page 541)
Runs a print operation *printOperation* modally.

shouldChangePrintInfo (page 547)
Returns whether the receiver should allow changes to the default NSPrintInfo object *newPrintInfo* used in printing the document.

printDocumentWithSettings (page 535)
Prints the document.

printOperationWithSettings (page 536)
Creates a print operation and returns it if successful.

Handling Errors

lastError (page 532)
Returns the NSError object that was most recently set.

presentErrorModalForWindow (page 534)
Presents an error alert to the user as a modal panel.

presentError (page 534)
Presents an error alert to the user as a modal panel.

setLastError (page 546)
Sets the NSError object that will be returned by the lastError member function.

willPresentError (page 549)
Called when the receiver is about to present an error. Returns the error that should actually be presented.

Working with Undo Manager

hasUndoManager (page 530)
Returns whether the receiver owns or should own an NSUndoManager.

setHasUndoManager (page 546)
Sets whether the receiver has its own NSUndoManager.

`setUndoManager` (page 547)
: Sets the undo manager owned by the receiver to *undoManager* and releases any undo manager currently owned by the receiver.

`undoManager` (page 548)
: Returns the NSUndoManager used by the receiver or `null` if the receiver should not own one.

Managing File Types

`setFileType` (page 545)
: Sets the document type under which the file is saved to *docType*.

`fileType` (page 528)
: Returns the document type under which the receiver is saved.

`fileTypeFromLastRunSavePanel` (page 528)
: Returns the file type that was last selected in the Save panel.

`isNativeType` (page 522)
: Returns whether document data of type *aType* is a native type—one the receiver can both read and write.

`readableTypes` (page 522)
: Returns the types of data the receiver can read natively and any types filterable to that native type.

`writableTypes` (page 522)
: Returns the types of data the receiver can write natively and any types filterable to that native type.

`writableTypesForSaveOperation` (page 551)
: Returns the names of the types to which this document can be saved for a specified kind of save operation.

Managing Menu Commands

`validateMenuItem` (page 549)
: Validates the Revert menu item and items selected from the Save panel's pop-up list of writable document types items.

Deprecated Methods

`canCloseDocument` (page 524)
: This method is no longer supported.

`dataRepresentationOfType` (page 525)

`fileAttributesToWriteToFile` (page 526)

`fileName` (page 527)

`fileNameFromRunningSavePanelForSaveOperation` (page 528) `fileWrapperRepresentationOfType` (page 529)

`loadDataRepresentation` (page 532)
: This method has been deprecated.

`loadFileWrapperRepresentation` (page 533)
: This method has been deprecated.

`printShowingPrintPanel` (page 536)
: This method has been deprecated.

`readFromFile` (page 537)
: This method has been deprecated.

`readFromURL` (page 538)
: This method has been deprecated.

`revertToSavedFromFile` (page 539)
: This method has been deprecated.

`revertToSavedFromURL` (page 540)
: This method has been deprecated.

`runModalSavePanel` (page 541)
: Runs the modal Save panel *savePanel* with accessory view *accessoryView* and returns the result constant (indicating the button clicked by the user).

`runModalPageLayoutWithPrintInfo` (page 540)
: This method has been deprecated. Use `runModalPageLayout` (page 540) instead.

`setFileName` (page 545)
: This method has been deprecated.

`saveToFile` (page 543)
: This method has been deprecated.

`shouldCloseWindowController` (page 547)
: This method variant is no longer supported. Instead use the other variant of this method.

`writeToURL` (page 553)

`writeWithBackupToFile` (page 554)
: This method has been deprecated.

Constructors

NSDocument

Creates an empty NSDocument object.

```
public NSDocument()
```

Creates an NSDocument object of document type *docType* containing data stored in the file *fileName*.

```
public NSDocument(String fileName, String docType)
```

Discussion
If the file cannot be opened, displays an alert dialog informing the user and then returns `null`. In opening the file, invokes the `readFromFile` (page 537) method. If the document successfully opens the file, it calls setFileName and `setFileType` (page 545) with *fileName* and *docType*, respectively, as arguments.

Creates an NSDocument of document type *docType* containing data stored at *aURL*.

```
public NSDocument(java.net.URL aURL, String docType)
```

Discussion
If the location cannot be opened, displays an alert dialog informing the user and then returns `null`. In opening the location, invokes the `readFromURL` (page 538) method. If the document successfully opens the file, it calls setFileName and `setFileType` (page 545) with *fileName* and *docType*, respectively, as arguments.

Static Methods

isNativeType

Returns whether document data of type *aType* is a native type—one the receiver can both read and write.

```
public static boolean isNativeType(String aType)
```

See Also
`readableTypes` (page 522)
`writableTypes` (page 522)

readableTypes

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

```
public static NSArray readableTypes()
```

See Also
`isNativeType` (page 522)
`writableTypes` (page 522)

writableTypes

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

```
public static NSArray writableTypes()
```

See Also
`isNativeType` (page 522)
`readableTypes` (page 522)

Instance Methods

addWindowController

Adds the window controller *aController* to the list of window controllers associated with the receiver.

```
public void addWindowController(NSWindowController aController)
```

Discussion

An NSDocument uses this list when it displays all document windows, sets window edited status upon an undo or redo operation, and modifies window titles. The method also sets the document outlet of the window controller to `this` if it is not already set. If you create window controllers by overriding `windowNibName` (page 551), this method is invoked automatically. If you create window controllers in `makeWindowControllers` (page 533) or in any other context, such as in response to a user event, you should invoke this method for each created window controller. To remove a window controller from the list of active controllers, send it NSWindowController's `close` (page 1891) message.

See Also

`setDocument` (page 1892) (NSWindowController)

autosavedContentsFileURL

Returns the location of the most recently autosaved document contents.

```
public URL autosavedContentsFileURL()
```

Discussion

The default implementation of this method just returns whatever was stored by a previous invocation of the default implementation of `setAutosavedContentsFileURL` (page 544).

Availability

Available in Mac OS X v10.4 and later.

See Also

`setAutosavedContentsFileURL` (page 544)

autosaveDocument

Autosaves the document's contents at an appropriate location.

```
public void autosaveDocument(Object delegate, NSSelector didAutosaveSelector, Object
      contextInfo)
```

Discussion

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

```
public void documentDidAutosave (NSDocument document, boolean
didAutosaveSuccessfully,  Object contextInfo)
```

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

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

See Also
`autosavedContentsFileURL` (page 523)

autosavingFileType

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

```
public String autosavingFileType()
```

Discussion
The default implementation just returns `[self fileType]`. You can override this method and return `null` in your override to completely disable autosaving of individual documents. You can also override it if your application defines a document type that is specifically designed for autosaving, for example, one that efficiently represents document content changes instead of complete document contents.

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

canCloseDocument

This method is no longer supported.

```
public boolean canCloseDocument()
```

Discussion
Instead use the three parameter version of this method.

If the receiver is not dirty, this method will immediately call the *`shouldCloseSelector`* callback on *`delegate`* with `true`.

```
public void canCloseDocument(Object delegate, NSSelector shouldCloseSelector, Object
     contextInfo)
```

Discussion
If the receiver is dirty, an alert will be presented giving the user a chance to save, not save, or cancel. If the user chooses to save, this method will save the document. If the save completes successfully, this method will call the callback with `true`. If the save is canceled or otherwise unsuccessful, this method will call the callback with `false`. This method may be called by `shouldCloseWindowController` (page 547). It is also called by NSDocumentController's `closeAllDocuments` (page 562). You should call it before you call `close` (page 525) if you are closing the document and want to give the user a chance save any edits. Pass the *`contextInfo`* object with the callback.

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

```
public void documentShouldClose (NSDocument doc, boolean shouldClose,  Object
 contextInfo)
```

Availability

Deprecated in Mac OS X v10.0.

Not supported in Mac OS X v10.4 and later.

close

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

```
public void close()
```

Discussion

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

This method may not always be called. Specifically, this method is not called when a user quits an application. Additional information on application termination can be found in Graceful Application Termination.

See Also

`canCloseDocument` (page 524)

`shouldCloseWindowController` (page 547)

dataOfType

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

```
public NSData dataOfType(String typeName)
```

Discussion

The *`typeName`* argument specifies the document type. If the data object cannot be created, returns `null`. The default implementation of this method throws an exception because at least one of the writing methods (this method, `writeToURLOfType` (page 553), or `fileWrapperOfType` (page 529)) must be overridden.

Availability

Available in Mac OS X v10.4 and later.

See Also

`writeToURLOfType` (page 553)

`fileWrapperOfType` (page 529)

dataRepresentationOfType

```
public NSData dataRepresentationOfType(String aType)
```

Discussion

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

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

Availability
Deprecated in Mac OS X v10.4.

See Also
`loadDataRepresentation` (page 532)

displayName

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

```
public String displayName()
```

Discussion
If the document has been saved, the display name is the last component of the directory location of the saved file (for example, "`MyDocument`" if the path is "`/tmp/MyDocument.rtf`"). If the document is new, NSDocument makes the display name "Untitled *n*," where *n* is a number in a sequence of new and unsaved documents. The displayable name also takes into account whether the document's filename extension should be hidden. Subclasses of NSWindowController can override `windowTitleForDocumentDisplayName` (page 1896) to modify the display name as it appears in window titles.

fileAttributesToWriteToFile

```
public NSDictionary fileAttributesToWriteToFile(String fullDocumentPath, String
    docType, int saveOperationType)
```

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

Availability
Deprecated in Mac OS X v10.4.

fileAttributesToWriteToURLOfType

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

```
public NSDictionary fileAttributesToWriteToURLOfType(URL absoluteURL, String
    typeName, int saveOperation, URL absoluteOriginalContentsURL)
```

Discussion
If not successful, returns `null`.The set of valid file attributes is a subset of those understood by the NSPathUtilities class. The default implementation of this method returns a dictionary with `FileHFSCreatorCode` and `FileHFSTypeCode` entries that have a value of 0 for `SaveOperation`, or a dictionary with an appropriate `FileExtensionHidden` entry for `SaveAsOperation` and `SaveToOperation`. You can override this method to customize the attributes that are written to document files.

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

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

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

An override of `writeSafelyToURLOfType` (page 552) should invoke this method and set the returned attributes on the written document file.

Implementers of overrides of this method should not assume that:

- The file pointed to by *`absoluteURL`* at the moment the method is invoked, if there is one, is related to the document itself. It may be an unrelated file that is about to be overwritten.
- The `fileURL` (page 529) or `fileType` (page 528) method will return anything useful at the moment.

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

fileModificationDate

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

```
public NSDate fileModificationDate()
```

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

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

See Also
`setFileModificationDate` (page 544)

fileName

```
public String fileName()
```

Discussion
Returns the filename (as a fully qualified path) under which the receiver has been saved.

Availability
Deprecated in Mac OS X v10.4.

See Also
`fileNameFromRunningSavePanelForSaveOperation` (page 528)
`setFileName` (page 545)

fileNameExtensionWasHiddenInLastRunSavePanel

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

```
public boolean fileNameExtensionWasHiddenInLastRunSavePanel()
```

Discussion
Returns `false` otherwise.

fileNameFromRunningSavePanelForSaveOperation

```
public String fileNameFromRunningSavePanelForSaveOperation(int saveOperation)
```

Discussion

Runs the modal Save panel and returns the filename (as a fully qualified path) selected for the receiver. *saveOperation* determines the title of the Save panel (Save, Save As, Save To). It also affects whether the Save panel includes an accessory view with a pop-up list containing the document's native or writable types. If *saveOperation* is `SaveOperation` or `SaveAsOperation`, the accessory pop-up list contains only those document types the application can read and write. If *saveOperation* is `SaveToOperation`, the pop-up list additionally includes the document types that the application can write (but can't read). If there is only one type the document can be written to, or if `shouldRunSavePanelWithAccessoryView` (page 548) returns `false`, the accessory view isn't shown. The default extension for saved documents is the first extension assigned for the document's native type or, if there is no native type, the extension for the first writable type specified in the `CFBundleDocumentTypes` property. File packages are treated as files.

Availability
Deprecated in Mac OS X v10.0.
Not supported in Mac OS X v10.4 and later.

See Also
`fileName` (page 527)
`runModalSavePanel` (page 541)

fileType

Returns the document type under which the receiver is saved.

```
public String fileType()
```

Discussion
When a document is saved, the type is determined by the file extension, as defined in the custom info dictionary (specified in `Info.plist`).

See Also
`setFileType` (page 545)

fileTypeFromLastRunSavePanel

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

```
public String fileTypeFromLastRunSavePanel()
```

Discussion

This type is primarily used by the `saveDocument` (page 542), `saveDocumentAs` (page 542), and `saveDocumentTo` (page 543) methods to determine the type the user chose after the Save panel has been run.

fileURL

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

```
public URL fileURL()
```

Discussion

The default implementation of this method returns whatever was stored by a previous invocation of the default implementation of `setFileURL` (page 545).For backward binary compatibility with Mac OS X v10.3 and earlier, if `fileName` (page 527) is overridden, the default implementation of this method instead invokes `[self fileName]` and returns the result as a URL.

Availability

Available in Mac OS X v10.4 and later.

See Also

`setFileURL` (page 545)

fileWrapperOfType

Creates and returns a file wrapper that contains the contents of the document.

```
public NSFileWrapper fileWrapperOfType(String typeName)
```

Discussion

The document is formatted to a specified type *`typeName`*. If unsuccessful, the default implementation of this method returns `null`.

Availability

Available in Mac OS X v10.4 and later.

See Also

`dataOfType` (page 525)

fileWrapperRepresentationOfType

```
public NSFileWrapper fileWrapperRepresentationOfType(String aType)
```

Discussion

Returns an NSFileWrapper object that represents the data of the receiver in a given type (*`aType`*). This method invokes dataRepresentationOfType to get the data object from which to create a plain-file file wrapper. Subclasses can override this method if dataRepresentationOfType is not adequate for their needs. This method is invoked by the default implementation of `writeToFile` (page 552).

Availability
Deprecated in Mac OS X v10.4.

See Also
`loadFileWrapperRepresentation` (page 533)

hasUnautosavedChanges

Return `true` if the document has changes that have not been autosaved, `false` otherwise, as determined by the history of previous invocations of `updateChangeCount` (page 549).

```
public boolean hasUnautosavedChanges()
```

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

hasUndoManager

Returns whether the receiver owns or should own an NSUndoManager.

```
public boolean hasUndoManager()
```

See Also
`setHasUndoManager` (page 546)

initForURLWithContentsOfURLOfType

Initializes a document located by a URL.

```
public boolean initForURLWithContentsOfURLOfType(URL absoluteDocumentURL, URL
    absoluteDocumentContentsURL, String typeName)
```

Discussion
Initializes the document located by *`absoluteDocumentURL`*, of the specified type *`typeName`*, but by reading the contents for the document from *`absoluteDocumentContentsURL`*, and returns it if successful. If not successful, returns `null`.

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

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

```
ChangeReadOtherContents
```

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

initWithContentsOfURLOfType

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

```
public boolean initWithContentsOfURLOfType(URL absoluteURL, String typeName)
```

Discussion

The URL is specified by *absoluteURL*, and the type is specified by *typeName*. Returns the document if successful. If not successful, returns `null`. You can override this method to customize the reopening of autosaved documents.

This method is invoked by the NSDocumentController method `makeDocumentWithContentsOfURLOfType` (page 568). The default implementation of this method invokes `readFromURLOfType` (page 538), `setFileURL` (page 545), `setFileType` (page 545), and `setFileModificationDate` (page 544).

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

Availability

Available in Mac OS X v10.4 and later.

initWithType

Initializes a document of a specified type, and returns it if successful.

```
public boolean initializeWithType(String typeName)
```

Discussion

The type is specified by *typeName*. If not successful, returns `null`.

You can override this method to perform initialization that must be done when creating new documents but should not be done when opening existing documents. Your override should typically invoke `super` to initialize NSDocument's private instance variables.

Availability

Available in Mac OS X v10.4 and later.

isDocumentEdited

Returns `true` if the receiver has been edited since it was last saved or if the document is new; otherwise, returns `false`.

```
public boolean isDocumentEdited()
```

Discussion

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

See Also

`updateChangeCount` (page 549)
`setDocumentEdited` (page 1859) (NSWindow)

keepBackupFile

Returns whether the receiver should keep the backup files created before document data is written to a file (false by default).

```
public boolean keepBackupFile()
```

Discussion

Override this method if you want different behavior.

See Also

writeToFile (page 552)

lastError

Returns the NSError object that was most recently set.

```
public NSError lastError()
```

Discussion

Errors are set by NSDocument Java member functions or overrides using the setLastError (page 546) member function. Typically, this member function is called only by the Application Kit itself.

Availability

Available in Mac OS X v10.4 and later.

See Also

setLastError (page 546)

loadDataRepresentation

This method has been deprecated.

```
public boolean loadDataRepresentation(NSData docData, String docType)
```

Discussion

Overridden by subclasses to load document data (*docData*) of type *docType* into the receiver, display it in windows, and return whether the operation was successful. This method is typically invoked by loadFileWrapperRepresentation (page 533) after an NSData object is created from the contents of the file wrapper (which can include directories). The default implementation throws an InternalInconsistencyException. Subclasses must override this method unless they override readFromFile (page 537) or loadFileWrapperRepresentation (page 533) to do specialized reading and loading of document data.

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

Availability

Deprecated in Mac OS X v10.4.

See Also

dataRepresentationOfType (page 525)

loadFileWrapperRepresentation

This method has been deprecated.

```
public boolean loadFileWrapperRepresentation(NSFileWrapper wrapper, String docType)
```

Discussion
Loads document data in file wrapper *wrapper* of type *docType* into the receiver, displays it in windows, and returns whether the operation was successful. If *wrapper* is a simple file, it invokes `loadDataRepresentation` (page 532) to load the data. If *wrapper* is a directory, it returns `false` by default; subclasses can override to handle file wrappers that are directories. This method is typically invoked by `readFromFile` (page 537) after it creates an NSData object from the contents of the file.

Availability
Deprecated in Mac OS X v10.4.

See Also
`fileWrapperRepresentationOfType` (page 529)

makeWindowControllers

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

```
public void makeWindowControllers()
```

Discussion
The base class implementation creates an NSWindowController with `windowNibName` (page 551) and with the document as the file's owner if `windowNibName` (page 551) returns a name. If you override this method to create your own window controllers, be sure to use `addWindowController` (page 523) to add them to the document after creating them.

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

See Also
`windowControllers` (page 550)

preparePageLayout

Invoked by `runModalPageLayoutWithPrintInfo` (page 540) and `runModalPageLayout` (page 540) to do any customization of the Page Layout panel *pageLayout*, such as adding an accessory view.

```
public boolean preparePageLayout(NSPageLayout pageLayout)
```

Discussion
Returns `true` if successfully prepared, and `false` otherwise. The default implementation is empty and returns `true`.

prepareSavePanel

Invoked by `runModalSavePanel` (page 541) to do any customization of the Save panel *savePanel*.

```
public boolean prepareSavePanel(NSSavePanel savePanel)
```

Discussion

Returns `true` if successfully prepared, and `false` otherwise. The default implementation is empty and returns `true`.

presentError

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

```
public boolean presentError(NSError error)
```

Discussion

Returns `true` if error recovery was done, `false` otherwise. This method does not return until the user dismisses the alert.

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

The default implementation of this method invokes `willPresentError` (page 549) to give subclassers an opportunity to customize error presentation. You should not override this method but should instead override `willPresentError` (page 549).

Availability

Available in Mac OS X v10.4 and later.

See Also

`willPresentError` (page 549)

`presentErrorModalForWindow` (page 534)

presentErrorModalForWindow

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

```
public void presentErrorModalForWindow(NSError error, NSWindow window, Object
    delegate, NSSelector didPresentSelector, Object contextInfo)
```

Discussion

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

```
public void didPresentErrorWithRecovery (boolean didRecover, Object  contextInfo)
```

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

The default implementation of this method invokes `willPresentError` (page 549) to give subclassers an opportunity to customize error presentation. You should not override this method but should instead override `willPresentError`.

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

See Also
presentError (page 534)
willPresentError (page 549)

printDocument

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

```
public void printDocument(Object sender)
```

Discussion
An NSDocument receives this action message as it travels up the responder chain. The default implementation invokes printShowingPrintPanel with an argument of `true`.

See Also
printInfo (page 536)
runPageLayout (page 541)
setPrintInfo (page 546)
shouldChangePrintInfo (page 547)

printDocumentWithSettings

Prints the document.

```
public void printDocumentWithSettings(NSDictionary printSettings, boolean
    showPrintPanel, Object delegate, NSSelector didPrintSelector, Object contextInfo)
```

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

```
public void documentDidPrint (NSDocument document, boolean didPrintSuccessfully,
  Object  contextInfo)
```

The default implementation of this method invokes printOperationWithSettings (page 536). If `null` is returned it presents the error to the user in a document-modal panel before messaging the delegate..

For backward binary compatibility with Mac OS X v10.3 and earlier, the default implementation of this method invokes printShowingPrintPanel (page 536) if it is overridden. When doing this it uses private functionality to arrange for the print settings to take effect (despite the fact that the override of `printShowingPrintPanel` can't possibly know about them) and to get notified when the print operation has been completed, so it can message the delegate at the correct time. Correct messaging of the delegate is necessary for correct handling of the Print Apple event.

Availability

Available in Mac OS X v10.4 and later.

See Also

`printOperationWithSettings` (page 536)

printInfo

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

```
public NSPrintInfo printInfo()
```

Discussion

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

See Also

`runPageLayout` (page 541)
`setPrintInfo` (page 546)
`shouldChangePrintInfo` (page 547)

printOperationWithSettings

Creates a print operation and returns it if successful.

```
public NSPrintOperation printOperationWithSettings(NSDictionary printSettings)
```

Discussion

If not successful, returns `null`. The print operation can be run to print the document's current contents. The NSPrintInfo attributes in the passed-in *`printSettings`* dictionary are added to a copy of the document's print info, and the resulting print info is used for the operation. The default implementation of this method does nothing. You must override it to enable printing in your application.

Availability

Available in Mac OS X v10.4 and later.

See Also

`printDocumentWithSettings` (page 535)

printShowingPrintPanel

This method has been deprecated.

```
public void printShowingPrintPanel(boolean flag)
```

Discussion

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

Availability
Deprecated in Mac OS X v10.4.

See Also
`printInfo` (page 536)

readFromDataOfType

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

```
public boolean readFromDataOfType(NSData data, String typeName)
```

Discussion
The type is specified by *typeName*. If not successful, the method returns `false`. The default implementation of this method throws an exception because at least one of the three reading methods (this method, `readFromURLOfType` (page 538), `readFromFileWrapperOfType` (page 537)), or every method that may invoke `readFromURLOfType` (page 538), must be overridden.

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

readFromFile

This method has been deprecated.

```
public boolean readFromFile(String fileName, String docType)
```

Discussion
Reads and loads document data of type *docType* from the file *fileName*, returning whether the operation was successful. This method invokes loadDataRepresentation and is invoked when the receiver is first created and initialized.

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

Availability
Deprecated in Mac OS X v10.4.

See Also
`dataRepresentationOfType` (page 525)
`writeToFile` (page 552)

readFromFileWrapperOfType

Set the contents of this document by reading from a file wrapper of a specified type, and return `true` if successful.

```
public boolean readFromFileWrapperOfType(NSFileWrapper fileWrapper, StringtypeName)
```

Discussion
If not successful, return `false`.

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

See Also
`readFromURLOfType` (page 538)
`readFromDataOfType` (page 537)

readFromURL

This method has been deprecated.

```
public boolean readFromURL(java.net.URL aURL, String docType)
```

Discussion
Reads and loads document data of type *docType* from the URL *aURL*, returning whether the operation was successful. This method only supports URLs of the `file:` scheme and calls `readFromFile` (page 537).

Availability
Deprecated in Mac OS X v10.4.

readFromURLOfType

Sets the contents of this document by reading from a file or file package located by a URL, of a specified type, and returns `true` if successful.

```
public boolean readFromURLOfType(URL absoluteURL, String typeName)
```

Discussion
The URL is represented by *absoluteURL*, and the document type by *typeName*. If not successful, returns `false`.

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

See Also
`readFromFileWrapperOfType` (page 537)
`readFromDataOfType` (page 537)

removeWindowController

Removes *windowController* from the receiver.

```
public void removeWindowController(NSWindowController windowController)
```

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

See Also

shouldCloseDocument (page 1894) (NSWindowController)

revertDocumentToSaved

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

```
public void revertDocumentToSaved(Object sender)
```

Discussion

The default implementation of this method presents an alert dialog giving the user the opportunity to cancel the operation. If the user chooses to continue, the method ensures that any editor registered using the Cocoa Bindings NSEditorRegistration informal protocol has discarded its changes and then invokes revertToContentsOfURLOfType (page 539). If that returns false, the method presents the error to the user in an document-modal alert dialog.

See Also

updateChangeCount (page 549)

revertToContentsOfURLOfType

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

```
public boolean revertToContentsOfURLOfType(URL absoluteURL, String typeName)
```

Discussion

The URL is represented by *absoluteURL*, and the document type by *typeName*. If not successful, returns `false`.

Availability

Available in Mac OS X v10.4 and later.

revertToSavedFromFile

This method has been deprecated.

```
public boolean revertToSavedFromFile(String fileName, String type)
```

Discussion

Reverts the receiver to the data stored in the file system in file named *fileName* of file type *type*. Invokes readFromFile (page 537) and returns whether that method successfully read the file and processed the document data.

Availability

Deprecated in Mac OS X v10.4.

See Also

revertDocumentToSaved (page 539)

revertToSavedFromURL

This method has been deprecated.

```
public boolean revertToSavedFromURL(java.net.URL aURL, String type)
```

Discussion

Reverts the receiver to the data stored at *aURL* of type *type*. Invokes `readFromURL` (page 538) and returns whether that method successfully read the file and processed the document data.

Availability

Deprecated in Mac OS X v10.4.

See Also

`revertDocumentToSaved` (page 539)

runModalPageLayout

Runs the modal page layout panel with the receiver's printing information object (*printInfo*).

```
public void runModalPageLayout(NSPrintInfo printInfo, Object delegate, NSSelector
     didRunSelector, Object contextInfo)
```

Discussion

Invoked from the action method `runPageLayout` (page 541). The *contextInfo* argument provides any additional context information. Presents the page layout panel application modally if there is no document window to which it can be presented document modally.

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

```
public void documentDidRunModalPageLayout (NSDocument document,  boolean accepted,
 void  contextInfo)
```

runModalPageLayoutWithPrintInfo

This method has been deprecated. Use `runModalPageLayout` (page 540) instead.

```
public int runModalPageLayoutWithPrintInfo(NSPrintInfo printInfo)
```

Discussion

Runs the page layout modal panel with the receiver's printing information object (*printInfo*) as argument and returns the result constant (indicating the button pressed by the user). To run as sheet on the receiver's document window, use `runModalPageLayout` (page 540) instead.

Availability

Deprecated in Mac OS X v10.4.

See Also

`shouldChangePrintInfo` (page 547)

`runModalWithPrintInfo` (page 1049) (NSPageLayout)

runModalPrintOperation

Runs a print operation *printOperation* modally.

```
public void runModalPrintOperation(NSPrintOperation printOperation, Object delegate,
     NSSelector didRunSelector, Object contextInfo)
```

Discussion

Overrides of printShowingPrintPanel (page 536) can invoke this method.

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

```
public void documentDidRunModalPrintOperation(NSDocument document,  boolean
success, void  contextInfo)
```

runModalSavePanel

Runs the modal Save panel *savePanel* with accessory view *accessoryView* and returns the result constant (indicating the button clicked by the user).

```
public int runModalSavePanel(NSSavePanel savePanel, NSView accessoryView)
```

Discussion

accessoryView is usually a pop-up list containing the receiver's native types and its supported writable types. Invoked by fileNameFromRunningSavePanelForSaveOperation.

This method is no longer supported.

Prepares and runs the modal Save panel.

```
public void runModalSavePanel(int saveOperation, Object delegate, NSSelector
    didSaveSelector, Object contextInfo)
```

Discussion

Invoked from saveDocument (page 542), and the action methods saveDocumentAs (page 542) and saveDocumentTo (page 543). Calls prepareSavePanel (page 533) to allow further customization of the Save panel. The *delegate* is assigned to the Save panel. Pass the *contextInfo* object with the callback.

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

```
public void documentDidSave (NSDocument doc, boolean didSave, Object  contextInfo)
```

Availability

Deprecated in Mac OS X v10.0.

Not supported in Mac OS X v10.4 and later.

See Also

shouldRunSavePanelWithAccessoryView (page 548)

runPageLayout

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

```
public void runPageLayout(Object sender)
```

Discussion

The default implementation invokes `runModalPageLayout` (page 540) with the document's current NSPrintInfo object as argument; if the user clicks the OK button and the document authorizes changes to its printing information (shouldChangePrintInfo), the method sets the document's new NSPrintInfo object and increments the document's change count.

See Also

`setPrintInfo` (page 546)

`updateChangeCount` (page 549)

saveDocument

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

```
public void saveDocument(Object sender)
```

Discussion

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

See Also

`fileNameFromRunningSavePanelForSaveOperation` (page 528)

`setFileName` (page 545)

`setFileType` (page 545)

`updateChangeCount` (page 549)

Runs the Save panel and invokes `saveToFile` (page 543) with the result.

```
public void saveDocument(Object delegate, NSSelector didSaveSelector, Object
    contextInfo)
```

Discussion

It is called from the `saveDocument` (page 542) action method and from `canCloseDocument` (page 524) if the user chooses to save. Pass the *contextInfo* object with the callback. The *delegate* is assigned to the Save panel.

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

```
public void documentDidSave (NSDocument doc, boolean didSave, Object  contextInfo)
```

saveDocumentAs

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

```
public void saveDocumentAs(Object sender)
```

Discussion

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

See Also

`fileNameFromRunningSavePanelForSaveOperation` (page 528)

`setFileName` (page 545)

`setFileType` (page 545)

`updateChangeCount` (page 549)

saveDocumentTo

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

```
public void saveDocumentTo(Object sender)
```

Discussion

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

See Also

`fileNameFromRunningSavePanelForSaveOperation` (page 528)

saveToFile

This method has been deprecated.

```
public void saveToFile(String fileName, int saveOperation, Object delegate,
    NSSelector didSaveSelector, Object contextInfo)
```

Discussion

Called after the user has been given the opportunity to select a destination through the modal Save panel presented by `runModalSavePanel` (page 541). The *delegate* is assigned to the Save panel. If *fileName* is non-`null`, this method writes the document to *fileName*, sets the document's file location and document type (if a native type), and clears the document's edited status. *didSaveSelector* gets called with `true` if the document is saved successfully, and `false` otherwise. The *saveOperation* is one of the constants in "Constants" (page 554). Pass *contextInfo* with the callback.

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

```
public void documentDidSave (NSDocument doc, boolean didSave, Object  contextInfo)
```

Availability

Deprecated in Mac OS X v10.4.

saveToURLOfType

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

```
public boolean saveToURLOfType(URL absoluteURL, String typeName, int saveOperation)
```

Discussion

The URL is represented by *absoluteURL*, the document type by *typeName*, and the save operation type by *saveOperation*. If not successful, returns `false`.

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

```
public void saveToURLOfType(URL absoluteURL, String typeName, int saveOperation,
    Object delegate, NSSelector didSaveSelector, Object contextInfo)
```

Discussion

The URL is represented by *absoluteURL*,, the document type by *typeName*, and the save operation type by *saveOperation*. When saving is completed, regardless of success or failure, the method sends the message selected by *didSaveSelector* to the *delegate*, with the *contextInfo* as the last argument. The method selected by *didSaveSelector* must have the same signature as:

```
public void documentDidSave (NSDocument document, boolean didSaveSuccessfully,
  Object  contextInfo)
```

Availability

Available in Mac OS X v10.4 and later.

setAutosavedContentsFileURL

Sets the location of the most recently autosaved document contents.

```
public void setAutosavedContentsFileURL(URL absoluteURL)
```

Discussion

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

Availability

Available in Mac OS X v10.4 and later.

See Also

`autosavedContentsFileURL` (page 523)

setFileModificationDate

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

```
public void setFileModificationDate(NSDate modificationDate)
```

Discussion

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

Availability

Available in Mac OS X v10.4 and later.

See Also

fileModificationDate (page 527)

setFileName

This method has been deprecated.

```
public void setFileName(String fileName)
```

Discussion

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

Availability

Deprecated in Mac OS X v10.4.

See Also

fileName (page 527)

setFileType

Sets the document type under which the file is saved to *docType*.

```
public void setFileType(String docType)
```

Discussion

The document type affects how the data is filtered when it is written to or read from a file.

See Also

fileType (page 528)

setFileURL

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

```
public void setFileURL(URL absoluteURL)
```

Discussion

This method doesn't actually rename the document; it's just for recording the document's location during initial opening or saving. The default implementation of this method just records the URL so that the default implementation of fileURL (page 529) can return it.

For backward binary compatibility with Mac OS X v10.3 and earlier, the default implementation of this method instead invokes [self setFileName:[absoluteURL path]] if setFileName (page 545) is overridden and the URL is null or uses the file: scheme.

Availability

Available in Mac OS X v10.4 and later.

See Also

`fileURL` (page 529)

setHasUndoManager

Sets whether the receiver has its own NSUndoManager.

```
public void setHasUndoManager(boolean flag)
```

Discussion

If *flag* is `false` and the receiver currently owns an NSUndoManager, the NSUndoManager is released after being removed as an observer of undo-related notifications.

See Also

`hasUndoManager` (page 530)

setLastError

Sets the NSError object that will be returned by the `lastError` member function.

```
public void setLastError(NSError inError)
```

Discussion

The *inError* parameter represents the NSError object to be set.

Whenever your application's override of an NSDocument member function detects failure and is going to return `null` or whatever signals failure for that particular function, it should first call this function and pass in an NSError object that encapsulates the reason for the failure. However, it needn't do so if it's going to signal failure because it called a member function that itself failed and is expected to have already called `setLastError`.

Availability

Available in Mac OS X v10.4 and later.

See Also

`lastError` (page 532)

setPrintInfo

Sets the receiver's NSPrintInfo object to *printInfo*; this object is used in laying out the document for printing.

```
public void setPrintInfo(NSPrintInfo printInfo)
```

See Also

`printInfo` (page 536)

setUndoManager

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

```
public void setUndoManager(NSUndoManager undoManager)
```

Discussion

If *undoManager* is `null`, it turns off the hasUndoManager flag. If *undoManager* is non-`null`, it adds the receiver as an observer of `NSUndoManager.DidUndoChangeNotification`, `NSUndoManager.DidRedoChangeNotification`, and `NSUndoManager.WillCloseUndoGroupNotification`.

See Also

`undoManager` (page 548)

setWindow

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

```
public void setWindow(NSWindow aWindow)
```

Discussion

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

shouldChangePrintInfo

Returns whether the receiver should allow changes to the default NSPrintInfo object *newPrintInfo* used in printing the document.

```
public boolean shouldChangePrintInfo(NSPrintInfo newPrintInfo)
```

Discussion

The default implementation returns `true`. Subclasses can override this method to return `false`. This method is invoked by the runPageLayout method, which sets a new NSPrintInfo for the document only if this method returns `true`.

shouldCloseWindowController

This method variant is no longer supported. Instead use the other variant of this method.

```
public boolean shouldCloseWindowController(NSWindowController windowController)
```

Discussion

If closing the *windowController* would cause the receiver to be closed, invokes `canCloseDocument` (page 524) to display a Save panel and give the user an opportunity to save the document. Returns the return value of `canCloseDocument`. Note that the receiver doesn't close until its window controller closes.

Availability

Deprecated in Mac OS X v10.0.

Not supported in Mac OS X v10.4 and later.

Invokes *shouldCloseSelector* with the result of canCloseDocument (page 524) if the *windowController* that is closing is the last one or is marked as causing the document to close.

```
public void shouldCloseWindowController(NSWindowController windowController, Object
     delegate, NSSelector shouldCloseSelector, Object contextInfo)
```

Discussion
Otherwise it invokes *shouldCloseSelector* with `true`. This method is called automatically by NSWindow for any window that has a window controller and a document associated with it. NSWindow calls this method prior to asking its *delegate* windowShouldClose (page 1880). Pass *contextInfo* with the callback.

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

```
public void documentShouldClose (NSDocument document, boolean shouldClose,
Object  contextInfo)
```

See Also
close (page 525)
shouldCloseDocument (page 1894) (NSWindowController)

shouldRunSavePanelWithAccessoryView

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

```
public boolean shouldRunSavePanelWithAccessoryView()
```

Discussion
Subclasses can override to return `false`, thus excluding the accessory view from the Save panel.

See Also
runModalSavePanel (page 541)

showWindows

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

```
public void showWindows()
```

undoManager

Returns the NSUndoManager used by the receiver or `null` if the receiver should not own one.

```
public NSUndoManager undoManager()
```

Discussion
If the undo manager doesn't exist and hasUndoManager returns `true`, it creates one and invokes setUndoManager with the NSUndoManager as argument.

updateChangeCount

Updates the receiver's change count according to *changeType*.

```
public void updateChangeCount(int changeType)
```

Discussion

The change count indicates the document's edited status; if the change count is 0, the document has no changes to save, and if the change count is greater than 0, the document has been edited and is unsaved. *changeType* is described in "Constants" (page 554). If you are implementing undo and redo in an application, you should increment the change count every time you create an undo group and decrement the change count when an undo or redo operation is performed.

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

validateMenuItem

Validates the Revert menu item and items selected from the Save panel's pop-up list of writable document types items.

```
public boolean validateMenuItem(NSMenuItem anItem)
```

Discussion

Returns `true` if *anItem* should be enabled, `false` otherwise. Returns `true` for Revert if the document has been edited and a file exists for the document. Returns `true` for an item representing a writable type if, during a Save or Save As operation, it is a native type for the document. Subclasses can override this method to perform additional validations.

willPresentError

Called when the receiver is about to present an error. Returns the error that should actually be presented.

```
public NSError willPresentError(NSError error)
```

Discussion

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

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

Availability

Available in Mac OS X v10.4 and later.

See Also

`presentError` (page 534)

`presentErrorModalForWindow` (page 534)

windowControllerDidLoadNib

Sent after *windowController* loads a nib file if the receiver is the nib file's owner.

```
public void windowControllerDidLoadNib(NSWindowController windowController)
```

Discussion

See the class description for NSWindowController (page 1887) for additional information about nib files and "File's Owner".

Typically an NSDocument subclass overrides `windowNibName` (page 551) or `makeWindowControllers` (page 533), but not both. If `windowNibName` is overridden, the default implementation of `makeWindowControllers` will load the named nib file, making the NSDocument the nib file's owner. In that case, you can override `windowControllerDidLoadNib` and do custom processing after the nib file is loaded.

The default implementation of this method does nothing.

See Also

`windowControllerWillLoadNib` (page 550)

`windowControllers` (page 550)

windowControllers

Returns the receiver's current window controllers.

```
public NSArray windowControllers()
```

Discussion

If there are no window controllers, returns an empty NSArray.

See Also

`makeWindowControllers` (page 533)

`windowControllerDidLoadNib` (page 550)

`windowControllerWillLoadNib` (page 550)

`windowNibName` (page 551)

windowControllerWillLoadNib

Sent before *windowController* loads a nib file if the receiver is the nib file's owner.

```
public void windowControllerWillLoadNib(NSWindowController windowController)
```

Discussion

See the class description for NSWindowController (page 1887) for additional information about nib files and "File's Owner".

Typically an NSDocument subclass overrides `windowNibName` (page 551) or `makeWindowControllers` (page 533), but not both. If `windowNibName` is overridden, the default implementation of `makeWindowControllers` will load the named nib file, making the NSDocument the nib file's owner. In that case, you can override `windowControllerWillLoadNib` and do custom processing before the nib file is loaded.

The default implementation of this method does nothing.

See Also

windowControllerDidLoadNib (page 550)

windowControllers (page 550)

windowForSheet

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

```
public NSWindow windowForSheet()
```

Discussion

May return null, in which case the sender should present an application-modal panel. NSDocument's implementation of this method returns the window of the first window controller, or NSApplication.sharedApplication(). mainWindow() if there are no window controllers or if the first window controller has no window.

windowNibName

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

```
public String windowNibName()
```

Discussion

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

The default implementation returns null.

See Also

windowControllers (page 550)

writableTypesForSaveOperation

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

```
public NSArray writableTypesForSaveOperation(int saveOperation)
```

Discussion

The save operation type is represented by *saveOperation*. For every kind of save operation except NSSaveToOperation, the returned array must only include types for which the the application can play the Editor role. For NSSaveToOperation the returned array may include types for which the application can only play the Viewer role, and other types that the application can merely export. The default implementation of this method returns [[self class] writableTypes] with, except during NSSaveToOperations, types for which isNativeType (page 522) returns false filtered out.

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

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

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

writeSafelyToURLOfType

Writes the contents of the document to a file or file package.

```
public boolean writeSafelyToURLOfType(URL absoluteURL, String typeName, int
    saveOperation)
```

Discussion
The file or file package is located by the URL *absoluteURL*, formatted to the specified type *typeName*, for a particular kind of save operation *saveOperation*. Returns `true` if successful. If not successful, returns `false`.

The default implementation of this method invokes `fileAttributesToWriteToURLOfType` (page 526) and writes the returned attributes, if any, to the file. It may copy some attributes from the old on-disk revision of the document at the same time, if applicable.

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

For `SaveOperation`, the default implementation of this method invokes `keepBackupFile` (page 532) to determine whether or not the old on-disk revision of the document, if there was one, should be preserved after being renamed.

For backward binary compatibility with Mac OS X v10.3 and earlier, the default implementation of this method instead invokes `writeWithBackupToFile` (page 554) if that method is is overridden and the URL uses the `file:` scheme. The save operation in this case is never `AutosaveOperation`; `SaveToOperation` is used instead.

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

See Also
`fileAttributesToWriteToURLOfType` (page 526)

writeToFile

Writes document data of type *docType* to the file *fileName*, returning whether the operation was successful.

```
public boolean writeToFile(String fileName, String docType)
```

Discussion
This method invokes `dataRepresentationOfType` (page 525) and is indirectly invoked whenever the document file is saved. It uses NSData's `writeToFile` method to write to the file.

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

See "NSDocument Saving Behavior" (page 514) for additional information about saving documents.

See Also

`loadDataRepresentation` (page 532)

`readFromFile` (page 537)

This method is called from `writeWithBackupToFile` (page 554) to actually write the file of type *docType* to *fullDocumentPath*.

```
public boolean writeToFile(String fullDocumentPath, String docType, String
    fullOriginalDocumentPath, int saveOperationType)
```

Discussion

fullOriginalDocumentPath is the path to the original file if there is one and `null` otherwise. The default implementation simply calls `writeToFile` (page 552) with no arguments. You should not need to call this method directly, but subclasses that need access to the previously saved copy of their document while saving the new one can override this method. The *saveOperationType* argument is one of the constants listed in "Constants" (page 554).

See "NSDocument Saving Behavior" (page 514) for additional information about saving documents.

writeToURL

```
public boolean writeToURL(java.net.URL aURL, String docType)
```

Discussion

Writes document data of type *docType* to the URL *aURL*, returning whether the operation was successful. This method only supports URLs of the `file:` scheme and calls `writeToFile` (page 552).

writeToURLOfType

Writes the contents of the file to a file or file package located by a URL, formatted to a specified type, and returns `true` if successful.

```
public boolean writeToURLOfType(URL absoluteURL, String typeName)
```

Discussion

The URL is represented by *absoluteURL* and the document type by *typeName*. If not successful, the method returns `false`.

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

```
public boolean writeToURLOfType(URL absoluteURL, String typeName, int saveOperation,
     URL absoluteOriginalContentsURL)
```

Discussion

The URL is represented by *absoluteURL*, the document type by *typeName*, and the save operation type by *saveOperation*. If not successful, returns false.

You can override this method instead of one of the three simple writing methods (readFromURLOfType (page 538), readFromFileWrapperOfType (page 537), readFromDataOfType (page 537)) if your document writing machinery needs access to the on-disk representation of the document revision that is about to be overwritten. The value of *absoluteURL* is often not the same as that returned by fileURL (page 529). Other times it is not the same as the URL for the final save destination. Likewise, *absoluteOriginalContentsURL* is often not the same value as that returned by - fileURL.

Availability

Available in Mac OS X v10.4 and later.

See Also

fileWrapperOfType (page 529)

dataOfType (page 525)

writeWithBackupToFile

This method has been deprecated.

```
public boolean writeWithBackupToFile(String fullDocumentPath, String docType, int
     saveOperationType)
```

Discussion

This method is called by action methods like saveDocument (page 542), saveDocumentAs (page 542), and saveDocumentTo (page 543). It is responsible for handling backup of the existing file, if any, and removal of that backup if keepBackupFile (page 532) returns false. In between those two things, it calls writeToFile (page 552) to write the document of type *docType* to *fullDocumentPath*. You should never need to call writeWithBackupToFile, but subclasses that want to change the way the backup works can override it. The *saveOperationType* argument is one of the constants listed in "Constants" (page 554).

Availability

Deprecated in Mac OS X v10.4.

Constants

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

Constant	Description
SaveOperation	Specifies a Save operation.
SaveAsOperation	Specifies a Save As operation.
SaveToOperation	Specifies a Save To operation.

Constant	Description
AutosaveOperation	Specifies an autosave operation, writing a document's contents to a file or file package separate from the document's current one. Available in Mac OS X v10.4 and later.

Change counts indicate a document's edit status. The following constants indicate how a document should operate on its change count and are used by updateChangeCount (page 549)

Constant	Description
ChangeDone	Increment change count.
ChangeUndone	Decrement change count.
ChangeCleared	Set change count to 0.
ChangeReadOtherContents	Document was initialized with contents of file or file package other than the one whose location would be returned by fileURL. Available in Mac OS X v10.4 and later.
ChangeAutosaved	Document's contents have been autosaved. Available in Mac OS X v10.4 and later.

NSDocumentController

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.application
Companion guide	Document-Based Applications Overview

Overview

An NSDocumentController object manages an application's documents. As the first-responder target of New and Open menu commands, it creates and opens documents and tracks them throughout a session of the application. When opening documents, an NSDocumentController runs and manages the modal Open panel. NSDocumentControllers also maintain and manage the mappings of document types, extensions, and NSDocument subclasses as specified in the `CFBundleDocumentTypes` property loaded from the information property list (`Info.plist`).

You can use various NSDocumentController methods to get a list of the current documents; get the current document (which is the document whose window is currently key); get documents based on a given filename or window; and find out about a document's extension, type, display name, and document class.

In some situations, it is worthwhile to subclass NSDocumentController in non-NSDocument-based applications to get some of its features. For example, NSDocumentController's management of the Open Recent menu is useful in applications that don't use subclasses of NSDocument.

Tasks

Constructors

`NSDocumentController` (page 561)
Creates a new NSDocumentController.

Obtaining the Shared Document Controller

`sharedDocumentController` (page 561)
Returns the shared NSDocumentController instance.

Creating and Opening Documents

`documentForURL` (page 565)
  Returns, for a given URL, the open document whose file or file package is located by the URL, or `null` if there is no such open document.

`openUntitledDocument` (page 571)
  Creates a new untitled document, presents its user interface if *`displayDocument`* is `true`, and returns the document if successful.

`makeUntitledDocumentOfType` (page 568)
  Instantiates a new untitled document of the specified type and returns it if successful.

`openDocumentWithContentsOfURL` (page 570)
  Opens a document located by a URL *`absoluteURL`*, presents its user interface if *`displayDocuments`* is `true`, and returns the document if successful.

`makeDocumentWithContentsOfURLOfType` (page 568)
  Instantiates a document located by a URL, of a specified type, and return it if successful.

`reopenDocumentForURLWithContentsOfURL` (page 573)
  Reopens an autosaved document located by a URL, by reading the contents for the document from another URL, presents its user interface, and returns `true` if successful.

`makeDocumentForURLWithContentsOfURLOfType` (page 567)
  Instantiates a document located by a URL, of a specified type, but by reading the contents for the document from another URL, and returns it if successful.

Handling Errors

`lastError` (page 567)
  Returns the NSError object that was most recently set.

`presentError` (page 572)
  Presents an error alert to the user as a modal panel.

`presentErrorModalForWindow` (page 572)
  Presents an error alert to the user as a modal panel.

`setLastError` (page 575)
  Sets the NSError object that will be returned by the `lastError` member function.

`willPresentError` (page 577)
  Called when the receiver is about to present an error, returns the error that should actually be presented.

Managing the Open Panel

`runModalOpenPanel` (page 574)
  Invokes NSOpenPanel's `runModalForTypes` (page 1024), passing the *`openPanel`* object and the file *`extensions`* associated with a document type.

`currentDirectory` (page 563)
  Returns the directory path to be used as the starting point in the Open panel.

`URLsFromRunningOpenPanel` (page 576)
  Creates an NSOpenPanel and initializes it appropriately.

Autosaving

`autosavingDelay` (page 562)
Returns the time interval in seconds for periodic autosaving.

`setAutosavingDelay` (page 574)
Sets the time interval in seconds for periodic autosaving.

Responding to Action Messages

`newDocument` (page 569)
An action method invoked by the New menu command, this method creates a new NSDocument object and adds it to the list of such objects managed by the receiver.

`openDocument` (page 570)

`saveAllDocuments` (page 574)
As the action method invoked by the Save All command, saves all open documents of the application that need to be saved.

Managing Documents

`documentClassNames` (page 564)
Returns the names of NSDocument subclasses supported by this application.

`documents` (page 566)
Returns the NSDocument objects managed by the receiver.

`addDocument` (page 561)
Adds *`document`* to the list of open documents.

`currentDocument` (page 563)
Returns the NSDocument object associated with the main window.

`documentClassForType` (page 564)
Returns the NSDocument subclass associated with document type *`documentTypeName`*.

`documentForWindow` (page 565)
Returns the NSDocument object whose window controller owns *`window`*.

`hasEditedDocuments` (page 566)
Returns whether the receiver has any documents with unsaved changes.

`removeDocument` (page 573)
Removes *`document`* from the list of open documents.

Managing the Open Recent Menu

`maximumRecentDocumentCount` (page 568)
Returns the maximum number of items that may be presented in the standard Open Recent menu.

`clearRecentDocuments` (page 562)
Empties the recent documents list for the application.

`noteNewRecentDocumentURL` (page 569)
This method should be called by applications not based on NSDocument when they open or save documents identified by *aURL*.

`noteNewRecentDocument` (page 569)
This method is called by NSDocuments at appropriate times for managing the recents list.

`recentDocumentURLs` (page 573)
Returns the list of recent document URLs.

Managing Document Types

`typeForContentsOfURL` (page 576)
Returns, for a specified URL, the name of the document type that should be used when opening the document at that location, if successful.

`defaultType` (page 563)
Returns the name of the document type that should be used when creating new documents.

`displayNameForType` (page 564)
Returns the descriptive name for the document type (*documentTypeName*), which is often part of the document's window title.

`fileExtensionsFromType` (page 566)
Returns the allowable file extensions (as String objects) for document type *documentTypeName*.

`typeFromFileExtension` (page 576)
Returns the document type associated with files having extension *fileExtensionOrHFSFileType*.

Validating User Interface Items

`validateMenuItem` (page 576)
Validates menu item *anItem*, returning `true` if it should be enabled, `false` otherwise.

Deprecated Methods

`closeAllDocuments` (page 562)
Attempts to close all documents owned by the receiver and returns whether all documents were closed.

`documentForFileName` (page 565)
This method has been deprecated.

`fileNamesFromRunningOpenPanel` (page 566)
This method has been deprecated.

`makeDocumentWithContentsOfURL` (page 568)

`makeDocumentWithContentsOfFile` (page 567)
This method has been deprecated.

`openDocumentWithContentsOfFile` (page 570)
This method has been deprecated.

`openUntitledDocumentOfType` (page 571)

`reviewUnsavedDocumentsWithAlertTitle` (page 573)
This method variant has been deprecated. Instead use the other variant of this method.

`setShouldCreateUI` (page 575)
This method has been deprecated.

`shouldCreateUI` (page 575)
This method has been deprecated.

Constructors

NSDocumentController

Creates a new NSDocumentController.

```
public NSDocumentController()
```

Discussion
The first instance of NSDocumentController or any of its subclasses becomes the shared instance.

Static Methods

sharedDocumentController

Returns the shared NSDocumentController instance.

```
public static NSDocumentController sharedDocumentController()
```

Discussion
If one doesn't exist yet, it is created. Initialization reads in the document types from the `CFBundleDocumentTypes` property list (in `Info.plist`), registers the instance for `WorkspaceWillPowerOffNotification` (page 1915)s, and turns on the flag indicating that document user interfaces should be visible. You should always obtain your application's NSDocumentController using this method.

See Also
`setShouldCreateUI` (page 575)

Instance Methods

addDocument

Adds *`document`* to the list of open documents.

```
public void addDocument(NSDocument document)
```

Discussion

The `open...` methods automatically call `addDocument`. This method is mostly provided for subclassers that want to know when documents arrive.

autosavingDelay

Returns the time interval in seconds for periodic autosaving.

```
public double autosavingDelay()
```

Discussion

A value of 0 indicates that periodic autosaving should not be done at all. NSDocumentController uses this number as the amount of time to wait between detecting that a document has unautosaved changes and sending the document an `autosaveDocument` (page 523) message. The default value is 0.

Availability

Available in Mac OS X v10.4 and later.

See Also

`setAutosavingDelay` (page 574)

clearRecentDocuments

Empties the recent documents list for the application.

```
public void clearRecentDocuments(Object sender)
```

Discussion

This is the action for the Clear menu command, but it can be invoked directly if necessary.

closeAllDocuments

Attempts to close all documents owned by the receiver and returns whether all documents were closed.

```
public boolean closeAllDocuments()
```

Discussion

It does not ask users whether they want to save documents. This method is invoked in `reviewUnsavedDocumentsWithAlertTitle` (page 573) when users decide to discard all changes.

Iterates through all the open documents and tries to close them one by one using *delegate*.

```
public void closeAllDocuments(Object delegate, NSSelector didCloseAllSelector,
    Object contextInfo)
```

Discussion

Each NSDocument is sent `canCloseDocument` (page 524), which, if the document is dirty, gives it a chance to refuse to close or to save itself first. This method may ask whether to save or to perform a save.

The *didCloseAllSelector* callback method is invoked with `true` if all documents are closed, and `false` otherwise. Pass *contextInfo* with the callback. The *didCloseAllSelector* callback method should have the following signature:

```
public void documentControllerDidCloseAll (NSDocumentController  docController,
 boolean  didCloseAll, Object contextInfo)
```

currentDirectory

Returns the directory path to be used as the starting point in the Open panel.

```
public String currentDirectory()
```

Discussion

The first valid directory from the following list is returned:

- The directory location where the current document was last saved
- The last directory visited in the Open panel
- The user's home directory

See Also

`documentForFileName` (page 565)

currentDocument

Returns the NSDocument object associated with the main window.

```
public NSDocument currentDocument()
```

Discussion

This method returns `null` if it is called when its application is not active. This can occur during processing of a drag-and-drop operation, for example, in an implementation of `readSelectionFromPasteboard:`. In such a case, send the following message instead from an NSView subclass associated with the document:

```
[[[self window] windowController] document];
```

See Also

`documentForFileName` (page 565)

`documentForWindow` (page 565)

`documents` (page 566)

defaultType

Returns the name of the document type that should be used when creating new documents.

```
public String defaultType()
```

Discussion

The default implementation of this method returns the first Editor type declared in the application's Info.plist, or returns `null` if no Editor type is declared. You can override it to customize the type of document that is created when, for instance, the user chooses New in the File menu.

Availability

Available in Mac OS X v10.4 and later.

displayNameForType

Returns the descriptive name for the document type (*documentTypeName*), which is often part of the document's window title.

```
public String displayNameForType(String documentTypeName)
```

Discussion

This returned value is associated with the `TypeName` key in the `CFBundleDocumentTypes` property list. If there is no such value, *documentTypeName* is returned.

See Also

`fileExtensionsFromType` (page 566)

`typeFromFileExtension` (page 576)

documentClassForType

Returns the NSDocument subclass associated with document type *documentTypeName*.

```
public Class documentClassForType(String documentTypeName)
```

Discussion

The document type must be one the document can read. If the class cannot be found, returns `null`.

See Also

`displayNameForType` (page 564)

`fileExtensionsFromType` (page 566)

`typeFromFileExtension` (page 576)

documentClassNames

Returns the names of NSDocument subclasses supported by this application.

```
public NSArray documentClassNames()
```

Discussion

The default implementation of this method returns information derived from the application's Info.plist. You can override it to return the names of document classes that are dynamically loaded from plugins.

Availability

Available in Mac OS X v10.4 and later.

documentForFileName

This method has been deprecated.

```
public NSDocument documentForFileName(String fileName)
```

Discussion

Returns the NSDocument object for the file in which the document data is stored. The `fileName` argument is a fully qualified path in the file system. Returns `null` if no document can be found.

Availability

Deprecated in Mac OS X v10.4.

See Also

`documentForWindow` (page 565)

`documents` (page 566)

documentForURL

Returns, for a given URL, the open document whose file or file package is located by the URL, or `null` if there is no such open document.

```
public NSDocument documentForURL(URL absoluteURL)
```

Discussion

The default implementation of this method queries each open document to find one whose URL matches, and returns the first one whose URL does match.

For backward binary compatibility with Mac OS X v10.3 and earlier, the default implementation of this method instead invokes `documentForFileName` (page 565) if it is overridden and the URL uses the `file:` scheme.

Availability

Available in Mac OS X v10.4 and later.

documentForWindow

Returns the NSDocument object whose window controller owns *window*.

```
public NSDocument documentForWindow(NSWindow window)
```

Discussion

Returns `null` if *window* is `null`, if *window* has no window controller, or if the window controller does not have an association with an NSDocument.

See Also

`currentDocument` (page 563)

`documentForFileName` (page 565)

`documents` (page 566)

documents

Returns the NSDocument objects managed by the receiver.

```
public NSArray documents()
```

Discussion

If there are currently no documents, returns an empty NSArray.

See Also

`currentDocument` (page 563)

`documentForFileName` (page 565)

`documentForWindow` (page 565)

fileExtensionsFromType

Returns the allowable file extensions (as String objects) for document type *documentTypeName*.

```
public NSArray fileExtensionsFromType(String documentTypeName)
```

Discussion

The first string in the returned NSArray is typically the most common extension. The array may also contain encoded HFS file types as will as filename extensions.

See Also

`displayNameForType` (page 564)

`typeFromFileExtension` (page 576)

fileNamesFromRunningOpenPanel

This method has been deprecated.

```
public NSArray fileNamesFromRunningOpenPanel()
```

Discussion

Returns a selection of files chosen by the user in the Open panel. Each file in the returned NSArray is a fully qualified path to the file's location in the file system. This method is invoked by `openDocument` (page 570), and it invokes `runModalOpenPanel` (page 574) after initializing the Open panel (which includes getting the starting directory with `currentDirectory` (page 563)). Returns `null` if the user cancels the Open panel or makes no selection.

Availability

Deprecated in Mac OS X v10.4.

hasEditedDocuments

Returns whether the receiver has any documents with unsaved changes.

```
public boolean hasEditedDocuments()
```

See Also

`documents` (page 566)

lastError

Returns the NSError object that was most recently set.

```
public NSError lastError()
```

Discussion
Errors are set by NSDocument Java member functions or overrides using the `setLastError` (page 575) member function. Typically, this member function is called only by the Application Kit itself.

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

See Also
`setLastError` (page 575)

makeDocumentForURLWithContentsOfURLOfType

Instantiates a document located by a URL, of a specified type, but by reading the contents for the document from another URL, and returns it if successful.

```
public NSDocument makeDocumentForURLWithContentsOfURLOfType(URL absoluteDocumentURL,
      URL absoluteDocumentContentsURL, String typeName)
```

Discussion
The URL is specified by *absoluteDocumentURL*, the type by *typeName*, and the other URL providing the contents by *absoluteDocumentContentsURL*. If not successful, the method returns `null`. The default implementation of this method invokes `documentClassForType` (page 564) to find out the class of document to instantiate, allocates a document object, and initializes it by sending it an `initForURLWithContentsOfURLOfType` (page 530) message.

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

makeDocumentWithContentsOfFile

This method has been deprecated.

```
public NSDocument makeDocumentWithContentsOfFile(String fileName, String docType)
```

Discussion
Creates and returns an NSDocument object for document type *docType* from the contents of the file *fileName*, which must be a fully qualified path. Returns `null` if the NSDocument subclass for *docType* couldn't be determined or if the object couldn't be created. This method is invoked by `openDocumentWithContentsOfFile` (page 570).

Availability
Deprecated in Mac OS X v10.4.

See Also
`openDocument` (page 570)

makeDocumentWithContentsOfURL

```
public NSDocument makeDocumentWithContentsOfURL(java.net.URL aURL, String docType)
```

Discussion

Creates and returns an NSDocument object for document type *docType* from the contents of *aURL*. Returns `null` if the NSDocument subclass for *docType* couldn't be determined or if the object couldn't be created. This method is invoked by `openDocumentWithContentsOfURL` (page 570).

See Also

`makeUntitledDocumentOfType` (page 568)

`openDocument` (page 570)

makeDocumentWithContentsOfURLOfType

Instantiates a document located by a URL, of a specified type, and return it if successful.

```
public NSDocument makeDocumentWithContentsOfURLOfType(URL absoluteURL, String
    typeName)
```

Discussion

The URL is specifed by *absoluteURL* and the document type by *typeName*. If not successful, the method returns `null`. The default implementation of this method invokes `documentClassForType` (page 564) to find out the class of document to instantiate, allocates a document object, and initializes it by sending it an `initWithContentsOfURLOfType` (page 531) message.

For backward binary compatibility with Mac OS X v10.3 and earlier, the default implementation of this method instead invokes `makeDocumentWithContentsOfFile` (page 567) if it is overridden and the URL uses the `file:` scheme.

Availability

Available in Mac OS X v10.4 and later.

makeUntitledDocumentOfType

Instantiates a new untitled document of the specified type and returns it if successful.

```
public NSDocument makeUntitledDocumentOfType(String typeName)
```

Discussion

The document type is specified by *typeName*. If not successful, the method returns `null`. The default implementation of this method invokes `documentClassForType` (page 564) to find out the class of document to instantiate, then allocates and initializes a document by sending it `initWithType` (page 531).

Availability

Available in Mac OS X v10.4 and later.

maximumRecentDocumentCount

Returns the maximum number of items that may be presented in the standard Open Recent menu.

```
public int maximumRecentDocumentCount()
```

Discussion

A value of 0 indicates that NSDocumentController will not attempt to add an Open Recent menu to your application's File menu, although NSDocumentController will not attempt to remove any preexisting Open Recent menu item. The default implementation returns a value that is subject to change and may or may not be derived from a setting made by the user in System Preferences.

Availability

Available in Mac OS X v10.4 and later.

newDocument

An action method invoked by the New menu command, this method creates a new NSDocument object and adds it to the list of such objects managed by the receiver.

```
public void newDocument(Object sender)
```

Discussion

It invokes `openUntitledDocumentOfType` (page 571) with the document type (first argument) being the first one specified in the `CFBundleDocumentTypes` property (defined in `Info.plist`); the document type determines the NSDocument subclass used to instantiate the document object.

See Also

`openDocument` (page 570)

noteNewRecentDocument

This method is called by NSDocuments at appropriate times for managing the recents list.

```
public void noteNewRecentDocument(NSDocument aDocument)
```

Discussion

This method constructs a URL and calls `noteNewRecentDocumentURL` (page 569). Subclasses might override this method to prevent certain documents or kinds of documents from getting into the list.

noteNewRecentDocumentURL

This method should be called by applications not based on NSDocument when they open or save documents identified by *aURL*.

```
public void noteNewRecentDocumentURL(java.net.URL aURL)
```

Discussion

NSDocument automatically calls this method when appropriate for NSDocument-based applications. Applications not based on NSDocument must also implement the `applicationOpenFile` (page 133) method in the application delegate to handle requests from the Open Recent menu command. You can override this method in an NSDocument-based application to prevent certain kinds of documents from getting into the list (but you have to identify them by URL).

openDocument

```
public void openDocument(Object sender)
```

Discussion

An action method invoked by the Open menu command, it runs the modal Open panel and, based on the selected filenames, creates one or more NSDocument objects from the contents of the files; it adds these objects to the list of NSDocument objects managed by the receiver. This method invokes `openDocumentWithContentsOfFile` (page 570), which actually creates the NSDocument objects.

See Also

`fileNamesFromRunningOpenPanel` (page 566)

`newDocument` (page 569)

openDocumentWithContentsOfFile

This method has been deprecated.

```
public NSDocument openDocumentWithContentsOfFile(String fileName, boolean flag)
```

Discussion

Returns an NSDocument object created from the contents of the file *fileName* (an absolute path) and displays it if *flag* is `true`. The returned object is added to the receiver's list of managed documents. Returns `null` if the object could not be created, typically because *fileName* does not point to a valid file or because there is no NSDocument subclass for the document type (as indicated by the file extension or HFS file type). Even if *flag* is `true`, the document is not displayed if `shouldCreateUI` (page 575) returns `false`. This method invokes `makeDocumentWithContentsOfFile` (page 567) to obtain the created NSDocument object. If you override this method, your implementation should be prepared to handle either `true` or `false`.

To handle an Open Documents Apple event, the Application Kit's built-in Apple event handling automatically invokes this method with the path to the file to open and a display argument.

Invoked with a display argument of `true` instead of `false` when a Print Documents Apple event is handled. This may have been handled differently in versions of Mac OS X prior to version 10.3.

Availability

Deprecated in Mac OS X v10.4.

See Also

`openDocument` (page 570)

`openUntitledDocumentOfType` (page 571)

`setShouldCreateUI` (page 575)

openDocumentWithContentsOfURL

Opens a document located by a URL *absoluteURL*, presents its user interface if *displayDocuments* is `true`, and returns the document if successful.

```
public NSDocument openDocumentWithContentsOfURL(URL absoluteURL, boolean
    displayDocument)
```

Discussion

If not successful, the method returns `null`.

The default implementation of this method checks to see if the document is already open according to `documentForURL` (page 565), and if it is not open determines the type of the document, invokes `makeDocumentWithContentsOfURLOfType` (page 568) to instantiate it, then invokes `addDocument` (page 561) to record its opening, and sends the document `makeWindowControllers` (page 533) and `showWindows` (page 548) messages if *`displayDocument`* is `true`. If the document is already open it is just sent a `showWindows` (page 548) message if *`displayDocument`* is `true`.

For backward binary compatibility with Mac OS X v10.3 and earlier, the default implementation of this method instead invokes `openDocumentWithContentsOfFile` (page 570), if it is overridden and the URL uses the `file:` scheme.

openUntitledDocument

Creates a new untitled document, presents its user interface if *`displayDocument`* is `true`, and returns the document if successful.

```
public NSDocument openUntitledDocument(boolean displayDocument)
```

Discussion

If not successful, the method returns `null`.

The default implementation of this method invokes `defaultType` (page 563) to determine the type of new document to create, invokes `makeUntitledDocumentOfType` (page 568) to create it, then invokes `addDocument` (page 561) to record its opening. If *`displayDocument`* is `true`, it then sends the new document `makeWindowControllers` (page 533) and `showWindows` (page 548) messages.

For backward binary compatibility with Mac OS X v10.3 and earlier, the default implementation of this method instead invokes `openUntitledDocumentOfType` (page 571) if it is overridden.

Availability

Available in Mac OS X v10.4 and later.

openUntitledDocumentOfType

```
public NSDocument openUntitledDocumentOfType(String docType, boolean display)
```

Discussion

Returns an NSDocument object instantiated from the NSDocument subclass required by document type *`docType`* and displays it if *`flag`* is `true`. The returned object is added to the receiver's list of managed documents. Returns `null` if the object could not be created, typically because no NSDocument subclass could be found for *`docType`*. Even if *`flag`* is `true`, the document is not displayed if `shouldCreateUI` (page 575) returns `false`.

See Also

`newDocument` (page 569)

`openDocumentWithContentsOfFile` (page 570)

`setShouldCreateUI` (page 575)

presentError

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

```
public boolean presentError(NSError error)
```

Discussion

Returns `true` if error recovery was done, `false` otherwise. This method does not return until the user dismisses the alert.

NSDocumentController's default implementation of this method is equivalent to that of NSResponder while treating the application object as the next responder and forwarding error presentation messages to it. (NSDocument’s default implementation of this method treats the shared NSDocumentController as the next responder and forwards these messages to it.) The default implementations of several NSDocumentController methods invoke this method.

The default implementation of this method invokes `willPresentError` (page 577) to give subclassers an opportunity to customize error presentation. You should not override this method but should instead override `willPresentError` (page 577).

Availability

Available in Mac OS X v10.4 and later.

See Also

`willPresentError` (page 577)

`presentErrorModalForWindow` (page 572)

presentErrorModalForWindow

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

```
public void presentErrorModalForWindow(NSError error, NSWindow window, Object
    delegate, NSSelector didPresentSelector, Object contextInfo)
```

Discussion

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

```
public void didPresentErrorWithRecovery (boolean didRecover, Object  contextInfo)
```

NSDocumentController's default implementation of this method is equivalent to that of NSResponder while treating the application object as the next responder and forwarding error presentation messages to it. (NSDocument’s default implementation of this method treats the shared NSDocumentController as the next responder and forwards these messages to it.)

The default implementation of this method invokes `willPresentError` (page 577) to give subclassers an opportunity to customize error presentation. You should not override this method but should instead override `willPresentError` (page 577).

Availability

Available in Mac OS X v10.4 and later.

See Also

`willPresentError` (page 577)

`presentError` (page 572)

recentDocumentURLs

Returns the list of recent document URLs.

```
public NSArray recentDocumentURLs()
```

Discussion
This method is not a good one to override since the internals of NSDocumentController do not generally use it.

removeDocument

Removes *document* from the list of open documents.

```
public void removeDocument(NSDocument document)
```

Discussion
A document will automatically call `removeDocument` (page 573) when it closes. This method is mostly provided for subclassers that want to know when documents close.

reopenDocumentForURLWithContentsOfURL

Reopens an autosaved document located by a URL, by reading the contents for the document from another URL, presents its user interface, and returns `true` if successful.

```
public boolean reopenDocumentForURLWithContentsOfURL(URL absoluteDocumentURL, URL
     absoluteDocumentContentsURL)
```

Discussion
The document is located by *absoluteDocumentURL* and the contents are read from *absoluteDocumentContentsURL*. If not successful, the method returns `false`.

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

reviewUnsavedDocumentsWithAlertTitle

This method variant has been deprecated. Instead use the other variant of this method.

```
public boolean reviewUnsavedDocumentsWithAlertTitle(String title, boolean flag)
```

Discussion
Displays an alert dialog asking users if they want to review unsaved documents, quit regardless of unsaved documents, or (if *flag* is `true`) cancel the impending save-and-terminate operation. Returns `true` if the application is to quit and `false` if otherwise (used only when the application is terminating). If the user selects the Review Unsaved option, `closeAllDocuments` (page 562) is invoked. This method is invoked when users choose the Quit menu command and when the computer power is being turned off (in which case, *flag* is `false`).

Displays an alert dialog asking if the user wants to review unsaved documents, quit regardless of unsaved documents, or (if *cancellable* is true) cancel the impending save operation.

```
public void reviewUnsavedDocumentsWithAlertTitle(String title, boolean cancellable,
     Object delegate, NSSelector didReviewAllSelector, Object contextInfo)
```

Discussion

Invokes *didReviewAllSelector* with true if quit without saving is chosen or if there are no dirty documents, and false otherwise. Assigns *delegate* to the panel. If the user selects the “Review Unsaved” option, closeAllDocuments (page 562) is invoked. This method is invoked when the user chooses the Quit menu command and also when the computer power is being turned off (in which case, *cancellable* is false). Note that *title* is ignored. Pass the *contextInfo* object with the callback.

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

```
public void documentControllerDidReviewAll (NSDocumentController  docController,
 boolean  didReviewAll, Object contextInfo)
```

runModalOpenPanel

Invokes NSOpenPanel’s runModalForTypes (page 1024), passing the *openPanel* object and the file *extensions* associated with a document type.

```
public int runModalOpenPanel(NSOpenPanel openPanel, NSArray extensions)
```

Discussion

This method is invoked by the fileNamesFromRunningOpenPanel (page 566) method. *extensions* may also contain encoded HFS file types as well as filename extensions.

saveAllDocuments

As the action method invoked by the Save All command, saves all open documents of the application that need to be saved.

```
public void saveAllDocuments(Object sender)
```

See Also

saveDocument (page 542) (NSDocument)

setAutosavingDelay

Sets the time interval in seconds for periodic autosaving.

```
public void setAutosavingDelay(double autosavingDelay)
```

Discussion

A value of 0 indicates that periodic autosaving should not be done at all. NSDocumentController uses this number as the amount of time to wait between detecting that a document has unautosaved changes and sending the document an autosaveDocument (page 523) message. The default value is 0.

Availability

Available in Mac OS X v10.4 and later.

See Also

`autosavingDelay` (page 562)

setLastError

Sets the NSError object that will be returned by the `lastError` member function.

```
public void setLastError(NSError inError)
```

Discussion

The *inError* parameter represents the NSError object to be set.

Whenever your application's override of an NSDocumentController member function detects failure and is going to return `null` or whatever signals failure for that particular function, it should first call this function and pass in an NSError object that encapsulates the reason for the failure. However, it needn't do so if it's going to signal failure because it called a member function that itself failed and is expected to have already called `setLastError`.

Availability

Available in Mac OS X v10.4 and later.

See Also

`lastError` (page 567)

setShouldCreateUI

This method has been deprecated.

```
public void setShouldCreateUI(boolean flag)
```

Discussion

Sets whether the window controllers (NSWindowControllers) of a document should be created when the document is created. When a window controller is created, it loads the nib file containing the window it manages. Often *flag* is set to `false` for scripting or searching operations involving the document's data.

Availability

Deprecated in Mac OS X v10.4.

See Also

`shouldCreateUI` (page 575)

shouldCreateUI

This method has been deprecated.

```
public boolean shouldCreateUI()
```

Discussion

Returns whether the window controllers (NSWindowControllers) of a document should be created when the document is created.

Availability

Deprecated in Mac OS X v10.4.

See Also

`setShouldCreateUI` (page 575)

typeForContentsOfURL

Returns, for a specified URL, the name of the document type that should be used when opening the document at that location, if successful.

```
public String typeForContentsOfURL(URL absoluteURL)
```

Discussion

The URL is represented by *absoluteURL*. If not successful, the method returns `null`.

You can override this method to customize type determination for documents being opened.

Availability

Available in Mac OS X v10.4 and later.

typeFromFileExtension

Returns the document type associated with files having extension *fileExtensionOrHFSFileType*.

```
public String typeFromFileExtension(String fileExtensionOrHFSFileType)
```

Discussion

fileExtensionOrHFSFileType may also be an encoded HFS file type, as well as a filename extension.

See Also

`displayNameForType` (page 564)

`fileExtensionsFromType` (page 566)

URLsFromRunningOpenPanel

Creates an NSOpenPanel and initializes it appropriately.

```
public NSArray URLsFromRunningOpenPanel()
```

Discussion

Then uses `runModalOpenPanel` (page 574) to run the NSOpenPanel. Returns the chosen files as an array of URLs. Returns `null` if the user cancels the Open panel or makes no selection.

validateMenuItem

Validates menu item *anItem*, returning `true` if it should be enabled, `false` otherwise.

```
public boolean validateMenuItem(NSMenuItem anItem)
```

Discussion

As implemented, if *anItem* is the Save All menu item, returns `true` if there are any edited documents. Subclasses can override this method to perform additional validations. Subclasses should call `super` in their implementation for items they don't handle themselves.

willPresentError

Called when the receiver is about to present an error, returns the error that should actually be presented.

```
public NSError willPresentError(NSError error)
```

Discussion

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

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

Availability

Available in Mac OS X v10.4 and later.

See Also

`presentError` (page 572)

`presentErrorModalForWindow` (page 572)

NSDPSContext

Inherits from	NSGraphicsContext : NSObject
Package:	com.apple.cocoa.application
Companion guide	Basic Drawing

Overview

NSDPSContext provides the context for the Display PostScript graphics environment. Display PostScript is no longer supported. Use NSGraphicsContext (page 729) instead.

Tasks

Constructors

`NSDPSContext` (page 579)
Creates an empty instance of NSDPSContext.

Getting Current Context

`currentContext` (page 580)
Returns the current context of the current thread.

Constructors

NSDPSContext

Creates an empty instance of NSDPSContext.

```
public NSDPSContext()
```

Static Methods

currentContext

Returns the current context of the current thread.

```
public static NSGraphicsContext currentContext()
```

Constants

The following constant is defined by NSDPSContext:

Constant	Description
`DPSRunLoopMode`	A special run-loop mode for processing DPS events

NSDragDestination

Inherits from	NSObject
Implements	NSDraggingInfo
Package:	com.apple.cocoa.application
Companion guide	Drag and Drop Programming Topics for Cocoa

Overview

NSDragDestination implements the functionality of the NSDraggingInfo (page 1959) interface. Instances are created by the Application Kit during drag operations and passed to the NSDraggingDestination methods.

Interfaces Implemented

NSDraggingInfo

- `draggedImage` (page 1960)
- `draggedImageLocation` (page 1960)
- `draggingDestinationWindow` (page 1960)
- `draggingLocation` (page 1961)
- `draggingPasteboard` (page 1961)
- `draggingSequenceNumber` (page 1961)
- `draggingSource` (page 1961)
- `draggingSourceOperationMask` (page 1961)
- `slideDraggedImageTo` (page 1962)

Tasks

Constructors

`NSDragDestination` (page 582)

Creates an empty NSDragDestination.

Dragging Session Information

`draggingSource` (page 584)
: Returns the source, or owner, of the dragged data or `null` if the source isn't in the same application as the destination.

`draggingSourceOperationMask` (page 584)
: Returns the dragging operation mask declared by the dragging source (through its `draggingSourceOperationMaskForLocal` (page 1966) method).

`draggingDestinationWindow` (page 583)
: Returns the destination window for the dragging operation.

`draggingPasteboard` (page 584)
: Returns the pasteboard object that holds the data being dragged.

`draggingSequenceNumber` (page 584)
: Returns a number that uniquely identifies the dragging session.

`draggingLocation` (page 583)
: Returns the current location of the mouse pointer in the base coordinate system of the destination object's window.

`namesOfPromisedFilesDroppedAtDestination` (page 585)
: Sets the drop location for promised files to *`dropDestination`* and returns the names (not full paths) of the files that the receiver promises to create there.

Image Information

`draggedImage` (page 583)
: Returns the image being dragged.

`draggedImageLocation` (page 583)
: Returns the current location of the dragged image's origin in the base coordinate system of the destination object's window.

Sliding the Image

`slideDraggedImageTo` (page 585)
: Slides the image to *`aPoint`*, a specified location in the screen coordinate system.

Constructors

NSDragDestination

Creates an empty NSDragDestination.

```
public NSDragDestination()
```

Instance Methods

draggedImage

Returns the image being dragged.

```
public NSImage draggedImage()
```

Discussion

This image object visually represents the data put on the pasteboard during the drag operation; however, it is the pasteboard data and not this image that is ultimately utilized in the dragging operation.

See Also

`draggedImageLocation` (page 583)

draggedImageLocation

Returns the current location of the dragged image's origin in the base coordinate system of the destination object's window.

```
public NSPoint draggedImageLocation()
```

Discussion

The image moves along with the mouse pointer (the position of which is given by `draggingLocation` (page 583)) but may be positioned at some offset.

See Also

`draggedImage` (page 583)

draggingDestinationWindow

Returns the destination window for the dragging operation.

```
public NSWindow draggingDestinationWindow()
```

Discussion

Either this window is the destination itself, or it contains the view object that is the destination.

draggingLocation

Returns the current location of the mouse pointer in the base coordinate system of the destination object's window.

```
public NSPoint draggingLocation()
```

See Also

`draggedImageLocation` (page 583)

draggingPasteboard

Returns the pasteboard object that holds the data being dragged.

```
public NSPasteboard draggingPasteboard()
```

Discussion

The dragging operation that is ultimately performed utilizes this pasteboard data and not the image returned by the draggedImage (page 583) method.

draggingSequenceNumber

Returns a number that uniquely identifies the dragging session.

```
public int draggingSequenceNumber()
```

draggingSource

Returns the source, or owner, of the dragged data or null if the source isn't in the same application as the destination.

```
public Object draggingSource()
```

Discussion

The dragging source implements methods from the NSDraggingSource (page 1965) interface.

draggingSourceOperationMask

Returns the dragging operation mask declared by the dragging source (through its draggingSourceOperationMaskForLocal (page 1966) method).

```
public int draggingSourceOperationMask()
```

Discussion

If the source permits dragging operations, the elements in the mask will be one or more of the constants described in NSDraggingInfo's "Constants" (page 1962), combined using the C bitwise OR operator.

If the source does not permit any dragging operations, this method should return NSDraggingInfo.DragOperationNone.

If the user is holding down a modifier key during the dragging session and the source doesn't prohibit modifier keys from affecting the drag operation (through its ignoreModifierKeysWhileDragging method), then the operating system combines the dragging operation value that corresponds to the modifier key (see the descriptions below) with the source's mask using the C bitwise AND operator.

The modifier keys are associated with the dragging operation options shown below:

Modifier Key	Dragging Operation
Control	NSDraggingInfo.DragOperationLink
Option	NSDraggingInfo.DragOperationCopy

Modifier Key	Dragging Operation
Command	`NSDraggingInfo.DragOperationGeneric`

namesOfPromisedFilesDroppedAtDestination

Sets the drop location for promised files to *dropDestination* and returns the names (not full paths) of the files that the receiver promises to create there.

```
public abstract NSArray namesOfPromisedFilesDroppedAtDestination(java.net.URL
    dropDestination)
```

Discussion

Drag destinations should invoke this method within their `performDragOperation` (page 1957) method. The source may or may not have created the files by the time this method returns.

slideDraggedImageTo

Slides the image to *aPoint*, a specified location in the screen coordinate system.

```
public void slideDraggedImageTo(NSPoint aPoint)
```

Discussion

This method can be used to snap the image down to a particular location. It should be invoked only from within the destination's implementation of `prepareForDragOperation` (page 1958)—in other words, after the user has released the image but before it's removed from the screen.

NSDrawer

Inherits from	NSResponder : NSObject
Implements	NSCoding (NSResponder)
Package:	com.apple.cocoa.application
Companion guide	Drawers

Overview

NSDrawer is a user interface element that contains and displays view objects including NSTextView, NSScrollView, NSBrowserView, and other classes that inherit from NSView. A drawer is associated with a window, called its parent, and can appear only while its parent is visible onscreen. A drawer cannot be moved or ordered independently of a window, but is instead attached to one edge of its parent and moves along with it.

Tasks

Constructors

`NSDrawer` (page 589)

Opening and Closing Drawers

`close` (page 590)
If the receiver is open, this method closes it.

`open` (page 592)
If the receiver is closed, this method opens it.

`openOnEdge` (page 592)
Causes the receiver to open on the specified *edge*.

`toggle` (page 595)
If the receiver is closed, or in the process of either opening or closing, it is opened.

Managing Drawer Size

`contentSize` (page 590)
Returns the size of the receiver's content area.

`leadingOffset` (page 591)
Returns the receiver's leading offset.

`maxContentSize` (page 591)
Returns the maximum allowed size of the content area.

`minContentSize` (page 591)
Returns the minimum allowed size of the receiver's content area.

`setContentSize` (page 592)
Sets the size of the receiver's content area to *size*.

`setLeadingOffset` (page 593)
Sets the receiver's leading offset to *offset*.

`setMaxContentSize` (page 593)
Specifies the maximum *size* of the receiver's content area. See "Positioning and Sizing a Drawer" for additional detail.

`setMinContentSize` (page 594)
Specifies the minimum *size* of the receiver's content area. See "Positioning and Sizing a Drawer" for additional detail.

`setTrailingOffset` (page 594)
Sets the receiver's trailing offset to *offset*.

`trailingOffset` (page 595)
Returns the receiver's trailing offset.

Managing Drawer Edges

`edge` (page 591)
Returns the edge of the window that the receiver is connected to.

`preferredEdge` (page 592)
When the receiver is told to open and an edge is not specified at that time, it opens on this value.

`setPreferredEdge` (page 594)

Managing a Drawer's Views

`contentView` (page 590)
Returns the receiver's content view.

`parentWindow` (page 592)
Returns the receiver's parent window

`setContentView` (page 593)
Rather than connect a drawer to its content view in Interface Builder, you can specify it programatically with the *aView* argument of this method.

`setParentWindow` (page 594)
    Sets the receiver's parent window to *parent*.

Accessing Other Drawer Information

`delegate` (page 591)
    Returns the receiver's delegate.

`setDelegate` (page 593)

`state` (page 595)
    Returns the state of the receiver.

Opening a drawer

`drawerDidOpen` (page 596) *delegate method*

`drawerShouldOpen` (page 596) *delegate method*

`drawerWillOpen` (page 597) *delegate method*

Resizing a drawer

`drawerWillResizeContents` (page 597) *delegate method*

Closing a drawer

`drawerDidClose` (page 596) *delegate method*

`drawerShouldClose` (page 596) *delegate method*

`drawerWillClose` (page 596) *delegate method*

Constructors

NSDrawer

```
public NSDrawer()
```

Discussion
Creates a new NSDrawer object. You must specify the parent window, content view, and edge of the drawer before using it. If you create a drawer in Interface Builder, you don't need to invoke the constructor programmatically.

```
public NSDrawer(NSSize contentSize, int edge)
```

Discussion
Creates a new NSDrawer object with size specified by *contentSize* and the edge to attach to specified by *edge*. You must specify the parent window and content view of the drawer using the methods included. When you create a drawer in Interface Builder, this constructor is invoked. The NSDrawer Inspector in Interface Builder allows you to set *edge*, and you can specific *contentSize* by changing the content view in Interface Builder.

See "Positioning and Sizing a Drawer" for additional detail on content size and drawer positioning.

Instance Methods

close

If the receiver is open, this method closes it.

```
public void close()
```

Discussion
Calling *close* on a closed drawer does nothing. You can get the state of a drawer by sending it `state` (page 595).

See Also
`open` (page 592)

contentSize

Returns the size of the receiver's content area.

```
public NSSize contentSize()
```

See Also
`setContentSize` (page 592)
`setMaxContentSize` (page 593)
`setMinContentSize` (page 594)

contentView

Returns the receiver's content view.

```
public NSView contentView()
```

See Also
`setContentView` (page 593)

delegate

Returns the receiver's delegate.

```
public Object delegate()
```

See Also
`setDelegate` (page 593)

edge

Returns the edge of the window that the receiver is connected to.

```
public int edge()
```

Discussion
See "Constants" (page 595) for a list of edge constants and locations.

leadingOffset

Returns the receiver's leading offset.

```
public float leadingOffset()
```

See Also
`setLeadingOffset` (page 593)

maxContentSize

Returns the maximum allowed size of the content area.

```
public NSSize maxContentSize()
```

Discussion
Useful for determining if an opened drawer would fit onscreen given the current window position.

See Also
`setMaxContentSize` (page 593)

minContentSize

Returns the minimum allowed size of the receiver's content area.

```
public NSSize minContentSize()
```

See Also
`setMinContentSize` (page 594)

open

If the receiver is closed, this method opens it.

```
public void open()
```

Discussion

Calling *open* on an open drawer does nothing. You can get the state of a drawer by sending it `state` (page 595). If an edge is not specified, an attempt will be made to choose an edge based on the space available to display the drawer onscreen. If you need to ensure that a drawer opens on a particular edge, use `openOnEdge` (page 592).

See Also

`close` (page 590)

openOnEdge

Causes the receiver to open on the specified *edge*.

```
public void openOnEdge(int edge)
```

Discussion

See "Constants" (page 595) for a list of edge constants and locations.

parentWindow

Returns the receiver's parent window

```
public NSWindow parentWindow()
```

Discussion

. By definition, a drawer can appear onscreen only if it has a parent.

See Also

`setParentWindow` (page 594)

preferredEdge

When the receiver is told to open and an edge is not specified at that time, it opens on this value.

```
public int preferredEdge()
```

Discussion

When you a create a drawer with Interface Builder, the preferred edge is set to the left by default.

See Also

`setPreferredEdge` (page 594)

setContentSize

Sets the size of the receiver's content area to *size*.

```
public void setContentSize(NSSize size)
```

Discussion

See "Positioning and Sizing a Drawer" for additional detail.

See Also

`contentSize` (page 590)

`setMaxContentSize` (page 593)

`setMinContentSize` (page 594)

setContentView

Rather than connect a drawer to its content view in Interface Builder, you can specify it programatically with the *aView* argument of this method.

```
public void setContentView(NSView aView)
```

See Also

`contentView` (page 590)

setDelegate

```
public void setDelegate(Object anObject)
```

Discussion

You may find it useful to associate a delegate with a drawer, especially since drawers do not open and close instantly. A drawer's delegate can better regulate drawer behavior. However, a drawer can be used without a delegate.

See Also

`delegate` (page 591)

setLeadingOffset

Sets the receiver's leading offset to *offset*.

```
public void setLeadingOffset(float offset)
```

Discussion

See "Positioning and Sizing a Drawer" for additional detail.

See Also

`leadingOffset` (page 591)

`setTrailingOffset` (page 594)

setMaxContentSize

Specifies the maximum *size* of the receiver's content area. See "Positioning and Sizing a Drawer" for additional detail.

```
public void setMaxContentSize(NSSize size)
```

See Also

`maxContentSize` (page 591)

setMinContentSize

Specifies the minimum *size* of the receiver's content area. See "Positioning and Sizing a Drawer" for additional detail.

```
public void setMinContentSize(NSSize size)
```

See Also

`minContentSize` (page 591)

setParentWindow

Sets the receiver's parent window to *parent*.

```
public void setParentWindow(NSWindow parent)
```

Discussion

Every drawer must be associated with a parent window for a drawer to appear onscreen. Calling `setParentWindow` with a `null` argument removes a drawer from its parent. Changes in a drawer's parent window do not take place while the drawer is onscreen; they are delayed until the drawer next closes.

See Also

`parentWindow` (page 592)

setPreferredEdge

```
public void setPreferredEdge(int preferredEdge)
```

Discussion

A drawer can be told to open on a specific `edge` (page 591). However, when the edge is not specified, the drawer is opened on the *preferredEdge*.

See Also

`preferredEdge` (page 592)

setTrailingOffset

Sets the receiver's trailing offset to *offset*.

```
public void setTrailingOffset(float offset)
```

Discussion

See "Positioning and Sizing a Drawer" for additional detail.

See Also

`leadingOffset` (page 591)

state

Returns the state of the receiver.

```
public int state()
```

Discussion
Refer to "Constants" (page 595) for more details.

toggle

If the receiver is closed, or in the process of either opening or closing, it is opened.

```
public void toggle(Object sender)
```

Discussion
Otherwise, the drawer is closed.

trailingOffset

Returns the receiver's trailing offset.

```
public float trailingOffset()
```

See Also
`setTrailingOffset` (page 594)

Constants

The following constants are defined by NSDrawer and are returned by `state` (page 595):

Constant	Description
`ClosedState`	The drawer is closed (not visible onscreen).
`OpeningState`	The drawer is in the process of opening.
`OpenState`	The drawer is open (visible onscreen).
`ClosingState`	The drawer is in the process of closing.

The following constants are defined by NSRect and can be used to specify window edges:

Constant	Description
`NSRect.MinXEdge`	The left edge of a window
`NSRect.MinYEdge`	The bottom edge of a window

Constant	Description
NSRect.MaxXEdge	The right edge of a window
NSRect.MaxYEdge	The top edge of a window

Delegate Methods

drawerDidClose

```
public abstract void drawerDidClose(NSNotification notification)
```

Discussion
Sent by the default notification center with the `DrawerDidCloseNotification` (page 597) in *notification* immediately after an NSDrawer has closed.

drawerDidOpen

```
public abstract void drawerDidOpen(NSNotification notification)
```

Discussion
Sent by the default notification center with the `DrawerDidOpenNotification` (page 597) in *notification* immediately after an NSDrawer has opened.

drawerShouldClose

```
public abstract boolean drawerShouldClose(NSDrawer sender)
```

Discussion
Invoked on user-initiated attempts to close a drawer by dragging it , but not when the `close` (page 590) method is called. The delegate can return `false` to prevent *sender* from closing.

drawerShouldOpen

```
public abstract boolean drawerShouldOpen(NSDrawer sender)
```

Discussion
Invoked on user-initiated attempts to open a drawer by dragging it , but not when the `open` (page 592) method is called. The delegate can return `false` to prevent the *sender* from opening.

drawerWillClose

```
public abstract void drawerWillClose(NSNotification notification)
```

Discussion
Sent by the default notification center with the `DrawerWillCloseNotification` (page 597) in *`notification`* immediately before an NSDrawer is closed.

drawerWillOpen

```
public abstract void drawerWillOpen(NSNotification notification)
```

Discussion
Sent by the default notification center with the `DrawerWillOpenNotification` (page 597) in *`notification`* immediately before an NSDrawer is opened.

drawerWillResizeContents

```
public abstract NSSize drawerWillResizeContents(NSDrawer sender, NSSize contentSize)
```

Discussion
Invoked when the user resizes the drawer or parent. *`contentSize`* contains the size *`sender`* will be resized to. To resize to a different size, simply return the desired size from this method; to avoid resizing, return the current size. The receiver's minimum and maximum size constraints have already been applied when this method is invoked. While the user is resizing an NSDrawer or its parent, the delegate is sent a series of `windowWillResize` messages as the NSDrawer or parent window is dragged.

Notifications

DrawerDidCloseNotification

Posted whenever the NSDrawer is closed. The notification object is the NSDrawer that closed. This notification does not contain a *`userInfo`* dictionary.

DrawerDidOpenNotification

Posted whenever the NSDrawer is opened. The notification object is the NSDrawer that opened. This notification does not contain a *`userInfo`* dictionary.

DrawerWillCloseNotification

Posted whenever the NSDrawer is about to close. The notification object is the NSDrawer about to close. This notification does not contain a *`userInfo`* dictionary.

DrawerWillOpenNotification

Posted whenever the NSDrawer is about to open. The notification object is the NSDrawer about to open. This notification does not contain a *`userInfo`* dictionary.

NSEPSImageRep

Inherits from	NSImageRep : NSObject
Implements	NSCoding (NSImageRep)
Package:	com.apple.cocoa.application
Companion guide	Drawing and Images

Overview

An NSEPSImageRep is an object that can render an image from encapsulated PostScript (EPS) code.

Tasks

Constructors

`NSEPSImageRep` (page 600)
Throws an exception. Use the other constructor instead.

Creating an NSEPSImageRep

`imageRep` (page 600)
Creates a new NSEPSImageRep instance and then calls a constructor with *`epsData`*.

Getting Image Data

`boundingBox` (page 600)
Returns the rectangle that bounds the receiver.

`EPSRepresentation` (page 601)
Returns the EPS representation of the receiver.

Drawing the Image

`prepareGState` (page 601)
Implemented by subclasses to initialize the graphics state before the image is drawn.

Constructors

NSEPSImageRep

Throws an exception. Use the other constructor instead.

```
public NSEPSImageRep()
```

Creates a new NSEPSImageRep, with the contents of `epsData`.

```
public NSEPSImageRep(NSData epsData)
```

Discussion
If the new object can't be created for any reason (for example, `epsData` doesn't contain EPS code), returns `null`.

The size of the object is set from the bounding box specified in the EPS header comments.

Static Methods

imageRep

Creates a new NSEPSImageRep instance and then calls a constructor with `epsData`.

```
public static NSEPSImageRep imageRep(NSData epsData)
```

Discussion
If the new object can't be initialized for any reason (for example, `epsData` doesn't contain EPS code), this method frees the receiver and returns `null`. Otherwise, it returns a new instance of NSEPSImageRep.

The size of the object is set from the bounding box specified in the EPS header comments.

Instance Methods

boundingBox

Returns the rectangle that bounds the receiver.

```
public NSRect boundingBox()
```

Discussion
The rectangle is obtained from the "`%%BoundingBox:`" comment in the EPS header when the NSEPSImageRep is initialized.

See Also
`imageRep` (page 600)

EPSRepresentation

Returns the EPS representation of the receiver.

```
public NSData EPSRepresentation()
```

prepareGState

Implemented by subclasses to initialize the graphics state before the image is drawn.

```
public void prepareGState()
```

Discussion

NSEPSImageRep's `draw` (page 788) method sends a `prepareGState` message just before rendering the EPS code. The default implementation of `prepareGState` does nothing.

NSEvent

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.application
Companion guide	Cocoa Event-Handling Guide

Overview

An NSEvent object, or simply an event, contains information about an input action such as a mouse click or a key down. The Application Kit associates each such user action with a window, reporting the event to the application that created the window. The NSEvent object contains pertinent information about each event, such as where the cursor was located or which character was typed. As the application receives events, it temporarily places them in a buffer called the event queue. When the application is ready to process an event, it takes one from the queue.

Beginning with Mac OS X version 10.4, NSEvent objects can represent tablet-pointing and tablet-proximity events. A tablet-proximity event is generated when a pointing device enters or leaves proximity of its tablet; such event objects have a type of `TypeProximity` or a mouse subtype of `TabletProximityEventSubtype`. A tablet-pointing event is generated when a pointing device changes state, such as location, pressure, or tilt; such event objects have a type of `TypePoint` or a mouse subtype of `TabletPointEventSubtype`. The Application Kit reports all pure tablet events to responder objects through the NSResponder methods `tabletPoint` (page 1198) and `tabletProximity` (page 1199). Mouse events can also contain tablet data (as event subtypes), so you can handle these events by overriding the NSResponder methods `mouseDown` (page 1192), `mouseDragged` (page 1192), and `mouseUp` (page 1193).

Tasks

Constructors

`NSEvent` (page 607)
Creates an empty NSEvent.

Creating Events

`eventMaskFromType` (page 607)
Returns the event mask for the given *eventType*.

`keyEvent` (page 607)
: Returns a new NSEvent object describing a key event.

`mouseEvent` (page 608)
: Returns a new NSEvent object describing a mouse-down, -up, -moved, or -dragged event.

`otherEvent` (page 609)
: Returns a new NSEvent object describing a custom event.

Requesting and Stopping Periodic Events

`startPeriodicEvents` (page 609)
: Begins generating periodic events for the current thread every *periodSeconds*, after a delay of *delaySeconds*.

`stopPeriodicEvents` (page 610)
: Stops generating periodic events for the current thread and discards any periodic events remaining in the queue.

Getting General Event Information

`context` (page 613)
: Returns the display context of the receiver.

`locationInWindow` (page 616)
: Returns the receiver's location in the base coordinate system of the associated window.

`modifierFlags` (page 616)
: Returns an integer bit field indicating the modifier keys in effect for the receiver.

`timestamp` (page 620)
: Returns the time the receiver occurred in seconds since system startup.

`type` (page 621)
: Returns the type of the receiving event.

`window` (page 622)
: Returns the window object associated with the receiver.

`windowNumber` (page 623)
: Returns the identifier for the window device associated with the receiver.

Getting Key Event Information

`characters` (page 612)
: Returns the characters associated with the receiving key-up or key-down event.

`charactersIgnoringModifiers` (page 612)
: Returns the characters generated by the receiving key event as if no modifier key (except for Shift) applies.

`isARepeat` (page 615)
: Returns `true` if the receiving key event is a repeat caused by the user holding the key down, `false` if the key event is new.

`keyCode` (page 616)
  Returns the virtual key code for the keyboard key associated with the receiving key event.

Getting Mouse Event Information

`mouseLocation` (page 609)
  Exports the current mouse position, in screen coordinates. S

`buttonNumber` (page 612)
  Returns the button number for the mouse button that generated an “OtherMouse” event.

`clickCount` (page 613)
  Returns the number of mouse clicks associated with the receiver, a mouse-down or -up event.

`pressure` (page 618)
  Returns a value from 0.0 through 1.0 indicating the pressure applied to the input device (used for appropriate devices).

Getting Tracking-rectangle Event Information

`eventNumber` (page 615)
  Returns the counter value of the latest mouse or tracking-rectangle event; every system-generated mouse and tracking-rectangle event increments this counter.

`trackingNumber` (page 620)
  Returns the identifier of the tracking rectangle for a tracking-rectangle event.

Getting Custom Event Information

`data1` (page 613)
  Returns additional data associated with the receiver.

`data2` (page 614)
  Returns additional data associated with the receiver.

`subtype` (page 618)
  Returns the subtype of the receiving custom event.

Getting Scroll Wheel Event Information

`deltaX` (page 614)
  Returns the change in x for a scroll wheel, mouse-move, or mouse-drag event.

`deltaY` (page 614)
  Returns the change in y for a scroll wheel, mouse-move, or mouse-drag event.

`deltaZ` (page 614)
  Returns the change in z for a scroll wheel, mouse-move, or mouse-drag event.

Getting Tablet Proximity Information

capabilityMask (page 612)
: Returns a mask whose set bits indicate the capabilities of the tablet device that generated the event represented by the receiver.

deviceID (page 615)
: Returns a special identifier that is used to match tablet-pointer events with the tablet-proximity event represented by the receiver.

isEnteringProximity (page 616)
: Returns true to indicate that a pointing device is entering the proximity of its tablet and NO when it is leaving it.

pointingDeviceID (page 617)
: Returns the index of the pointing device currently in proximity with the tablet.

pointingDeviceSerialNumber (page 617)
: Returns the vendor-assigned serial number of a pointing device of a certain type.

pointingDeviceType (page 617)
: Returns a constant indicating the kind of pointing device associated with the receiver.

systemTabletID (page 619)
: Returns the index of the tablet device connected to the system.

tabletID (page 619)
: Returns the USB model identifier of the tablet device associated with the receiver.

uniqueID (page 621)
: Returns the unique identifier of the pointing device that generated the event represented by the receiver.

vendorID (page 622)
: Returns the vendor identifier of the tablet associated with the receiver.

vendorPointingDeviceType (page 622)
: Returns a coded bit field whose set bits indicate the type of pointing device (within a vendor selection) associated with the receiver.

Getting Tablet Pointing Information

absoluteX (page 610)
: Reports the absolute x coordinate of a pointing device on its tablet at full tablet resolution.

absoluteY (page 610)
: Reports the absolute y coordinate of a pointing device on its tablet at full tablet resolution.

absoluteZ (page 611)
: Reports the absolute z coordinate of pointing device on its tablet at full tablet resolution.

buttonMask (page 611)
: Returns a bit mask identifying the buttons pressed when the tablet event represented by the receiver was generated.

rotation (page 618)
: Returns the rotation in degrees of the tablet pointing device associated with the receiver.

tangentialPressure (page 619)
: Reports the tangential pressure on the device that generated the event represented by the receiver.

`tilt` (page 620)
Reports the scaled tilt values of the pointing device that generated the event represented by the receiver.

`vendorDefined` (page 622)
Returns an array of three vendor-defined NSNumber objects associated with the pointing-type event represented by the receiver.

Constructors

NSEvent

Creates an empty NSEvent.

```
public NSEvent()
```

Static Methods

eventMaskFromType

Returns the event mask for the given *eventType*.

```
public static final int eventMaskFromType(int eventType)
```

Discussion
Event masks and event types are described in "Constants" (page 623).

keyEvent

Returns a new NSEvent object describing a key event.

```
public static NSEvent keyEvent(int type, NSPoint location, int flags, double time,
     int windowNum, NSGraphicsContext context, String characters, String
    unmodCharacters, boolean repeatKey, short code)
```

Discussion
type must be one of the following, or an `InternalInconsistencyException` is thrown:

```
KeyDown
KeyUp
FlagsChanged
```

The *location* argument is the cursor location in the base coordinate system of the window specified by *windowNum*.

The *flags* argument is an integer bit field containing any of the modifier key masks described in "Constants" (page 623), combined using the C bitwise OR operator.

The *time* argument is the time the event occurred in seconds since system startup.

The *windowNum* argument identifies the window device associated with the event, which is associated with the NSWindow that will receive the event.

The *context* argument is the display context of the event.

The *characters* argument is a string of characters associated with the key event. Though most key events contain only one character, it is possible for a single keypress to generate a series of characters.

The *unmodCharacters* argument is the string of characters generated by the key event as if no modifier key had been pressed (except for Shift). This argument is useful for getting the "basic" key value in a hardware-independent manner.

The *repeatKey* argument is `true` if the key event is a repeat caused by the user holding the key down, `false` if the key event is new.

The *code* argument identifies the keyboard key associated with the key event. Its value is hardware-independent.

See Also

`characters` (page 612)
`charactersIgnoringModifiers` (page 612)
`isARepeat` (page 615)
`keyCode` (page 616)

mouseEvent

Returns a new NSEvent object describing a mouse-down, -up, -moved, or -dragged event.

```
public static NSEvent mouseEvent(int type, NSPoint location, int flags, double
    time, int windowNum, NSGraphicsContext context, int eventNumber, int clickNumber,
     float pressure)
```

Discussion

type must be one of the modifier key masks described in "Constants" (page 623), or an `InternalInconsistencyException` is thrown.

The *location*, *flags*, *time*, *windowNum*, and *context* arguments are as described under `keyEvent` (page 607).

The *eventNumber* argument is an identifier for the new event. It's normally taken from a counter for mouse events, which continually increases as the application runs.

The *clickNumber* argument is the number of mouse clicks associated with the mouse event.

The *pressure* argument is a value from 0.0 to 1.0 indicating the pressure applied to the input device on a mouse event, used for an appropriate device such as a graphics tablet. For devices that aren't pressure-sensitive, the value should be either 0.0 or 1.0.

See Also

`clickCount` (page 613)
`eventNumber` (page 615)
`pressure` (page 618)

mouseLocation

Exports the current mouse position, in screen coordinates. S

```
public static NSPoint mouseLocation()
```

Discussion

imilar to NSWindow's `mouseLocationOutsideOfEventStream` (page 1843), this method returns the location regardless of the current event or pending events. The difference between these methods is that `mouseLocationOutsideOfEventStream` returns a point in the receiving window's coordinates and `mouseLocation` returns the same information in screen coordinates.

otherEvent

Returns a new NSEvent object describing a custom event.

```
public static NSEvent otherEvent(int type, NSPoint location, int flags, double
    time, int windowNum, NSGraphicsContext context, short subtype, int data1, int
     data2)
```

Discussion

type must be one of the values below, or an `InternalInconsistencyException` is thrown. Your code should only create events of type `ApplicationDefined`.

```
AppKitDefined
SystemDefined
ApplicationDefined
Periodic
```

The *location*, *flags*, *time*, *windowNum*, and *context* arguments are as described under `keyEvent` (page 607). Arguments specific to custom events are:

- The *subtype* argument further differentiates custom events of types `AppKitDefined`, `SystemDefined`, and `ApplicationDefined`. `Periodic` events don't use this attribute.
- The *data1* and *data2* arguments contain additional data associated with the event. `Periodic` events don't use these attributes.

See Also

`subtype` (page 618)

`data1` (page 613)

`data2` (page 614)

startPeriodicEvents

Begins generating periodic events for the current thread every *periodSeconds*, after a delay of *delaySeconds*.

```
public static void startPeriodicEvents(double delaySeconds, double periodSeconds)
```

Discussion

Throws an `InternalInconsistencyException` if periodic events are already being generated for the current thread. This method is typically used in a modal loop while tracking mouse-dragged events.

See Also

`stopPeriodicEvents` (page 610)

stopPeriodicEvents

Stops generating periodic events for the current thread and discards any periodic events remaining in the queue.

```
public static void stopPeriodicEvents()
```

Discussion

This message is ignored if periodic events aren't currently being generated.

See Also

`startPeriodicEvents` (page 609)

Instance Methods

absoluteX

Reports the absolute x coordinate of a pointing device on its tablet at full tablet resolution.

```
public int absoluteX()
```

Discussion

For the coordinate to be valid, the receiver should represent an event generated by a tablet pointing device (otherwise 0 is returned). This method is valid only for mouse events with a subtype of `TabletPointEventSubtype` and for events of type `TabletPoint`. Use this value if you want to scale from tablet location to screen location yourself; otherwise use the class method `mouseLocation` (page 609) or the instance method `locationInWindow` (page 616).

Availability

Available in Mac OS X v10.4 and later.

See Also

`absoluteY` (page 610)

`absoluteZ` (page 611)

absoluteY

Reports the absolute y coordinate of a pointing device on its tablet at full tablet resolution.

```
public int absoluteY()
```

Discussion
For the coordinate to be valid, the receiver should represent an event generated by a tablet pointing device (otherwise 0 is returned). This method is valid only for mouse events with a subtype of `TabletPointEventSubtype` and for events of type `TabletPoint`. Use this value if you want to scale from tablet location to screen location yourself; otherwise use the class method `mouseLocation` (page 609) or the instance method `locationInWindow` (page 616).

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

See Also
`absoluteX` (page 610)
`absoluteZ` (page 611)

absoluteZ

Reports the absolute z coordinate of pointing device on its tablet at full tablet resolution.

```
public int absoluteZ()
```

Discussion
For the coordinate to be valid, the receiver should represent an event generated by a tablet pointing device (otherwise 0 is returned). The z coordinate does not represent pressure. It registers the depth coordinate returned by some tablet devices with wheels; if the device is something other than these, 0 is returned. This method is valid only for mouse events with a subtype of `TabletPointEventSubtype` and for events of type `TabletPoint`.

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

See Also
`absoluteX` (page 610)
`absoluteY` (page 610)

buttonMask

Returns a bit mask identifying the buttons pressed when the tablet event represented by the receiver was generated.

```
public int buttonMask()
```

Discussion
Use one or more of the button-mask constants described in "Constants" (page 623) to determine which buttons of the pointing device are pressed. This method is valid only for mouse events with a subtype of `TabletPointEventSubtype` and for events of type `TabletPoint`.

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

buttonNumber

Returns the button number for the mouse button that generated an “OtherMouse” event.

```
public int buttonNumber()
```

Discussion
This method is intended for use with the “OtherMouse” events, but will return values for LeftMouse and RightMouse events, also.

capabilityMask

Returns a mask whose set bits indicate the capabilities of the tablet device that generated the event represented by the receiver.

```
public int capabilityMask()
```

Discussion
These bits are vendor-defined. This method is valid only for mouse events with a subtype of `TabletProximityEventSubtype` and for events of type `TabletProximity`.

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

characters

Returns the characters associated with the receiving key-up or key-down event.

```
public String characters()
```

Discussion
These characters are derived from a keyboard mapping that associates various key combinations with Unicode characters. Throws an `InternalInconsistencyException` if sent to any other kind of event.

This method will return an empty string for dead keys, such as option-E.

See Also
`charactersIgnoringModifiers` (page 612)
`keyEvent` (page 607)

charactersIgnoringModifiers

Returns the characters generated by the receiving key event as if no modifier key (except for Shift) applies.

```
public String charactersIgnoringModifiers()
```

Discussion
Throws an `InternalInconsistencyException` if sent to a nonkey event.

This method will return an empty string for dead keys, such as option-E.

This method is useful for determining "basic" key values in a hardware-independent manner, enabling such features as keyboard equivalents defined in terms of modifier keys plus character keys. For example, to determine if the user typed Alt-S, you don't have to know whether Alt-S generates a German double ess, an integral sign, or a section symbol. You simply examine the string returned by this method along with the event's modifier flags, checking for "s" and `AlternateKeyMask`.

See Also
`characters` (page 612)
`modifierFlags` (page 616)
`keyEvent` (page 607)

clickCount

Returns the number of mouse clicks associated with the receiver, a mouse-down or -up event.

```
public int clickCount()
```

Discussion
Throws an `InternalInconsistencyException` if sent to a nonmouse event.

Returns 0 for a mouse-up event if a time threshold has passed since the corresponding mouse-down event. This is because if this time threshold passes before the mouse button is released, it is no longer considered a mouse click, but a mouse-down event followed by a mouse-up event.

The return value of this method is meaningless for events other than mouse-down or -up events.

See Also
`mouseEvent` (page 608)

context

Returns the display context of the receiver.

```
public NSGraphicsContext context()
```

data1

Returns additional data associated with the receiver.

```
public int data1()
```

Discussion
The value returned by this method is dependent on the event type, and is defined by the originator of the event. Throws an `InternalInconsistencyException` if sent to an event not of type `AppKitDefined`, `SystemDefined`, `ApplicationDefined`, or `Periodic`.

`Periodic` events don't use this attribute.

See Also
`data2` (page 614)
`subtype` (page 618)

otherEvent (page 609)

data2

Returns additional data associated with the receiver.

```
public int data2()
```

Discussion

The value returned by this method is dependent on the event type, and is defined by the originator of the event. Throws an InternalInconsistencyException if sent to an event not of type AppKitDefined, SystemDefined, ApplicationDefined, or Periodic.

Periodic events don't use this attribute.

See Also

data1 (page 613)

subtype (page 618)

otherEvent (page 609)

deltaX

Returns the change in x for a scroll wheel, mouse-move, or mouse-drag event.

```
public float deltaX()
```

See Also

deltaY (page 614)

deltaZ (page 614)

deltaY

Returns the change in y for a scroll wheel, mouse-move, or mouse-drag event.

```
public float deltaY()
```

See Also

deltaX (page 614)

deltaZ (page 614)

deltaZ

Returns the change in z for a scroll wheel, mouse-move, or mouse-drag event.

```
public float deltaZ()
```

Discussion

This value is typically 0.0.

See Also

deltaX (page 614)

deltaY (page 614)

deviceID

Returns a special identifier that is used to match tablet-pointer events with the tablet-proximity event represented by the receiver.

```
public int deviceID()
```

Discussion

All tablet-pointer events generated in the period between the device entering and leaving tablet proximity have the same device ID. This message is valid only for mouse events with subtype `TabletPointEventSubtype` or `TabletProximityEventSubtype`, and for `TabletPoint` and `TabletProximity` events.

Availability

Available in Mac OS X v10.4 and later.

See Also

pointingDeviceID (page 617)

systemTabletID (page 619)

tabletID (page 619)

eventNumber

Returns the counter value of the latest mouse or tracking-rectangle event; every system-generated mouse and tracking-rectangle event increments this counter.

```
public int eventNumber()
```

Discussion

Throws an `InternalInconsistencyException` if sent to any other type of event.

See Also

mouseEvent (page 608)

isARepeat

Returns `true` if the receiving key event is a repeat caused by the user holding the key down, `false` if the key event is new.

```
public boolean isARepeat()
```

Discussion

Throws an `InternalInconsistencyException` if sent to a `FlagsChanged` event or other nonkey event.

See Also

keyEvent (page 607)

isEnteringProximity

Returns `true` to indicate that a pointing device is entering the proximity of its tablet and `NO` when it is leaving it.

```
public boolean isEnteringProximity()
```

Discussion

This method is valid for mouse events with subtype `TabletProximityEventSubtype` and for `TabletProximity` events.

Availability

Available in Mac OS X v10.4 and later.

keyCode

Returns the virtual key code for the keyboard key associated with the receiving key event.

```
public short keyCode()
```

Discussion

Its value is hardware-independent. The value returned is the same as the value returned in the `kEventParamKeyCode` when using Carbon Events.

Throws an `InternalInconsistencyException` if sent to a non-key event.

See Also

`keyEvent` (page 607)

locationInWindow

Returns the receiver's location in the base coordinate system of the associated window.

```
public NSPoint locationInWindow()
```

Discussion

For nonmouse events the return value of this method is undefined.

In a method of a custom view that handles mouse events, you commonly use the `locationInWindow` method in conjunction with the NSView method `convertPointFromView` (page 1744)to get the mouse location in the view's coordinate system. For example:

```
NSPoint event_location = [theEvent locationInWindow];
NSPoint local_point = [self convertPoint:event_location fromView:nil];
```

See Also

`window` (page 622)

modifierFlags

Returns an integer bit field indicating the modifier keys in effect for the receiver.

```
public int modifierFlags()
```

Discussion

You can examine individual flag settings using the C bitwise AND operator with the predefined key masks described in "Constants" (page 623). The lower 16 bits of the modifier flags are reserved for device-dependent bits.

pointingDeviceID

Returns the index of the pointing device currently in proximity with the tablet.

```
public int pointingDeviceID()
```

Discussion

This index is significant for multimode (or Dual Tracking) tablets that support multiple concurrent pointing devices; the index is incremented for each pointing device that comes into proximity. Otherwise, zero is always returned. The receiver of this message should be a mouse event object with subtype `TabletProximityEventSubtype` or an event of type `TabletProximity`.

Availability

Available in Mac OS X v10.4 and later.

See Also

`pointingDeviceSerialNumber` (page 617)

`pointingDeviceType` (page 617)

`systemTabletID` (page 619)

pointingDeviceSerialNumber

Returns the vendor-assigned serial number of a pointing device of a certain type.

```
public int pointingDeviceSerialNumber()
```

Discussion

Devices of different types, such as a puck and a pen, may have the same serial number. The receiver of this message should be a mouse event object with subtype `TabletProximityEventSubtype` or an event of type `TabletProximity`.

Availability

Available in Mac OS X v10.4 and later.

See Also

`pointingDeviceID` (page 617)

`pointingDeviceType` (page 617)

pointingDeviceType

Returns a constant indicating the kind of pointing device associated with the receiver.

```
public int pointingDeviceType()
```

Discussion

For example, the device could be a pen, eraser, or cursor pointing device. This method is valid for mouse events with subtype `TabletProximityEventSubtype` and for `TabletProximity` events. See "Constants" (page 623) for descriptions of valid constants.

Availability

Available in Mac OS X v10.4 and later.

See Also

`pointingDeviceSerialNumber` (page 617)

`pointingDeviceType` (page 617)

pressure

Returns a value from 0.0 through 1.0 indicating the pressure applied to the input device (used for appropriate devices).

```
public float pressure()
```

Discussion

For devices that aren't pressure-sensitive, the value is either 0.0 or 1.0. Throws an `InternalInconsistencyException` if sent to a nonmouse event.

For tablet pointing devices that are in proximity, the pressure value is 0.0 if they are not actually touching the tablet. As the device is pressed into the tablet, the value is increased.

See Also

`mouseEvent` (page 608)

`rotation` (page 618)

rotation

Returns the rotation in degrees of the tablet pointing device associated with the receiver.

```
public float rotation()
```

Discussion

Many devices do not support rotation, in which case the returned value is 0.0. This method is valid only for mouse events with subtype `TabletPointEventSubtype` and for `TabletPoint` events.

Availability

Available in Mac OS X v10.4 and later.

See Also

`pressure` (page 618)

`tilt` (page 620)

subtype

Returns the subtype of the receiving custom event.

```
public short subtype()
```

Discussion

Throws an `InternalInconsistencyException` if sent to an event not of type `AppKitDefined`, `SystemDefined`, `ApplicationDefined`, or `Periodic`.

`Periodic` events don't use this attribute.

See Also

`data1` (page 613)

`data2` (page 614)

`otherEvent` (page 609)

systemTabletID

Returns the index of the tablet device connected to the system.

```
public int systemTabletID()
```

Discussion

If multiple tablets are connected to the system, the system-tablet ID is incremented for each subsequent one. If there is only one tablet device, its system-tablet ID is zero. The receiver of this message should be a mouse event object with subtype `TabletProximityEventSubtype` or an event of type `TabletProximity`.

Availability

Available in Mac OS X v10.4 and later.

See Also

`pointingDeviceID` (page 617)

`tabletID` (page 619)

tabletID

Returns the USB model identifier of the tablet device associated with the receiver.

```
public int tabletID()
```

Discussion

This method is valid for mouse events with subtype `TabletProximityEventSubtype` and for `TabletProximity` events.

Availability

Available in Mac OS X v10.4 and later.

See Also

`pointingDeviceID` (page 617)

`systemTabletID` (page 619)

tangentialPressure

Reports the tangential pressure on the device that generated the event represented by the receiver.

```
public float tangentialPressure()
```

Discussion

The value returned can range from -1.0 to 1.0. Tangential pressure is also known as barrel pressure. Only some pointing devices support tangential pressure. This method is valid for mouse events with subtype `TabletPointEventSubtype` and for `TabletPoint` events.

Availability

Available in Mac OS X v10.4 and later.

See Also

`pressure` (page 618)

tilt

Reports the scaled tilt values of the pointing device that generated the event represented by the receiver.

```
public NSPoint tilt()
```

Discussion

The value returned can range from -1.0 to 1.0 for both axes. A x value that is negative indicates a tilt to the left and a positive value indicates a tilt to the right; a y value that is negative indicates a tilt to the top and a positive value indicates a tilt to the bottom. If the device is perfectly perpendicular to the table surface, the values are 0.0 for both axes. This method is valid for mouse events with subtype `TabletPointEventSubtype` and for `TabletPoint` events.

Availability

Available in Mac OS X v10.4 and later.

See Also

`pressure` (page 618)

`rotation` (page 618)

timestamp

Returns the time the receiver occurred in seconds since system startup.

```
public double timestamp()
```

trackingNumber

Returns the identifier of the tracking rectangle for a tracking-rectangle event.

```
public int trackingNumber()
```

Discussion

Throws an `InternalInconsistencyException` if sent to any other type of event.

type

Returns the type of the receiving event.

```
public int type()
```

Discussion

The type must be one of the following:

```
LeftMouseDown
LeftMouseUp
RightMouseDown
RightMouseUp
OtherMouseDown
OtherMouseUp
MouseMoved
LeftMouseDragged
RightMouseDragged
OtherMouseDragged
MouseEntered
MouseExited
KeyDown
KeyUp
FlagsChanged
AppKitDefined
SystemDefined
ApplicationDefined
Periodic
CursorUpdate
ScrollWheel
```

uniqueID

Returns the unique identifier of the pointing device that generated the event represented by the receiver.

```
public int uniqueID()
```

Discussion

Also known as tool ID, this is a unique number recorded in the chip inside every pointing device. The unique ID makes it possible to assign a specific pointing device to a specific tablet. You can also use it to "sign" documents or to restrict access to document layers to a specific pointing device. This method is valid for mouse events with subtype `TabletProximityEventSubtype` and for `TabletProximity` events.

Availability

Available in Mac OS X v10.4 and later.

See Also

`vendorDefined` (page 622)

`vendorID` (page 622)

vendorDefined

Returns an array of three vendor-defined NSNumber objects associated with the pointing-type event represented by the receiver.

```
public Object vendorDefined()
```

Discussion

The NSNumbers encapsulate `short` values that vendors may return for various reasons; see the vendor documentation for details.This method is valid for mouse events with subtype `TabletPointEventSubtype` and for `TabletPoint` events.

Availability

Available in Mac OS X v10.4 and later.

vendorID

Returns the vendor identifier of the tablet associated with the receiver.

```
public int vendorID()
```

Discussion

The tablet is typically a USB device. This method is valid only for mouse events with subtype `TabletProximityEventSubtype` and for `TabletProximity` events.

Availability

Available in Mac OS X v10.4 and later.

See Also

`tabletID` (page 619)

`vendorPointingDeviceType` (page 622)

vendorPointingDeviceType

Returns a coded bit field whose set bits indicate the type of pointing device (within a vendor selection) associated with the receiver.

```
public int vendorPointingDeviceType()
```

Discussion

See the vendor documentation for an interpretation of significant bits. This method is valid only for mouse events with subtype `TabletProximityEventSubtype` and for `TabletProximity` events.

Availability

Available in Mac OS X v10.4 and later.

See Also

`vendorID` (page 622)

window

Returns the window object associated with the receiver.

```
public NSWindow window()
```

Discussion

A periodic event, however, has no window; in this case the return value is undefined.

See Also

`windowNumber` (page 623)

windowNumber

Returns the identifier for the window device associated with the receiver.

```
public int windowNumber()
```

Discussion

A periodic event, however, has no window; in this case the return value is undefined.

See Also

`window` (page 622)

Constants

These constants represent various kinds of events. They are returned by `type` (page 621) and are used as the first argument to the methods `keyEvent` (page 607), `mouseEvent` (page 608), and `otherEvent` (page 609).

Constant	Description
`LeftMouseDown`	See “Mouse Events”.
`LeftMouseUp`	See “Mouse Events”.
`RightMouseDown`	See “Mouse Events”.
`RightMouseUp`	See “Mouse Events”.
`OtherMouseDown`	See “Mouse Events”.
`OtherMouseUp`	See “Mouse Events”.
`MouseMoved`	See “Mouse Events”.
`LeftMouseDragged`	See “Mouse Events”.
`RightMouseDragged`	See “Mouse Events”.
`OtherMouseDragged`	See “Mouse Events”.
`MouseEntered`	See “Tracking-Rectangle and Cursor-Update Events”.
`MouseExited`	See “Tracking-Rectangle and Cursor-Update Events”.
`CursorUpdate`	See “Tracking-Rectangle and Cursor-Update Events”.

Constant	Description
KeyDown	See “Keyboard Events”.
KeyUp	See “Keyboard Events”.
FlagsChanged	See “Keyboard Events”.
AppKitDefined	See “Other Events”.
SystemDefined	See “Other Events”.
ApplicationDefined	See “Other Events”.
Periodic	See “Periodic Events”.
ScrollWheel	See “Mouse Events”.
TabletPoint	An event representing the current state of a tablet pointing device, including its location, pressure, and tilt. Available in Mac OS X v10.4 and later.
TabletProximity	An event representing the proximity of a pointing device to its tablet. Available in Mac OS X v10.4 and later.

These constants are masks for the events listed above. Pass them to the NSCell method `setEventMaskForSendingAction` (page 324) to specify when an NSCell should send its action message.

Constant	Description
LeftMouseDownMask	Corresponds to `LeftMouseDown`. See “Mouse Events”.
LeftMouseUpMask	Corresponds to `LeftMouseUp`. See “Mouse Events”.
RightMouseDownMask	Corresponds to `RightMouseDown`. See “Mouse Events”.
RightMouseUpMask	Corresponds to `RightMouseUp`. See “Mouse Events”.
OtherMouseDownMask	Corresponds to `OtherMouseDown`. See “Mouse Events”.
OtherMouseUpMask	Corresponds to `OtherMouseUp`. See “Mouse Events”.
MouseMovedMask	Corresponds to `MouseMoved`. See “Mouse Events”.
LeftMouseDraggedMask	Corresponds to `LeftMouseDragged`. See “Mouse Events”.
RightMouseDraggedMask	Corresponds to `RightMouseDragged`. See “Mouse Events”.
OtherMouseDraggedMask	Corresponds to `OtherMouseDragged`. See “Mouse Events”.
MouseEnteredMask	Corresponds to `MouseEntered`. See “Tracking-Rectangle and Cursor-Update Events”.
MouseExitedMask	Corresponds to `MouseExited`. See “Tracking-Rectangle and Cursor-Update Events”.

Constant	Description
CursorUpdateMask	Corresponds to CursorUpdate. See "Tracking-Rectangle and Cursor-Update Events".
KeyDownMask	Corresponds to KeyDown. See "Keyboard Events".
KeyUpMask	Corresponds to KeyUp. See "Keyboard Events".
FlagsChangedMask	Corresponds to FlagsChanged. See "Keyboard Events".
AppKitDefinedMask	Corresponds to AppKitDefined. See "Other Events".
SystemDefinedMask	Corresponds to SystemDefined. See "Other Events".
ApplicationDefinedMask	Corresponds to ApplicationDefined. See "Other Events".
PeriodicMask	Corresponds to Periodic. See "Periodic Events".
ScrollWheelMask	Corresponds to ScrollWheel. See "Mouse Events".
TabletPointMask	Corresponds to TabletPoint. Available in Mac OS X v10.4 and later.
TabletProximityMask	Corresponds to TabletProximity. Available in Mac OS X v10.4 and later.
AnyEventMask	Corresponds to any of the above events.

The following constants represent pointing-device types for TabletProximity events or mouse events with subtype TabletProximityEventSubtype. The pointingDeviceType (page 617) method returns one of these constants.

Constant	Description
UnknownPointingDevice	Represents an unknown type of pointing device. Available in Mac OS X v10.4 and later.
PenPointingDevice	Represents the tip end of a stylus-like pointing device. Available in Mac OS X v10.4 and later.
CursorPointingDevice	Represents a cursor (or puck-like) pointing device. Available in Mac OS X v10.4 and later.
EraserPointingDevice	Represents the eraser end of a stylus-like pointing device. Available in Mac OS X v10.4 and later.

The following constants represent button masks for TabletPoint events or mouse events with subtype TabletPointEventSubtype. The buttonMask (page 611) method returns a bit mask, which you test with one or more of these constants to determine the state of the buttons on a tablet pointing device.

Constant	Description
PenTipMask	The pen tip is activated. Available in Mac OS X v10.4 and later.
PenLowerSideMask	The button on the lower side of the device is activated. Available in Mac OS X v10.4 and later.
PenUpperSideMask	The button on the upper side of the device is activated. Available in Mac OS X v10.4 and later.

The following constants represent mouse-event subtypes for mouse and tablet events (accessed with the subtype (page 618) method).

Constant	Description
MouseEventSubtype	Indicates a purely mouse event. Available in Mac OS X v10.4 and later.
TabletPointEventSubtype	Indicates a tablet-pointer event; see description of TabletPoint. Available in Mac OS X v10.4 and later.
TabletProximityEventSubtype	Indicates a tablet-proximity event; see description of TabletProximity. Available in Mac OS X v10.4 and later.

The following constants (except for DeviceIndependentModifierFlagsMask) represent device-independent bits found in event modifier flags:

Constant	Description
AlphaShiftKeyMask	Set if Caps Lock key is pressed.
ShiftKeyMask	Set if Shift key is pressed.
ControlKeyMask	Set if Control key is pressed.
AlternateKeyMask	Set if Option or Alternate key is pressed.
CommandKeyMask	Set if Command key is pressed.
NumericPadKeyMask	Set if any key in the numeric keypad is pressed. The numeric keypad is generally on the right side of the keyboard.
HelpKeyMask	Set if the Help key is pressed.
FunctionKeyMask	Set if any function key is pressed. The function keys include the F keys at the top of most keyboards (F1, F2, and so on) and the navigation keys in the center of most keyboards (Help, Forward Delete, Home, End, Page Up, Page Down, and the arrow keys).

Constant	Description
DeviceIndependent-ModifierFlagsMask	Used to retrieve only the device-independent modifier flags, allowing applications to mask off the device-dependent modifier flags, including event coalescing information. Available in Mac OS X v10.4.

These constants represent Unicode characters (0xF700–0xF8FF) that are reserved for function keys on the keyboard. Combined in Strings, they are the return values of the NSEvent methods `characters` (page 612) and `charactersIgnoringModifiers` (page 612) and may be used in some parameters in the NSEvent method `keyEvent` (page 607).

Note that some function keys are handled at a lower level and are never seen by your application. They include the Volume Up key, Volume Down key, Volume Mute key, Eject key, and Function key found on many iBook and PowerBook computers.

Constant	Description
UpArrowFunctionKey	Up Arrow key.
DownArrowFunctionKey	Down Arrow key.
LeftArrowFunctionKey	Left Arrow key.
RightArrowFunctionKey	Right Arrow key.
F1FunctionKey	F1 key.
F2FunctionKey	F2 key.
F3FunctionKey	F3 key.
F4FunctionKey	F4 key.
F5FunctionKey	F5 key.
F6FunctionKey	F6 key.
F7FunctionKey	F7 key.
F8FunctionKey	F8 key.
F9FunctionKey	F9 key.
F10FunctionKey	F10 key.
F11FunctionKey	F11 key.
F12FunctionKey	F12 key.
F13FunctionKey	F13 key.
F14FunctionKey	F14 key.
F15FunctionKey	F15 key.

Constant	Description
`F16FunctionKey`	F16 key. Not on most Macintosh keyboards.
`F17FunctionKey`	F17 key. Not on most Macintosh keyboards.
`F18FunctionKey`	F18 key. Not on most Macintosh keyboards.
`F19FunctionKey`	F19 key. Not on most Macintosh keyboards.
`F20FunctionKey`	F20 key. Not on most Macintosh keyboards.
`F21FunctionKey`	F21 key. Not on most Macintosh keyboards.
`F22FunctionKey`	F22 key. Not on most Macintosh keyboards.
`F23FunctionKey`	F23 key. Not on most Macintosh keyboards.
`F24FunctionKey`	F24 key. Not on most Macintosh keyboards.
`F25FunctionKey`	F25 key. Not on most Macintosh keyboards.
`F26FunctionKey`	F26 key. Not on most Macintosh keyboards.
`F27FunctionKey`	F27 key. Not on most Macintosh keyboards.
`F28FunctionKey`	F28 key. Not on most Macintosh keyboards.
`F29FunctionKey`	F29 key. Not on most Macintosh keyboards.
`F30FunctionKey`	F30 key. Not on most Macintosh keyboards.
`F31FunctionKey`	F31 key. Not on most Macintosh keyboards.
`F32FunctionKey`	F32 key. Not on most Macintosh keyboards.
`F33FunctionKey`	F33 key. Not on most Macintosh keyboards.
`F34FunctionKey`	F34 key. Not on most Macintosh keyboards.
`F35FunctionKey`	F35 key. Not on most Macintosh keyboards.
`InsertFunctionKey`	Insert key. Not on most Macintosh keyboards.
`DeleteFunctionKey`	Forward Delete key.
`HomeFunctionKey`	Home key.
`BeginFunctionKey`	Begin key. Not on most Macintosh keyboards.
`EndFunctionKey`	End key.
`PageUpFunctionKey`	Page Up key.
`PageDownFunctionKey`	Page Down key.
`PrintScreenFunctionKey`	Print Screen key. Not on most Macintosh keyboards.

Constant	Description
ScrollLockFunctionKey	Scroll Lock key. Not on most Macintosh keyboards.
PauseFunctionKey	Pause key. Not on most Macintosh keyboards.
SysReqFunctionKey	System Request key. Not on most Macintosh keyboards.
BreakFunctionKey	Break key. Not on most Macintosh keyboards.
ResetFunctionKey	Reset key. Not on most Macintosh keyboards.
StopFunctionKey	Stop key. Not on most Macintosh keyboards.
MenuFunctionKey	Menu key. Not on most Macintosh keyboards.
UserFunctionKey	User key. Not on most Macintosh keyboards.
SystemFunctionKey	System key. Not on most Macintosh keyboards.
PrintFunctionKey	Print key. Not on most Macintosh keyboards.
ClearLineFunctionKey	Clear/Num Lock key.
ClearDisplayFunctionKey	Clear Display key. Not on most Macintosh keyboards.
InsertLineFunctionKey	Insert Line key. Not on most Macintosh keyboards.
DeleteLineFunctionKey	Delete Line key. Not on most Macintosh keyboards.
InsertCharFunctionKey	Insert Character key. Not on most Macintosh keyboards.
DeleteCharFunctionKey	Delete Character key. Not on most Macintosh keyboards.
PrevFunctionKey	Previous key. Not on most Macintosh keyboards.
NextFunctionKey	Next key. Not on most Macintosh keyboards.
SelectFunctionKey	Select key. Not on most Macintosh keyboards.
ExecuteFunctionKey	Execute key. Not on most Macintosh keyboards.
UndoFunctionKey	Undo key. Not on most Macintosh keyboards.
RedoFunctionKey	Redo key. Not on most Macintosh keyboards.
FindFunctionKey	Find key. Not on most Macintosh keyboards.
HelpFunctionKey	Help key.
ModeSwitchFunctionKey	Mode Switch key. Not on most Macintosh keyboards.

NSFileWrapper

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.application
Companion guide	Application File Management

Overview

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

Tasks

Constructors

`NSFileWrapper` (page 633)
Creates an empty NSFileWrapper.

Writing to a File or Serializing

`writeToFile` (page 641)
Writes the receiver's contents to a file or directory at *path*.

`serializedRepresentation` (page 639)
Returns the receiver's contents as an opaque collection of data, in the format used for the pasteboard type `NSPasteboard.FileContentsPboardType`.

Checking a File Wrapper's Type

`isRegularFile` (page 637)
Returns `true` if the receiver is a regular file wrapper, `false` otherwise.

`isDirectory` (page 637)
Returns `true` if the receiver is a directory wrapper, `false` otherwise.

`isSymbolicLink` (page 637)
: Returns `true` if the receiver is a link wrapper, `false` otherwise.

Setting Attributes

`setFilename` (page 639)
: Sets the filename for the disk representation of the receiver to *`filename`*.

`filename` (page 636)
: Returns the filename for the last known disk representation of the receiver, or `null` if the receiver has no filename.

`setPreferredFilename` (page 640)
: Sets the receiver's preferred filename to *`filename`*.

`preferredFilename` (page 638)
: Returns the file wrapper's preferred filename.

`setIcon` (page 640)
: Sets the image that can be used to represent the file wrapper to the user to *`anImage`*.

`icon` (page 636)
: Returns an image that can be used to represent the file wrapper to the user, or `null` if the file wrapper has none.

`setFileAttributes` (page 639)
: Sets the file attributes applied whenever the file wrapper is saved using `writeToFile` (page 641) to *`attributes`*.

`fileAttributes` (page 636)
: Returns the file attributes last read from disk or set using `setFileAttributes` (page 639).

Updating

`needsToBeUpdatedFromPath` (page 638)
: Returns `true` if the receiver's contents on disk may have changed, `false` otherwise.

`updateFromPath` (page 640)
: Rereads the file wrapper's information from the file or directory at *`path`*, including contents or link destination, icon, and file attributes.

Modifying a Directory Wrapper

`addFileWrapper` (page 634)
: Adds *`wrapper`* to the receiving directory wrapper.

`removeFileWrapper` (page 638)
: Removes *`wrapper`* from the receiving directory wrapper and releases it.

`addFileWithPath` (page 634)
: Adds a new file wrapper to the receiving directory wrapper.

`addRegularFileWithContents` (page 635)
: Adds a new regular file wrapper to the receiving directory wrapper.

`addSymbolicLinkWithDestination` (page 635)
Adds a new link wrapper to the receiving directory wrapper.

`fileWrappers` (page 636)
Returns the file wrappers contained in a directory wrapper.

`keyForFileWrapper` (page 637)
Returns the key by which the receiving directory wrapper stores *wrapper* in its dictionary (as returned by the `fileWrappers` (page 636) method).

Inspecting a Regular File Wrapper

`regularFileContents` (page 638)
Returns the contents of the receiving regular file wrapper.

Inspecting a Link Wrapper

`symbolicLinkDestination` (page 640)
Returns the actual path represented by the receiving link wrapper.

Constructors

NSFileWrapper

Creates an empty NSFileWrapper.

```
public NSFileWrapper()
```

Creates a new NSFileWrapper, based on *symLink*.

```
public NSFileWrapper(String path, boolean symLink)
```

Discussion
If *symLink* is `true`, the NSFileWrapper is a link wrapper pointing to *path*. The new file wrapper has no filename or associated disk representation until you save it using `writeToFile` (page 641). It's also initialized with open permissions; anyone can read or write the disk representations it saves.

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

Creates a new NSFileWrapper based on *serialized*.

```
public NSFileWrapper(NSData data, boolean serialized)
```

Discussion
If *serialized* is true, the NSFileWrapper is initialized with *data*, setting its type to regular file, directory, or link wrapper based on the nature of that data and reading the file attributes from the data as well. *data* is a serialized representation of a file's or directory's contents in the format used for the pasteboard type NSPasteboard.FileContentsPboardType. Data of this format is returned by such methods as serializedRepresentation (page 639) or NSAttributedString's RTFFromRange.

The new file wrapper has no filename or associated disk representation until you save it using writeToFile (page 641).

If *serialized* is false, the NSFileWrapper is initialized as a regular file wrapper with *data*. The new file wrapper has no filename or associated disk representation until you save it using writeToFile (page 641). It's also initialized with open permissions; anyone can read or write the disk representations that it saves.

Creates a new NSFileWrapper as a directory wrapper containing wrappers.

```
public NSFileWrapper(NSDictionary aDictionary)
```

Discussion
The new directory wrapper has no filename or associated disk representation until you save it using writeToFile (page 641). It's also initialized with open permissions; anyone can read, write, or change directory to the disk representations that it saves.

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

Instance Methods

addFileWithPath

Adds a new file wrapper to the receiving directory wrapper.

```
public String addFileWithPath(String path)
```

Discussion
Initializes the new file wrapper using *path* , then adds the new file wrapper by invoking addFileWrapper (page 634). Returns the dictionary key used for the newly added file wrapper within the directory wrapper. Throws an InternalInconsistencyException if sent to a regular file or link wrapper.

See Also
addRegularFileWithContents (page 635)
addSymbolicLinkWithDestination (page 635)
removeFileWrapper (page 638)
fileWrappers (page 636)

addFileWrapper

Adds *wrapper* to the receiving directory wrapper.

```
public String addFileWrapper(NSFileWrapper wrapper)
```

Discussion

Returns the dictionary key used for *wrapper* within the directory wrapper. Throws an InternalInconsistencyException if sent to a regular file or link wrapper or an InvalidArgumentException if *wrapper* doesn't have a preferred name (set using setPreferredFilename (page 640)).

See Also

addFileWithPath (page 634)

addRegularFileWithContents (page 635)

addSymbolicLinkWithDestination (page 635)

removeFileWrapper (page 638)

fileWrappers (page 636)

addRegularFileWithContents

Adds a new regular file wrapper to the receiving directory wrapper.

```
public String addRegularFileWithContents(NSData contents, String filename)
```

Discussion

Initializes the new file wrapper using *contents*, sets its preferred name with setPreferredFilename (page 640) using *filename* as the argument, then adds the new file wrapper by invoking addFileWrapper (page 634). Returns the dictionary key used for the newly added file wrapper within the directory wrapper. Throws an InternalInconsistencyException if sent to a regular file or link wrapper or an InvalidArgumentException if *filename* is null or empty.

See Also

addFileWithPath (page 634)

addSymbolicLinkWithDestination (page 635)

removeFileWrapper (page 638)

fileWrappers (page 636)

addSymbolicLinkWithDestination

Adds a new link wrapper to the receiving directory wrapper.

```
public String addSymbolicLinkWithDestination(String path, String filename)
```

Discussion

Initializes the new link wrapper , sets its preferred name with setPreferredFilename (page 640) using *filename* as the argument, then adds the new link wrapper by invoking addFileWrapper (page 634). Returns the dictionary key used for the newly added link wrapper within the directory wrapper. Throws an InternalInconsistencyException if sent to a regular file or link wrapper or an InvalidArgumentException if *filename* is null or empty.

See Also

addFileWithPath (page 634)

addFileWrapper (page 634)

addRegularFileWithContents (page 635)

removeFileWrapper (page 638)

fileWrappers (page 636)

fileAttributes

Returns the file attributes last read from disk or set using setFileAttributes (page 639).

```
public NSDictionary fileAttributes()
```

Discussion
These attributes are used whenever the file wrapper is saved using writeToFile (page 641). See NSPathUtilities for information on the contents of the attributes dictionary.

filename

Returns the filename for the last known disk representation of the receiver, or null if the receiver has no filename.

```
public String filename()
```

Discussion
The filename is used for record-keeping purposes only and is set automatically when the file wrapper is created from disk and when it's saved to a disk using writeToFile (page 641) (although this method allows you to request that the filename not be updated).

See Also
preferredFilename (page 638)
setFilename (page 639)

fileWrappers

Returns the file wrappers contained in a directory wrapper.

```
public NSDictionary fileWrappers()
```

Discussion
Throws an InternalInconsistencyException if sent to a regular file or link wrapper. See "Working With Directory Wrappers" for information on the dictionary.

See Also
filename (page 636)
addFileWrapper (page 634)

icon

Returns an image that can be used to represent the file wrapper to the user, or null if the file wrapper has none.

```
public NSImage icon()
```

Discussion

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

See Also

`setIcon` (page 640)

isDirectory

Returns `true` if the receiver is a directory wrapper, `false` otherwise.

```
public boolean isDirectory()
```

See Also

`isRegularFile` (page 637)

`isSymbolicLink` (page 637)

isRegularFile

Returns `true` if the receiver is a regular file wrapper, `false` otherwise.

```
public boolean isRegularFile()
```

See Also

`isDirectory` (page 637)

`isSymbolicLink` (page 637)

isSymbolicLink

Returns `true` if the receiver is a link wrapper, `false` otherwise.

```
public boolean isSymbolicLink()
```

See Also

`isDirectory` (page 637)

`isRegularFile` (page 637)

keyForFileWrapper

Returns the key by which the receiving directory wrapper stores *wrapper* in its dictionary (as returned by the `fileWrappers` (page 636) method).

```
public String keyForFileWrapper(NSFileWrapper wrapper)
```

Discussion

This key is not necessarily the filename for *wrapper*. Throws an `InternalInconsistencyException` if sent to a regular file or link wrapper.

See Also

`filename` (page 636)

needsToBeUpdatedFromPath

Returns `true` if the receiver's contents on disk may have changed, `false` otherwise.

```
public boolean needsToBeUpdatedFromPath(String path)
```

Discussion

For a regular file wrapper, whether contents have changed is determined by comparing the modification time and access permissions of the file or directory at *path* against those of the receiver. For a link wrapper, whether contents have changed is determined by checking whether the destination path has changed (not by checking the modification time or access attributes of the destination). For a directory, whether contents have changed is determined as needed recursively for each file wrapper contained in the directory; added or removed files also count as changes.

See Also

`updateFromPath` (page 640)

`fileAttributes` (page 636)

preferredFilename

Returns the file wrapper's preferred filename.

```
public String preferredFilename()
```

Discussion

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

See Also

`setPreferredFilename` (page 640)

`filename` (page 636)

regularFileContents

Returns the contents of the receiving regular file wrapper.

```
public NSData regularFileContents()
```

Discussion

Throws an `InternalInconsistencyException` if sent to a directory or link wrapper.

removeFileWrapper

Removes *wrapper* from the receiving directory wrapper and releases it.

```
public void removeFileWrapper(NSFileWrapper wrapper)
```

Discussion

Throws an `InternalInconsistencyException` if sent to a regular file or link wrapper.

See Also

`addFileWithPath` (page 634)

`addFileWrapper` (page 634)

`addRegularFileWithContents` (page 635)

`addSymbolicLinkWithDestination` (page 635)

`fileWrappers` (page 636)

serializedRepresentation

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

```
public NSData serializedRepresentation()
```

setFileAttributes

Sets the file attributes applied whenever the file wrapper is saved using `writeToFile` (page 641) to *attributes*.

```
public void setFileAttributes(NSDictionary attributes)
```

Discussion

See NSPathUtilities for information on the contents of the *attributes* dictionary.

See Also

`fileAttributes` (page 636)

setFilename

Sets the filename for the disk representation of the receiver to *filename*.

```
public void setFilename(String filename)
```

Discussion

The filename is used for record-keeping purposes only and is set automatically when the file wrapper is saved to a disk using `writeToFile` (page 641) (although this method allows you to request that the filename not be updated). You should rarely need to invoke this method.

Throws an `InvalidArgumentException` if *filename* is `null` or empty.

See Also

`setPreferredFilename` (page 640)

`filename` (page 636)

setIcon

Sets the image that can be used to represent the file wrapper to the user to *anImage*.

```
public void setIcon(NSImage anImage)
```

Discussion

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

See Also

`icon` (page 636)

setPreferredFilename

Sets the receiver's preferred filename to *filename*.

```
public void setPreferredFilename(String filename)
```

Discussion

This name is used as the default dictionary key and filename when a file wrapper is added to a directory wrapper. However, if another file wrapper with the same preferred name already exists in the directory wrapper when the receiver is added, the dictionary key and filename assigned may differ from the preferred filename. Throws an `InvalidArgumentException` if *filename* is `null` or empty.

See Also

`preferredFilename` (page 638)

`addFileWrapper` (page 634)

`setFilename` (page 639)

symbolicLinkDestination

Returns the actual path represented by the receiving link wrapper.

```
public String symbolicLinkDestination()
```

Discussion

Throws `InternalInconsistencyException` if sent to a regular file or directory wrapper.

updateFromPath

Rereads the file wrapper's information from the file or directory at *path*, including contents or link destination, icon, and file attributes.

```
public boolean updateFromPath(String path)
```

Discussion

For a directory wrapper, the contained file wrappers are also sent `updateFromPath` messages. If files in the directory on disk have been added or removed, corresponding file wrappers are released or created as needed. Returns `true` if updating actually occurred, `false` if it wasn't necessary.

See Also

`needsToBeUpdatedFromPath` (page 638)

writeToFile

Writes the receiver's contents to a file or directory at *path*.

```
public boolean writeToFile(String path, boolean atomicFlag, boolean updateNamesFlag)
```

Discussion

Returns `true` on success and `false` on failure. If *atomicFlag* is `true`, attempts to write the file safely so that an existing file at *path* is not overwritten, nor does a new file at *path* actually get created, unless the write is successful. If *updateNamesFlag* is `true` and the contents are successfully written, changes the receiver's filename to the last component of *path*, and the filenames of any children of a directory wrapper to the filenames under which they're written to disk.

If you're executing a Save or Save As operation, pass `true` for *updateNamesFlag*; if you're executing a Save To operation, pass `false` for *updateNamesFlag*.

See Also

`filename` (page 636)

NSFont

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.application
Companion guide	Font Handling

Overview

NSFont objects represent fonts to an application, providing access to characteristics of the font and assistance in laying out glyphs relative to one another. Font objects are also used to establish the current font when drawing in an NSView, using the `set` (page 661) method.

You don't create NSFont objects using a constructor. Instead, you use `fontWithNameAndSize` (page 648) to look up an available font and alter its size or matrix to your needs. These methods check for an existing font object with the specified characteristics, returning it if there is one. Otherwise, they look up the font data requested and create the appropriate object. NSFont also defines a number of methods for getting standard system fonts, such as `systemFontOfSize` (page 652), `userFontOfSize` (page 654), and `messageFontOfSize` (page 650). To request the default size for these standard fonts, pass a negative number or 0 as the font size.

Tasks

Constructors

`NSFont` (page 647)
Creates an empty NSFont.

Creating Arbitrary Fonts

`fontWithNameAndSize` (page 648)
Returns a font object for *`fontName`* and *`fontSize`*.

`fontWithNameAndMatrix` (page 648)
Returns a font object for *`typeface`* and *`fontMatrix`*.

Creating User Fonts

`userFontOfSize` (page 654)
Returns the font used by default for documents and other text under the user's control (that is, text whose font the user can normally change).

`userFixedPitchFontOfSize` (page 653)
Returns the font used by default for documents and other text under the user's control (that is, text whose font the user can normally change), when that font should be fixed-pitch.

Creating System Fonts

`boldSystemFontOfSize` (page 648)
Returns the Aqua system font used for standard interface items that are rendered in boldface type, in *fontSize*.

`controlContentFontOfSize` (page 648)
Returns the font used for the content of controls, in *fontSize*.

`labelFontOfSize` (page 649)
Returns the Aqua font used for standard interface labels in *fontSize*.

`menuFontOfSize` (page 649)
Returns the font used for menu items in *fontSize*.

`menuBarFontOfSize` (page 649)
Returns the font used for menu bar items in *fontSize*.

`messageFontOfSize` (page 650)
Returns the font used for standard interface items, such as button labels, menu items, and so on, in *fontSize*.

`paletteFontOfSize` (page 650)
Returns the font used for palette window title bars.

`systemFontOfSize` (page 652)
Returns the Aqua system font used for standard interface items, such as button labels, menu items, and so on, in *fontSize*.

`titleBarFontOfSize` (page 652)
Returns the font used for window title bars, in *fontSize*.

`toolTipsFontOfSize` (page 653)
Returns the font used for tool tips labels, in *fontSize*.

Getting Preferred Fonts

`setPreferredFontNames` (page 650)
This method is deprecated.

`preferredFontNames` (page 650)
This method is deprecated. NSFontCascadeListAttribute offers more powerful font substitution management.

Using a Font to Draw

set (page 661)
: Establishes the receiver as the current font for PostScript show and other text-drawing operators.

Getting General Font Information

coveredCharacterSet (page 656)
: Returns an NSCharacterSet containing all of the nominal characters renderable by the receiver, which is all of the entries mapped in the receiver's 'cmap' table.

encodingScheme (page 657)
: This method is deprecated.

fontDescriptor (page 657)
: Returns the receiver's font descriptor.

isBaseFont (page 658)
: This method is deprecated.

isFixedPitch (page 659)
: Returns true if all glyphs in the receiver have the same advancement, false if any advancements differ.

mostCompatibleStringEncoding (page 659)
: Returns the string encoding that works best with the receiver, where there are the fewest possible unmatched characters in the string encoding and glyphs in the font.

Getting Information About Glyphs

glyphIsEncoded (page 658)
: This method is deprecated. The value can be deduced by aGlyph < [NSFont numberOfGlyphs] since only NSNativeShortGlyphPacking is supported.

glyphPacking (page 658)
: This method is deprecated.

glyphWithName (page 658)
: Returns the encoded glyph named *glyphName*, or –1 if the receiver contains no such glyph.

Getting Metrics Information

labelFontSize (page 649)
: Returns the size of the standard label font.

smallSystemFontSize (page 651)
: Returns the size of the standard small system font.

systemFontSize (page 652)
: Returns the size of the standard system font.

systemFontSizeForControlSize (page 652)
: Returns the font size used for *controlSize*. If

`advancementForGlyph` (page 654)
: Returns the nominal spacing for *`aGlyph`*—the distance the current point moves after showing the glyph—accounting for the receiver's size.

`afmDictionary` (page 654)
: This method is deprecated.

`afmFileContents` (page 655)
: This method is deprecated.

`ascender` (page 655)
: Returns the top y coordinate, offset from the baseline, of the receiver's longest ascender.

`boundingRectForFont` (page 655)
: Returns the receiver's bounding rectangle, scaled to the font's size.

`boundingRectForGlyph` (page 655)
: Returns the bounding rectangle for *`aGlyph`*, scaled to the receiver's size.

`capHeight` (page 656)
: Returns the receiver's cap height.

`descender` (page 656)
: Returns the bottom y coordinate, offset from the baseline, of the receiver's longest descender.

`italicAngle` (page 659)
: Returns the receiver's italic angle, the amount that the font is slanted in degrees counterclockwise from the vertical, as read from its AFM file. Because the slant is measured counterclockwise, English italic fonts typically return a negative value.

`maximumAdvancement` (page 659)
: Returns the greatest advancement of any of the receiver's glyphs.

`numberOfGlyphs` (page 660)
: Returns the number of glyphs in the receiver.

`pointSize` (page 660)
: Returns the receiver's point size, or the effective vertical point size for a font with a nonstandard matrix.

`underlinePosition` (page 662)
: Returns the baseline offset that should be used when drawing underlines with the receiver, as determined by the font's AFM file.

`underlineThickness` (page 662)
: Returns the thickness that should be used when drawing underlines with the receiver, as determined by the font's AFM file.

`widthOfString` (page 662)
: Returns the x axis offset of the current point when *`aString`* is drawn with a PostScript `show` operator in the receiving font.

`xHeight` (page 662)
: Returns the x height of the receiver.

Getting Font Names

`displayName` (page 656)
: Returns the name, including family and face, used to represent the font in the user interface, typically localized for the user's language.

`familyName` (page 657)
Returns the receiver's family name—for example, "Times" or "Helvetica."

`fontName` (page 657)
Returns the receiver's full font name, as used in PostScript language code—for example, "Times-Roman" or "Helvetica-Oblique."

Laying out Overstruck Glyphs

`positionOfGlyphForCharacterStruckOverRect` (page 660)
This method is deprecated. Context-sensitive inter-glyph spacing is now performed at the typesetting stage.

Setting User Fonts

`setUserFont` (page 651)
Sets the font used by default for documents and other text under the user's control to *aFont*.

`setUserFixedPitchFont` (page 651)
Sets the font used by default for documents and other text under the user's control, when that font should be fixed-pitch, to *aFont*.

Getting Corresponding Device Fonts

`printerFont` (page 661)

`screenFont` (page 661)

Deprecated Methods

`useFont` (page 653)
This method is deprecated. This is now automatically handled by Quartz.

Constructors

NSFont

Creates an empty NSFont.

```
public NSFont()
```

Static Methods

boldSystemFontOfSize

Returns the Aqua system font used for standard interface items that are rendered in boldface type, in *fontSize*.

```
public static NSFont boldSystemFontOfSize(float fontSize)
```

Discussion
If *fontSize* is 0 or negative, returns the boldface system font at the default size.

See Also
`fontWithNameAndSize` (page 648)

controlContentFontOfSize

Returns the font used for the content of controls, in *fontSize*.

```
public static NSFont controlContentFontOfSize(float fontSize)
```

Discussion
For example, in a table, the user's input uses the control content font, and the table's header uses another font. If *fontSize* is 0 or negative, returns the control content font at the default size.

See Also
`fontWithNameAndSize` (page 648)

fontWithNameAndMatrix

Returns a font object for *typeface* and *fontMatrix*.

```
public static Object fontWithNameAndMatrix(String typeface, NSArray fontMatrix)
```

Discussion
typeface is a fully specified family-face name, such as Helvetica-BoldOblique or Times-Roman (not a name as shown in the Font Panel). *fontMatrix* is a standard 6-element transformation matrix as used in the PostScript language, specifically with the `makefont` operator. In most cases, you can simply use `fontWithNameAndSize` (page 648) to create standard scaled fonts.

See Also
`isFlipped` (page 1756) (NSView)

fontWithNameAndSize

Returns a font object for *fontName* and *fontSize*.

```
public static NSFont fontWithNameAndSize(String fontName, float fontSize)
```

Discussion

fontName is a fully specified family-face name, such as Helvetica-BoldOblique or Times-Roman. *fontSize* is used to scale the font. If you use a *fontSize* of 0.0, this method uses the default User Font size.

Fonts created with this method automatically flip themselves in flipped views. This method is the preferred means for creating fonts.

labelFontOfSize

Returns the Aqua font used for standard interface labels in *fontSize*.

```
public static NSFont labelFontOfSize(float fontSize)
```

Discussion

If *fontSize* is 0 or negative, returns the label font with the default size.

labelFontSize

Returns the size of the standard label font.

```
public static float labelFontSize()
```

Availability

Available in Mac OS X v10.3 and later.

menuBarFontOfSize

Returns the font used for menu bar items in *fontSize*.

```
public static NSFont menuBarFontOfSize(float fontSize)
```

Discussion

If *fontSize* is 0 or negative, returns the menu bar font with the default size.

Availability

Available in Mac OS X v10.3 and later.

See Also

`fontWithNameAndSize` (page 648)

menuFontOfSize

Returns the font used for menu items in *fontSize*.

```
public static NSFont menuFontOfSize(float fontSize)
```

Discussion

If *fontSize* is 0 or negative, returns the menu items font with the default size.

See Also

`fontWithNameAndSize` (page 648)

messageFontOfSize

Returns the font used for standard interface items, such as button labels, menu items, and so on, in *fontSize*.

```
public static NSFont messageFontOfSize(float fontSize)
```

Discussion
If *fontSize* is 0 or negative, returns this font at the default size. This method is equivalent to `systemFontOfSize` (page 652).

See Also
`fontWithNameAndSize` (page 648)

paletteFontOfSize

Returns the font used for palette window title bars.

```
public static NSFont paletteFontOfSize(float fontSize)
```

Discussion
If *fontSize* is 0 or negative, returns the palette title font at the default size.

See Also
`fontWithNameAndSize` (page 648)

`titleBarFontOfSize` (page 652)

preferredFontNames

This method is deprecated. NSFontCascadeListAttribute offers more powerful font substitution management.

```
public static NSArray preferredFontNames()
```

Discussion
Returns the names of fonts that the Application Kit tries first when a character has no font specified or when the font specified doesn't have a glyph for that character. If none of these fonts provides a glyph, the remaining fonts on the system are searched for a glyph.

Availability
Deprecated.

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

See Also
`setPreferredFontNames` (page 650)

setPreferredFontNames

This method is deprecated.

```
public static void setPreferredFontNames(NSArray fontNames)
```

Discussion

Sets the list of preferred font names to *fontNames* and records them in the user defaults database for all applications. The Application Kit tries these fonts first when a character has no font specified or when the font specified doesn't have a glyph for that character. If none of these fonts provides a glyph, the remaining fonts on the system are searched for a glyph.

This method is useful for optimizing glyph rendering for uncommon scripts, by guaranteeing that appropriate fonts are searched first. For example, suppose you have three hundred Latin alphabet fonts and one Cyrillic alphabet font. When you read a document in Russian, you want it to find the Cyrillic font quickly. Ordinarily, the Application Kit will search for the Cyrillic font among all 301 fonts. But if it is in the list of preferred fonts, the Cyrillic font will be one of the first searched.

Availability

Deprecated.

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

See Also

`preferredFontNames` (page 650)

setUserFixedPitchFont

Sets the font used by default for documents and other text under the user's control, when that font should be fixed-pitch, to *aFont*.

```
public static void setUserFixedPitchFont(NSFont aFont)
```

Discussion

Specifying *aFont* as `null` causes the default to be removed from the application domain.

See Also

`setUserFont` (page 651)

`userFixedPitchFontOfSize` (page 653)

setUserFont

Sets the font used by default for documents and other text under the user's control to *aFont*.

```
public static void setUserFont(NSFont aFont)
```

Discussion

Specifying *aFont* as `null` causes the default to be removed from the application domain.

See Also

`setUserFixedPitchFont` (page 651)

`userFontOfSize` (page 654)

smallSystemFontSize

Returns the size of the standard small system font.

```
public static float smallSystemFontSize()
```

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

systemFontOfSize

Returns the Aqua system font used for standard interface items, such as button labels, menu items, and so on, in *fontSize*.

```
public static NSFont systemFontOfSize(float fontSize)
```

Discussion
If *fontSize* is 0 or negative, returns the system font at the default size.

See Also
boldSystemFontOfSize (page 648)

userFontOfSize (page 654)

userFixedPitchFontOfSize (page 653)

fontWithNameAndSize (page 648)

systemFontSize

Returns the size of the standard system font.

```
public static float systemFontSize()
```

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

systemFontSizeForControlSize

Returns the font size used for *controlSize*. If

```
public static float systemFontSizeForControlSize(int controlSize)
```

Discussion
controlSize does not correspond to a valid control size, it returns the size of the standard system font.

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

titleBarFontOfSize

Returns the font used for window title bars, in *fontSize*.

```
public static NSFont titleBarFontOfSize(float fontSize)
```

Discussion
If *fontSize* is 0 or negative, returns the title bar font at the default size. This method is equivalent to boldSystemFontOfSize (page 648).

See Also

paletteFontOfSize (page 650)

toolTipsFontOfSize

Returns the font used for tool tips labels, in *fontSize*.

```
public static NSFont toolTipsFontOfSize(float fontSize)
```

Discussion

If *fontSize* is 0 or negative, returns the tool tips font at the default size.

See Also

fontWithNameAndSize (page 648)

useFont

This method is deprecated. This is now automatically handled by Quartz.

```
public static void useFont(String fontName)
```

Discussion

Records *fontName* as one used in the current print operation.

The NSFont class object keeps track of the fonts used in an NSView by recording each one that receives a set (page 661) message. When the view is called upon to generate conforming PostScript language output (such as during printing), the NSFont class provides the list of fonts required for the %%DocumentFonts comment, as required by Adobe's document structuring conventions.

The useFont argument augments this system by providing a way to register fonts that are included in the document but not set using NSFont's set (page 661) method. For example, you might set a font by executing the setfont operator within a function created by the pswrap utility. In such a case, be sure to pair the use of the font with a useFont message to register the font for listing in the document comments.

Availability

Deprecated.

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

userFixedPitchFontOfSize

Returns the font used by default for documents and other text under the user's control (that is, text whose font the user can normally change), when that font should be fixed-pitch.

```
public static NSFont userFixedPitchFontOfSize(float fontSize)
```

Discussion

If *fontSize* is 0 or negative, returns the fixed-pitch font at the default size.

The system does not guarantee that all the glyphs in a fixed-pitch font are the same width. For example, certain Japanese fonts are dual-pitch, and other fonts may have nonspacing marks that can affect the display of other glyphs.

See Also

`userFontOfSize` (page 654)

`fontWithNameAndSize` (page 648)

`setUserFixedPitchFont` (page 651)

userFontOfSize

Returns the font used by default for documents and other text under the user's control (that is, text whose font the user can normally change).

```
public static NSFont userFontOfSize(float fontSize)
```

Discussion

If *fontSize* is 0 or negative, returns the user font at the default size.

See Also

`userFixedPitchFontOfSize` (page 653)

`fontWithNameAndSize` (page 648)

`setUserFont` (page 651)

Instance Methods

advancementForGlyph

Returns the nominal spacing for *aGlyph*—the distance the current point moves after showing the glyph—accounting for the receiver's size.

```
public NSSize advancementForGlyph(int aGlyph)
```

Discussion

This spacing is given according to the glyph's movement direction, which is either strictly horizontal or strictly vertical.

See Also

`boundingRectForGlyph` (page 655)

`maximumAdvancement` (page 659)

afmDictionary

This method is deprecated.

```
public NSDictionary afmDictionary()
```

Discussion

It simply returns `null`.

Availability

Deprecated.

Available in Mac OS X v10.0 and later. Deprecated in Mac OS X v10.3.

See Also
`afmFileContents` (page 655)

afmFileContents

This method is deprecated.

```
public String afmFileContents()
```

Discussion
It simply returns `null`.

Availability
Deprecated.
Available in Mac OS X v10.0 and later. Deprecated in Mac OS X v10.3.

ascender

Returns the top y coordinate, offset from the baseline, of the receiver's longest ascender.

```
public float ascender()
```

See Also
`descender` (page 656)
`capHeight` (page 656)
`xHeight` (page 662)

boundingRectForFont

Returns the receiver's bounding rectangle, scaled to the font's size.

```
public NSRect boundingRectForFont()
```

Discussion
The bounding rectangle is the union of the bounding rectangles of every glyph in the font.

See Also
`boundingRectForGlyph` (page 655)

boundingRectForGlyph

Returns the bounding rectangle for *aGlyph*, scaled to the receiver's size.

```
public NSRect boundingRectForGlyph(int aGlyph)
```

Discussion
Japanese fonts encoded with the scheme "EUC12-NJE-CFEncoding" do not have individual metrics or bounding boxes available for the glyphs above 127. For those glyphs, this method returns the bounding rectangle for the font instead.

See Also
`boundingRectForFont` (page 655),

capHeight

Returns the receiver's cap height.

```
public float capHeight()
```

See Also
`ascender` (page 655)
`descender` (page 656)
`xHeight` (page 662)

coveredCharacterSet

Returns an NSCharacterSet containing all of the nominal characters renderable by the receiver, which is all of the entries mapped in the receiver's '`cmap`' table.

```
public NSCharacterSet coveredCharacterSet()
```

Discussion
The number of glyphs supported by a given font is often larger than the number of characters contained in the character set returned by this method.

Availability
Available in Mac OS X v10.2 and later.

descender

Returns the bottom y coordinate, offset from the baseline, of the receiver's longest descender.

```
public float descender()
```

Discussion
Thus, if the longest descender extends 2 pixels below the baseline, `descender` will return –2.

displayName

Returns the name, including family and face, used to represent the font in the user interface, typically localized for the user's language.

```
public String displayName()
```

encodingScheme

This method is deprecated.

```
public String encodingScheme()
```

Discussion
Returns the name of the receiver's encoding scheme, such as "AdobeStandardEncoding," "ISOLatin1Encoding," "FontSpecific," and so on.

Availability
Deprecated.

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

familyName

Returns the receiver's family name—for example, "Times" or "Helvetica."

```
public String familyName()
```

Discussion
This name is the one that NSFontManager uses and may differ slightly from the AFM name.

The value returned by this method is intended for an application's internal usage and not for display. Use `displayName` (page 656) instead.

See Also
`fontName` (page 657)

fontDescriptor

Returns the receiver's font descriptor.

```
public NSFontDescriptor fontDescriptor()
```

Discussion
The font descriptor contains a mutable dictionary of optional attributes for creating an NSFont object. See documentation on NSFontDescriptor (page 665) for more information.

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

fontName

Returns the receiver's full font name, as used in PostScript language code—for example, "Times-Roman" or "Helvetica-Oblique."

```
public String fontName()
```

Discussion
The value returned by this method is intended for an application's internal usage and not for display. Use `displayName` (page 656) instead.

See Also

`familyName` (page 657)

glyphIsEncoded

This method is deprecated. The value can be deduced by `aGlyph < [NSFont numberOfGlyphs]` since only `NSNativeShortGlyphPacking` is supported.

```
public boolean glyphIsEncoded(int aGlyph)
```

Discussion

Returns `true` if the receiver encodes *`aGlyph`*, `false` if it doesn't contain it.

Availability

Deprecated.

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

glyphPacking

This method is deprecated.

```
public int glyphPacking()
```

Discussion

Returns the best way to encode the receiver's glyphs into an array of bytes. The return value is one of values described in "Constants" (page 663).

Availability

Deprecated.

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

glyphWithName

Returns the encoded glyph named *`glyphName`*, or –1 if the receiver contains no such glyph.

```
public int glyphWithName(String glyphName)
```

Discussion

Returns –1 if the glyph named *`glyphName`* isn't encoded.

Glyph names in fonts do not always accurately identify the glyph. If possible, look up the appropriate glyph on your own.

isBaseFont

This method is deprecated.

```
public boolean isBaseFont()
```

Discussion

Returns true if the receiver is a PostScript base font, false if it's a PostScript composite font composed of other base fonts.

Availability

Deprecated.

Available in Mac OS X v10.0 and later. Deprecated in Mac OS X v10.3.

isFixedPitch

Returns true if all glyphs in the receiver have the same advancement, false if any advancements differ.

```
public boolean isFixedPitch()
```

Discussion

Some Japanese fonts encoded with the scheme "EUC12-NJE-CFEncoding" return that they have the same advancement, but actually encode glyphs with one of two advancements, for historical compatibility. You may need to handle such fonts specially for some applications.

See Also

advancementForGlyph (page 654)

italicAngle

Returns the receiver's italic angle, the amount that the font is slanted in degrees counterclockwise from the vertical, as read from its AFM file. Because the slant is measured counterclockwise, English italic fonts typically return a negative value.

```
public float italicAngle()
```

maximumAdvancement

Returns the greatest advancement of any of the receiver's glyphs.

```
public NSSize maximumAdvancement()
```

Discussion

This advancement is always either strictly horizontal or strictly vertical.

See Also

advancementForGlyph (page 654)

mostCompatibleStringEncoding

Returns the string encoding that works best with the receiver, where there are the fewest possible unmatched characters in the string encoding and glyphs in the font.

```
public int mostCompatibleStringEncoding()
```

Discussion

You can use NSStringReferences's `dataUsingEncoding` method to convert the string to this encoding.

If this method returns `NSStringReference.ASCIIStringEncoding`, it could not determine the correct encoding and assumed that the font can render only ASCII characters.

This method works heuristically using well-known font encodings, so for nonstandard encodings it may not in fact return the optimal string encoding.

See Also

`widthOfString` (page 662)

numberOfGlyphs

Returns the number of glyphs in the receiver.

```
public int numberOfGlyphs()
```

Discussion

Glyphs are numbered starting at 0.

pointSize

Returns the receiver's point size, or the effective vertical point size for a font with a nonstandard matrix.

```
public float pointSize()
```

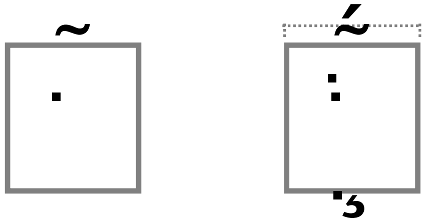
positionOfGlyphForCharacterStruckOverRect

This method is deprecated. Context-sensitive inter-glyph spacing is now performed at the typesetting stage.

```
public NSPoint positionOfGlyphForCharacterStruckOverRect(int aGlyph, char aChar,
    NSRect aRect)
```

Discussion

Calculates and returns a suitable location for *aGlyph* to be drawn as a diacritic or nonspacing mark relative to *aRect*, assuming that *aGlyph* represents *aChar*. Returns `NSPoint.ZeroPoint` if the location can't be calculated. The nature of *aChar* as one appearing above or below its base character determines the location returned. For example, in the first figure below, the gray tilde and box represent *aGlyph* and *aRect*, and the black dot is the point returned (defined relative to the origin of the *aRect*).



To place multiple glyphs with respect to a rectangle, work from the innermost glyphs to the outermost. As you calculate the position of each glyph, enlarge the rectangle to include the bounding rectangle of the glyph in preparation for the next glyph. The second figure shows a tilde, acute accent, and cedilla all placed in their appropriate positions with respect to a rectangle, with the acute accent placed relative to the expanded bounding box of the base rectangle and the tilde.

Availability
Deprecated.

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

printerFont

```
public NSFont printerFont()
```

Discussion
When sent to a font object representing a scalable PostScript font, returns `this`. When sent to a font object representing a bitmapped screen font, returns its corresponding scalable PostScript font.

See Also
`screenFont` (page 661)

screenFont

```
public NSFont screenFont()
```

Discussion
When sent to a font object representing a scalable PostScript font, returns a bitmapped screen font matching the receiver in typeface and matrix (or size), or `null` if such a font can't be found. When sent to a font object representing a bitmapped screen font, returns `null`.

Screen fonts are for direct use with the window server only. Never use them with Application Kit objects, such as in `setFont` methods. Internally, the Application Kit automatically uses the corresponding screen font for a font object as long as the view is not rotated or scaled.

See Also
`printerFont` (page 661)

set

Establishes the receiver as the current font for PostScript `show` and other text-drawing operators.

```
public void set()
```

Discussion
During a print operation, also records the font as used in the PostScript code emitted.

See Also
`useFont` (page 653)

underlinePosition

Returns the baseline offset that should be used when drawing underlines with the receiver, as determined by the font's AFM file.

```
public float underlinePosition()
```

Discussion

This value is usually negative, which must be considered when drawing in a flipped coordinate system.

See Also

`underlineThickness` (page 662)

underlineThickness

Returns the thickness that should be used when drawing underlines with the receiver, as determined by the font's AFM file.

```
public float underlineThickness()
```

See Also

`underlinePosition` (page 662)

widthOfString

Returns the x axis offset of the current point when *aString* is drawn with a PostScript `show` operator in the receiving font.

```
public float widthOfString(String aString)
```

Discussion

This method performs lossy conversion of *aString* to the most compatible encoding for the receiving font.

Use this method only when you're sure all of *aString* can be rendered with the receiving font.

This method is for backward compatibility only.

See Also

`mostCompatibleStringEncoding` (page 659)

xHeight

Returns the x height of the receiver.

```
public float xHeight()
```

See Also

`ascender` (page 655)

`descender` (page 656)

Constants

Cocoa stores all text data as Unicode. The text system converts Unicode into glyph IDs and places them in 1-, 2-, or 4-byte storage depending on the context. To render text, you must convert the storage into a format the text engine understands. The following constants describe the glyph packing schemes the text rendering engine can use. They are used to extract glyphs from a font for making a multibyte (or single-byte) array of glyphs for passing to an interpreter, such as the window server, which expects a big-endian multibyte stream (that is, "packed glyphs") instead of a pure NSGlyph stream. They're used by `glyphPacking` (page 658). With Quartz, the engine always expects the format to be in 2 -byte short array, so `NativeShortGlyphPacking` is the only format currently in use.

Constant	Description
`OneByteGlyphPacking`	Deprecated in Mac OS X v10.3
`JapaneseEUCGlyphPacking`	Deprecated in Mac OS X v10.3
`NativeShortGlyphPacking`	The native format for Mac OS X
`WithDoubleByteEUCGlyphPacking`	Deprecated in Mac OS X v10.3
`TwoByteGlyphPacking`	Deprecated in Mac OS X v10.4
`FourByteGlyphPacking`	Deprecated in Mac OS X v10.4

NSFontDescriptor

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.application
Availability	Available in Mac OS X v10.3 and later.
Companion guide	Font Handling

Overview

NSFontDescriptor objects provide a mechanism to describe a font with a dictionary of attributes. This font descriptor can be used later to create or modify an NSFont object.

All attributes in the attributes dictionary are optional.

Tasks

Constructors

`NSFontDescriptor` (page 665)
Creates an empty NSFontDescriptor.

Getting Information About a Font Descriptor

`fontAttributes` (page 666)
Returns the receiver's dictionary of attributes.

Constructors

NSFontDescriptor

Creates an empty NSFontDescriptor.

```
public NSFontDescriptor()
```

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

Creates an NSFontDescriptor with a dictionary of *attributes*.

```
public NSFontDescriptor(NSDictionary attributes)
```

Discussion
If *attributes* is `null`, the font descriptor's dictionary will be empty.

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

Instance Methods

fontAttributes

Returns the receiver's dictionary of attributes.

```
public NSDictionary fontAttributes()
```

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

Constants

The following font attributes are defined by NSFontDescriptor:

Attribute Identifier	Value Class	Default Value
`FontFamilyAttribute`	String	None (optional)
`FontNameAttribute`	String	None (optional)
`FontFaceAttribute`	String	None (optional)
`FontSizeAttribute`	String, as a float	None (optional)
`FontVisibleNameAttribute`	String	None (optional)
`FontColorAttribute`	NSData	None (optional)

NSFontManager

Inherits from NSObject

Package: com.apple.cocoa.application

Companion guide Font Panel

Overview

NSFontManager is the center of activity for the font conversion system. It records the currently selected font, updates the Font panel and Font menu to reflect the selected font, initiates font changes, and converts fonts in response to requests from text-bearing objects. In a more prosaic role, NSFontManager can be queried for the fonts available to the application and for the particular attributes of a font, such as whether it's condensed or extended.

You normally set up a font manager and the Font menu using Interface Builder. However, you can also do so programmatically by getting the shared font manager instance and having it create the standard Font menu at runtime.

You can then add the Font menu to your 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.

As of Mac OS X version 10.3, font collections are managed by NSFontManager.

Tasks

Constructors

`NSFontManager` (page 671)
Returns the shared font manager, creating it if necessary.

Getting the Shared Font Manager

`sharedFontManager` (page 671)
Returns the shared instance of the font manager for the application, creating it if necessary.

Getting Available Fonts

`availableFonts` (page 673)
: Returns the names of the fonts available in the system (not the NSFont objects themselves).

`availableFontFamilies` (page 673)
: Returns the names of the font families available in the system.

`availableFontNamesWithTraits` (page 673)
: Returns the names of the fonts available in the system whose traits are described exactly by *`fontTraitMask`* (not the NSFont objects themselves).

Setting and Examining the Selected Font

`setSelectedFont` (page 684)
: Records *`aFont`* as the currently selected font and updates the Font panel to reflect this.

`selectedFont` (page 682)
: Returns the last font recorded with a `setSelectedFont` (page 684) message.

`isMultiple` (page 679)
: Returns `true` if the last font selection recorded has multiple fonts, `false` if it's a single font.

`sendAction` (page 682)
: Sends the receiver's action message up the responder chain, initiating a font change for whatever conversion and trait to change were last requested.

`localizedNameForFamily` (page 680)
: Returns a localized string with the name of the specified font *`family`* and *`face`* (for example, "`Times`" and "`Roman`"), if one exists.

Action Methods

`addFontTrait` (page 672)
: This action method causes the receiver to send its action message up the responder chain.

`removeFontTrait` (page 682)
: This action method causes the receiver to send its action message up the responder chain.

`modifyFont` (page 680)
: This action method causes the receiver to send its action message up the responder chain.

`modifyFontViaPanel` (page 680)
: This action method causes the receiver to send its action message up the responder chain.

`orderFrontStylesPanel` (page 681)
: This action method opens the Font styles panel.

`orderFrontFontPanel` (page 681)
: This action method opens the Font panel by sending it an `orderFront` (page 1844) message, creating the Font panel if necessary.

Converting Fonts Automatically

`convertFont` (page 674)
: Converts *aFont* according to the object that initiated a font change, typically the Font panel or Font menu.

Converting Fonts Manually

`convertFontToFace` (page 675)
: Returns an NSFont whose traits are as similar as possible to those of *aFont* except for the typeface, which is changed to *typeface*.

`convertFontToFamily` (page 675)
: Returns an NSFont whose traits are as similar as possible to those of *aFont* except for the font family, which is changed to *family*.

`convertFontToHaveTrait` (page 675)
: Returns an NSFont whose traits are the same as those of *aFont* except for the traits, which are changed to include the single trait *fontTrait*.

`convertFontToNotHaveTrait` (page 676)
: Returns an NSFont with the same traits as *aFont* except for the traits in *fontTraitMask*, which are removed.

`convertFontToSize` (page 676)
: Returns an NSFont whose traits are the same as those of *aFont* except for the size, which is changed to *size*.

`convertWeight` (page 677)
: Returns an NSFont whose weight is greater or lesser than that of *aFont*, if possible.

Getting a Particular Font

`fontWithFamily` (page 679)
: Attempts to load a font with the specified characteristics, returning the font if successful and `null` if not.

Examining Fonts

`traitsOfFont` (page 684)
: Returns the traits of *aFont*.

`fontWithNameHasTraits` (page 679)
: Returns `true` if the font named *typeface* has all the traits specified in *fontTraitMask*, `false` if it doesn't.

`weightOfFont` (page 684)
: Returns a rough numeric measure the weight of *aFont*, where 0 indicates the lightest possible weight, 5 indicates a normal or book weight, and 9 or more indicates a bold or heavier weight.

Enabling the Font Panel and Font Menu

`setEnabled` (page 683)
: Controls whether the font conversion system's user interface items (the Font panel and Font menu items) are enabled.

`isEnabled` (page 679)
: Returns `true` if the font conversion system's user interface items (the Font panel and Font menu items) are enabled, `false` if they're not.

Setting the Font Menu

`setFontMenu` (page 683)
: Records *`aMenu`* as the application's Font menu.

`fontMenu` (page 678)
: Returns the menu that's hooked up to the font conversion system, creating it if necessary if *`createFlag`* is `true`.

Getting the Font Panel

`fontPanel` (page 678)
: Returns the application's shared Font panel object, creating it if necessary if *`createFlag`* is `true`.

Setting the Delegate

`setDelegate` (page 683)
: Sets the receiver's delegate to *`anObject`*.

`delegate` (page 678)
: Returns the receiver's delegate.

Setting the Action Method

`setAction` (page 683)
: Sets the action that's sent to the first responder when the user selects a new font from the Font panel or chooses a command from the Font menu to *`aSelector`*.

`action` (page 672)
: Returns the action sent to the first responder when the user selects a new font from the Font panel or chooses a command from the Font menu.

Setting Attributes

`setSelectedAttributes` (page 683)
: Informs the paragraph and character formatting panels when text in a selection has changed attributes.

`convertAttributes` (page 674)

Working with Font Descriptors

`collectionNames` (page 674)
Returns the names of the currently loaded font collections.

`fontDescriptorsInCollection` (page 678)
Returns an array of the font descriptors in the collection specified by *`collectionNames`*.

`addCollection` (page 672)
Adds a font collection named *`collectionName`* to the font manager with a set of options described in "Constants" (page 685).

`removeCollection` (page 681)
Removes the collection specified by *`collectionName`*.

`addFontDescriptors` (page 672)
Adds an array of font descriptors to the font collection specified by *`collectionName`*.

`removeFontDescriptor` (page 681)
Removes the font collection named *`collectionName`* from the collection specified by *`collection`*.

Implemented by responders

`fontManagerWillIncludeFont` (page 685) *delegate method*
Requests permission from the delegate to display *`fontName`* in the Font panel.

Constructors

NSFontManager

Returns the shared font manager, creating it if necessary.

```
public NSFontManager()
```

See Also
`sharedFontManager` (page 671)

Static Methods

sharedFontManager

Returns the shared instance of the font manager for the application, creating it if necessary.

```
public static NSFontManager sharedFontManager()
```

Instance Methods

action

Returns the action sent to the first responder when the user selects a new font from the Font panel or chooses a command from the Font menu.

```
public NSSelector action()
```

See Also
`setAction` (page 683)

addCollection

Adds a font collection named *collectionName* to the font manager with a set of options described in "Constants" (page 685).

```
public boolean addCollection(String collectionName, int collectionOptions)
```

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

See Also
`removeCollection` (page 681)

addFontDescriptors

Adds an array of font descriptors to the font collection specified by *collectionName*.

```
public void addFontDescriptors(NSArray descriptors, String collectionName)
```

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

See Also
`removeFontDescriptor` (page 681)

addFontTrait

This action method causes the receiver to send its action message up the responder chain.

```
public void addFontTrait(Object sender)
```

Discussion
When a responder replies by providing a font to convert in a `convertFont` (page 674) message, the receiver converts the font by adding the trait specified by *sender*. This trait is determined by sending a `tag` message to *sender* and interpreting it as a font trait mask for a `convertFontToHaveTrait` (page 675) message.

See Also

`removeFontTrait` (page 682)

`modifyFont` (page 680)

`modifyFontViaPanel` (page 680)

availableFontFamilies

Returns the names of the font families available in the system.

```
public NSArray availableFontFamilies()
```

Discussion

These fonts are in various system font directories.

See Also

`availableFontNamesWithTraits` (page 673)

`availableFonts` (page 673)

availableFontNamesWithTraits

Returns the names of the fonts available in the system whose traits are described exactly by *fontTraitMask* (not the NSFont objects themselves).

```
public NSArray availableFontNamesWithTraits(int fontTraitMask)
```

Discussion

These fonts are in various system font directories. You specify the desired traits by combining the font trait mask values described in "Constants" (page 685) using the C bitwise OR operator.

If *fontTraitMask* is 0, this method returns all fonts that are neither italic nor bold. This result is the same one you'd get if *fontTraitMask* were `UnitalicMask | UnboldMask`.

See Also

`availableFontFamilies` (page 673)

`availableFonts` (page 673)

availableFonts

Returns the names of the fonts available in the system (not the NSFont objects themselves).

```
public NSArray availableFonts()
```

Discussion

These fonts are in various system font directories.

See Also

`availableFontFamilies` (page 673)

`availableFontNamesWithTraits` (page 673)

collectionNames

Returns the names of the currently loaded font collections.

```
public NSArray collectionNames()
```

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

See Also
`fontDescriptorsInCollection` (page 678)

convertAttributes

```
public NSDictionary convertAttributes(NSDictionary attributes)
```

Discussion
Converts *attributes* in response to an object initiating an attribute change, typically the Font panel or Font menu. Returns the converted attributes dictionary, or *attributes* itself if the conversion isn't possible. Attributes unused by the sender should not be changed or removed.

This method is usually invoked on the sender of `changeAttributes` (page 1625). See NSTextView (page 1609) for more information.

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

See Also
`setSelectedAttributes` (page 683)

convertFont

Converts *aFont* according to the object that initiated a font change, typically the Font panel or Font menu.

```
public NSFont convertFont(NSFont aFont)
```

Discussion
Returns the converted font, or *aFont* itself if the conversion isn't possible.

This method is invoked in response to an action message such as `addFontTrait` (page 672) or `modifyFontViaPanel` (page 680). These initiating methods cause the font manager to query the sender for the action to take and the traits to change. See "Converting Fonts Manually" for more information.

See Also
`convertFontToFace` (page 675)
`convertFontToFamily` (page 675)
`convertFontToHaveTrait` (page 675)
`convertFontToNotHaveTrait` (page 676)
`convertFontToSize` (page 676)
`convertWeight` (page 677)

convertFontToFace

Returns an NSFont whose traits are as similar as possible to those of *aFont* except for the typeface, which is changed to *typeface*.

```
public NSFont convertFontToFace(NSFont aFont, String typeface)
```

Discussion

Returns *aFont* if it can't be converted. A typeface is a fully specified family-face name, such as Helvetica-BoldOblique or Times-Roman.

This method attempts to match the weight and posture of *aFont* as closely as possible. Italic is mapped to Oblique, for example. Weights are mapped based on an approximate numeric scale of 0 to 15.

See Also

`convertFontToFamily` (page 675)

`convertFontToHaveTrait` (page 675)

`convertFontToNotHaveTrait` (page 676)

`convertFontToSize` (page 676)

`convertWeight` (page 677)

`convertFont` (page 674)

convertFontToFamily

Returns an NSFont whose traits are as similar as possible to those of *aFont* except for the font family, which is changed to *family*.

```
public NSFont convertFontToFamily(NSFont aFont, String family)
```

Discussion

Returns *aFont* if it can't be converted. A family is a generic font name, such as Helvetica or Times.

This method attempts to match the weight and posture of *aFont* as closely as possible. Italic is mapped to Oblique, for example. Weights are mapped based on an approximate numeric scale of 0 to 15.

See Also

`convertFontToFace` (page 675)

`convertFontToHaveTrait` (page 675)

`convertFontToNotHaveTrait` (page 676)

`convertFontToSize` (page 676)

`convertWeight` (page 677)

`convertFont` (page 674)

convertFontToHaveTrait

Returns an NSFont whose traits are the same as those of *aFont* except for the traits, which are changed to include the single trait *fontTrait*.

```
public NSFont convertFontToHaveTrait(NSFont aFont, int fontTrait)
```

Discussion

fontTrait may be any one of the traits described in "Constants" (page 685).

Using UnboldMask or UnitalicMask removes the bold or italic trait, respectively.

Returns *aFont* if it can't be converted.

See Also

convertFontToNotHaveTrait (page 676)
convertFontToFace (page 675)
convertFontToFamily (page 675)
convertFontToSize (page 676)
convertWeight (page 677)
convertFont (page 674)

convertFontToNotHaveTrait

Returns an NSFont with the same traits as *aFont* except for the traits in *fontTraitMask*, which are removed.

```
public NSFont convertFontToNotHaveTrait(NSFont aFont, int fontTraitMask)
```

Discussion

fontTraitMask is a mask created using the C bitwise OR operator to combine the traits described in "Constants" (page 685).

Using BoldMask or ItalicMask removes the bold or italic trait, respectively.

Returns *aFont* if it can't be converted.

See Also

convertFontToHaveTrait (page 675)
convertFontToFace (page 675)
convertFontToFamily (page 675)
convertFontToSize (page 676)
convertWeight (page 677)
convertFont (page 674)

convertFontToSize

Returns an NSFont whose traits are the same as those of *aFont* except for the size, which is changed to *size*.

```
public NSFont convertFontToSize(NSFont aFont, float size)
```

Discussion

Returns *aFont* if it can't be converted.

See Also

convertFontToFace (page 675)
convertFontToFamily (page 675)
convertFontToHaveTrait (page 675)
convertFontToNotHaveTrait (page 676)

`convertWeight` (page 677)
`convertFont` (page 674)

convertWeight

Returns an NSFont whose weight is greater or lesser than that of *aFont*, if possible.

```
public NSFont convertWeight(boolean increaseFlag, NSFont aFont)
```

Discussion

If *increaseFlag* is `true`, a heavier font is returned; if it's `false`, a lighter font is returned. Returns *aFont* unchanged if it can't be converted.

Weights are graded along the following scale. The list on the left gives Apple's terminology, and the list on the right gives the ISO equivalents. Names on the same line are treated as identical:

Apple Terminology	ISO Equivalent
1. ultralight	
2. thin	W1. ultralight
3. light, extralight	W2. extralight
4. book	W3. light
5. regular, plain, display, roman	W4. semilight
6. medium	W5. medium
7. demi, demibold	
8. semi, semibold	W6. semibold
9. bold	W7. bold
10. extra, extrabold	W8. extrabold
11. heavy, heavyface	
12. black, super	W9. ultrabold
13. ultra, ultrablack, fat	
14. extrablack, obese, nord	

NSFontManager's implementation of this method refuses to convert a font's weight if it can't maintain all other traits, such as italic and condensed. You might wish to override this method to allow a looser interpretation of weight conversion.

See Also

`convertFontToFace` (page 675)
`convertFontToFamily` (page 675)

`convertFontToHaveTrait` (page 675)
`convertFontToNotHaveTrait` (page 676)
`convertFontToSize` (page 676)
`convertFont` (page 674)

delegate

Returns the receiver's delegate.

```
public Object delegate()
```

See Also
`setDelegate` (page 683)

fontDescriptorsInCollection

Returns an array of the font descriptors in the collection specified by *`collectionNames`*.

```
public NSArray fontDescriptorsInCollection(String collectionNames)
```

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

See Also
`collectionNames` (page 674)

fontMenu

Returns the menu that's hooked up to the font conversion system, creating it if necessary if *`createFlag`* is `true`.

```
public NSMenu fontMenu(boolean createFlag)
```

See Also
`setFontMenu` (page 683)

fontPanel

Returns the application's shared Font panel object, creating it if necessary if *`createFlag`* is `true`.

```
public NSFontPanel fontPanel(boolean createFlag)
```

See Also
`sharedFontPanel` (page 689) (NSFontPanel)
`sharedFontPanelExists` (page 689) (NSFontPanel)

fontWithFamily

Attempts to load a font with the specified characteristics, returning the font if successful and `null` if not.

```
public NSFont fontWithFamily(String family, int fontTraitMask, int weight, float
    size)
```

Discussion

family is the generic name of the font desired, such as Times or Helvetica. *weight* is a hint for the weight desired, on a scale of 0 to 15: a value of 5 indicates a normal or book weight, and 9 or more a bold or heavier weight. The weight is ignored if *fontTraitMask* includes `BoldMask`.

You specify *fontTraitMask* by combining the font trait mask values described in "Constants" (page 685) using the C bitwise OR operator.

Using `UnboldMask` or `UnitalicMask` loads a font that doesn't have either the bold or italic trait, respectively.

fontWithNameHasTraits

Returns `true` if the font named *typeface* has all the traits specified in *fontTraitMask*, `false` if it doesn't.

```
public boolean fontWithNameHasTraits(String typeface, int fontTraitMask)
```

Discussion

You specify the desired traits by combining the font trait mask values described in "Constants" (page 685) using the C bitwise OR operator.

Using `UnboldMask` returns `true` if the font is not bold, `false` otherwise. Using `UnitalicMask` returns `true` if the font is not italic, `false` otherwise.

isEnabled

Returns `true` if the font conversion system's user interface items (the Font panel and Font menu items) are enabled, `false` if they're not.

```
public boolean isEnabled()
```

See Also

`isEnabled` (page 690) (NSFontPanel)
`isEnabled` (page 1924) (NSMenuItem)
`setEnabled` (page 683)

isMultiple

Returns `true` if the last font selection recorded has multiple fonts, `false` if it's a single font.

```
public boolean isMultiple()
```

See Also

`setSelectedFont` (page 684)
`selectedFont` (page 682)

localizedNameForFamily

Returns a localized string with the name of the specified font *family* and *face* (for example, "Times" and "Roman"), if one exists.

```
public String localizedNameForFamily(String family, String face)
```

Discussion

The user's location is determined from the user's Languages default setting. The method also loads the localized strings for the font, if they aren't already loaded.

If *face* is null, this method returns the font family only.

modifyFont

This action method causes the receiver to send its action message up the responder chain.

```
public void modifyFont(Object sender)
```

Discussion

When a responder replies by providing a font to convert in a convertFont (page 674) message, the receiver converts the font in the manner specified by *sender*. The conversion is determined by sending a tag message to *sender* and invoking a corresponding method:

Sender's Tag	Method Used
NoFontChangeAction	None; the font is returned unchanged.
ViaPanelFontAction	The Font panel's panelConvertFont (page 690).
AddTraitFontAction	convertFontToHaveTrait (page 675).
RemoveTraitFontAction	convertFontToNotHaveTrait (page 676).
SizeUpFontAction	convertFontToSize (page 676).
SizeDownFontAction	convertFontToSize (page 676).
HeavierFontAction	convertWeight (page 677).
LighterFontAction	convertWeight (page 677).

See Also

addFontTrait (page 672)

removeFontTrait (page 682)

modifyFontViaPanel (page 680)

modifyFontViaPanel

This action method causes the receiver to send its action message up the responder chain.

```
public void modifyFontViaPanel(Object sender)
```

Discussion

When a responder replies by providing a font to convert in a `convertFont` (page 674) message, the receiver converts the font by sending a `panelConvertFont` (page 690) message to the Font panel. The panel in turn may send `convertFontToFamily` (page 675), `convertFontToHaveTrait` (page 675), and other specific conversion methods to make its change.

See Also

`addFontTrait` (page 672)

`removeFontTrait` (page 682)

`modifyFont` (page 680)

orderFrontFontPanel

This action method opens the Font panel by sending it an `orderFront` (page 1844) message, creating the Font panel if necessary.

```
public void orderFrontFontPanel(Object sender)
```

See Also

`fontPanel` (page 678)

orderFrontStylesPanel

This action method opens the Font styles panel.

```
public void orderFrontStylesPanel(Object sender)
```

Availability

Available in Mac OS X v10.3 and later.

removeCollection

Removes the collection specified by *collectionName*.

```
public boolean removeCollection(String collectionName)
```

Availability

Available in Mac OS X v10.3 and later.

See Also

`addCollection` (page 672)

removeFontDescriptor

Removes the font collection named *collectionName* from the collection specified by *collection*.

```
public void removeFontDescriptor(NSFontDescriptor descriptor, String collection)
```

Availability

Available in Mac OS X v10.3 and later.

See Also

addFontDescriptors (page 672)

removeFontTrait

This action method causes the receiver to send its action message up the responder chain.

```
public void removeFontTrait(Object sender)
```

Discussion

When a responder replies by providing a font to convert in a convertFont (page 674) message, the receiver converts the font by removing the trait specified by *sender*. This trait is determined by sending a tag message to *sender* and interpreting it as a font trait mask for a convertFontToNotHaveTrait (page 676) message.

See Also

addFontTrait (page 672)

modifyFont (page 680)

modifyFontViaPanel (page 680)

selectedFont

Returns the last font recorded with a setSelectedFont (page 684) message.

```
public NSFont selectedFont()
```

Discussion

While fonts are being converted in response to a convertFont (page 674) message, you can determine the font selected in the Font panel like this:

```
NSFontManager fontManager = NSFontManager.sharedFontManager();
panelFont = fontManager.convertFont(fontManager.selectedFont());
```

See Also

isMultiple (page 679)

sendAction

Sends the receiver's action message up the responder chain, initiating a font change for whatever conversion and trait to change were last requested.

```
public boolean sendAction()
```

Discussion

Returns true if some object handled the message, false if the message went unheard.

This method is used internally by the font conversion system. You should never need to invoke it directly. Instead, use the action methods such as addFontTrait (page 672) or modifyFontViaPanel (page 680).

See Also

setAction (page 683)

setAction

Sets the action that's sent to the first responder when the user selects a new font from the Font panel or chooses a command from the Font menu to *aSelector*.

```
public void setAction(NSSelector aSelector)
```

Discussion

You should rarely need to change this setting.

See Also

`action` (page 672)

setDelegate

Sets the receiver's delegate to *anObject*.

```
public void setDelegate(Object anObject)
```

See Also

`delegate` (page 678)

setEnabled

Controls whether the font conversion system's user interface items (the Font panel and Font menu items) are enabled.

```
public void setEnabled(boolean flag)
```

Discussion

If *flag* is `true` they're enabled; if *flag* is `false` they're disabled.

See Also

`setEnabled` (page 691) (NSFontPanel)

`isEnabled` (page 679)

setFontMenu

Records *aMenu* as the application's Font menu.

```
public void setFontMenu(NSMenu aMenu)
```

See Also

`fontMenu` (page 678)

setSelectedAttributes

Informs the paragraph and character formatting panels when text in a selection has changed attributes.

```
public void setSelectedAttributes(NSDictionary attributes, boolean flag)
```

Discussion

flag is used to inform the panel that multiple fonts or attributes are enclosed within the selection. Used primarily by NSTextView.

Availability

Available in Mac OS X v10.3 and later.

See Also

`convertAttributes` (page 674)

setSelectedFont

Records *aFont* as the currently selected font and updates the Font panel to reflect this.

```
public void setSelectedFont(NSFont aFont, boolean flag)
```

Discussion

If *flag* is `true`, the Font panel indicates that more than one font is contained in the selection.

An object that manipulates fonts should invoke this method whenever it becomes first responder and whenever its selection changes. After all fonts have been converted, the font manager itself records the new selected font.

See Also

`selectedFont` (page 682)

`isMultiple` (page 679)

traitsOfFont

Returns the traits of *aFont*.

```
public int traitsOfFont(NSFont aFont)
```

Discussion

The traits are returned as a mask created by combining these options with the C bitwise OR operator.

Font trait mask values are listed in "Constants" (page 685).

weightOfFont

Returns a rough numeric measure the weight of *aFont*, where 0 indicates the lightest possible weight, 5 indicates a normal or book weight, and 9 or more indicates a bold or heavier weight.

```
public int weightOfFont(NSFont aFont)
```

Constants

NSFontManager categorizes fonts according to a small set of traits. You can convert fonts by adding and removing individual traits, and you can get a font with a specific combination of traits. The traits defined and available for your use are:

`BoldMask`
`CompressedMask`
`CondensedMask`
`ExpandedMask`
`FixedPitchMask`
`ItalicMask`
`NarrowMask`
`NonstandardCharacterSetMask`
`PosterMask`
`SmallCapsMask`
`UnboldMask`
`UnitalicMask`

These pairs of traits are mutually exclusive:

`CondensedMask` and `ExpandedMask`
`BoldMask` and `UnboldMask`
`ItalicMask` and `UnitalicMask`

This constant specifies options accepted by `addCollection` (page 672):

`FontCollectionApplicationOnlyMask`

Delegate Methods

fontManagerWillIncludeFont

Requests permission from the delegate to display *fontName* in the Font panel.

```
public abstract boolean fontManagerWillIncludeFont(Object theFontManager, String
    fontName)
```

Discussion

fontName is the full PostScript name of the font, such as Helvetica-BoldOblique or Helvetica-Narrow-Bold. If the delegate returns `true`, *fontName* is listed in *theFontManager*; if the delegate returns `false`, it isn’t.

Important: This delegate method is not called in Mac OS X versions 10.3 and 10.4.

This method is invoked repeatedly as necessary whenever the Font panel needs updating, such as when the Font panel is first loaded, and when the user selects a family name to see which typefaces in that family are available. Your implementation should execute fairly quickly to ensure the responsiveness of the Font panel.

NSFontPanel

Inherits from	NSPanel : NSWindow : NSResponder : NSObject
Implements	NSCoding (NSResponder)
Package:	com.apple.cocoa.application
Companion guide	Font Panel

Overview

The NSFontPanel class implements the Font panel—a user interface object that displays a list of available fonts, letting the user preview them and change the font used to display text. The actual changes are made through conversion messages sent to the shared NSFontManager instance. There's only one Font panel for each application.

Tasks

Constructors

`NSFontPanel` (page 688)
Creates an empty NSFontPanel.

Getting the Font Panel

`sharedFontPanel` (page 689)
Returns the single NSFontPanel instance for the application, creating it if necessary.

`sharedFontPanelExists` (page 689)
Returns `true` if the shared Font panel has been created, `false` if it hasn't.

Enabling Font Changes

`setEnabled` (page 691)
Controls whether the receiver's Set button is enabled.

`isEnabled` (page 690)
Returns `true` if the receiver's Set button is enabled, `false` if it isn't.

Updating the Font Panel

`setPanelFont` (page 691)
: Sets the selected font in the receiver to *aFont* if *flag* is `false`; otherwise selects no font and displays a message in the preview area indicating that multiple fonts are selected.

Converting Fonts

`panelConvertFont` (page 690)
: Converts *aFont* using the settings in the receiver, with the aid of the shared NSFontManager if necessary, and returns the converted font.

Working in Modal Loops

`worksWhenModal` (page 691)
: Returns `true`, regardless of the setting established using the NSPanel method `setWorksWhenModal` (page 1054).

Setting an Accessory View

`setAccessoryView` (page 690)
: Establishes *aView* as the receiver's accessory view, allowing you to add custom controls to your application's Font panel without having to create a subclass.

`accessoryView` (page 690)
: Returns the receiver's accessory view.

Constructors

NSFontPanel

Creates an empty NSFontPanel.

```
public NSFontPanel()
```

Creates a new NSFontPanel.

```
public NSFontPanel(NSRect contentRect, int styleMask, int backingType, boolean
    defer)
```

Discussion

The *contentRect* argument specifies the location and size of the panel's content area in screen coordinates. Note that the window server limits window position coordinates to ±16,000 and sizes to 10,000.

The *styleMask* argument specifies the panel's style. Either it can be `NSWindow.BorderlessWindowMask`, or it can contain any of the options described in NSWindow's "Constants" (page 1875), combined using the C bitwise OR operator.

Borderless windows display none of the usual peripheral elements and are generally useful only for display or caching purposes; you should normally not need to create them. Also, note that an NSWindow's style mask should include `NSWindow.TitledWindowMask` if it includes any of the others.

The *backingType* argument specifies how the drawing done in the panel is buffered by the object's window device, and possible values are described in NSWindow's "Constants" (page 1875).

The *defer* argument determines whether the window server creates a window device for the new panel immediately. If *defer* is `true`, it defers creating the window until the panel is moved onscreen. All display messages sent are postponed until the panel is created, just before it's moved onscreen. Deferring the creation of the window improves launch time and minimizes the virtual memory load on the window server.

The new panel creates an instance of NSView to be its default content view. You can replace it with your own object by using the `setContentView` (page 1858) method.

Creates a new NSFontPanel.

```
public NSFontPanel(NSRect contentRect, int styleMask, int backingType, boolean
    defer, NSScreen aScreen)
```

Discussion
This constructor is equivalent to the preceding one, except *contentRect* is specified relative to the lower-left corner of *aScreen*.

If *aScreen* is `null`, *contentRect* 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.

Static Methods

sharedFontPanel

Returns the single NSFontPanel instance for the application, creating it if necessary.

```
public static NSFontPanel sharedFontPanel()
```

See Also
`sharedFontPanelExists` (page 689)

sharedFontPanelExists

Returns `true` if the shared Font panel has been created, `false` if it hasn't.

```
public static boolean sharedFontPanelExists()
```

See Also
`sharedFontPanel` (page 689)

Instance Methods

accessoryView

Returns the receiver's accessory view.

```
public NSView accessoryView()
```

See Also
`setAccessoryView` (page 690)

isEnabled

Returns `true` if the receiver's Set button is enabled, `false` if it isn't.

```
public boolean isEnabled()
```

Discussion
The receiver continues to reflect the font of the selection for cooperating text objects regardless of this setting.

See Also
`setEnabled` (page 691)

panelConvertFont

Converts *aFont* using the settings in the receiver, with the aid of the shared NSFontManager if necessary, and returns the converted font.

```
public NSFont panelConvertFont(NSFont aFont)
```

Discussion
If *aFont* can't be converted it's returned unchanged.

For example, if *aFont* is Helvetica Oblique 12.0 point and the user has selected the Times font family (and nothing else) in the Font panel, the font returned is Times Italic 12.0 point.

See Also
`convertFont` (page 674) (NSFontManager)

setAccessoryView

Establishes *aView* as the receiver's accessory view, allowing you to add custom controls to your application's Font panel without having to create a subclass.

```
public void setAccessoryView(NSView aView)
```

See Also
`accessoryView` (page 690)

setEnabled

Controls whether the receiver's Set button is enabled.

```
public void setEnabled(boolean flag)
```

Discussion

If *flag* is `true` the Set button is enabled; if *flag* is `false` it's disabled. The receiver continues to reflect the font of the selection for cooperating text objects regardless of this setting.

See Also

`isEnabled` (page 690)

setPanelFont

Sets the selected font in the receiver to *aFont* if *flag* is `false`; otherwise selects no font and displays a message in the preview area indicating that multiple fonts are selected.

```
public void setPanelFont(NSFont aFont, boolean flag)
```

Discussion

You normally don't use this method directly; instead, you send `setSelectedFont` (page 684) to the shared NSFontManager, which in turn invokes this method.

worksWhenModal

Returns `true`, regardless of the setting established using the NSPanel method `setWorksWhenModal` (page 1054).

```
public boolean worksWhenModal()
```

Discussion

This method allows fonts to be changed in modal windows and panels.

See Also

`worksWhenModal` (page 1874) (NSWindow)

`worksWhenModal` (page 1054) (NSPanel)

Constants

The constants in the following table specify the available font panel mode masks. They are available only on Mac OS X v10.3 and later.

Constant	Description
`FaceModeMask`	Display the typeface column.
`SizeModeMask`	Display the font size column.
`CollectionModeMask`	Display the font collections column.

Constant	Description
StandardModesMask	Display the standard default font panel—that is, including the collections, typeface, and size columns.
AllModesMask	Display all the available adornments.

NSForm

Inherits from NSMatrix : NSControl : NSView : NSResponder : NSObject

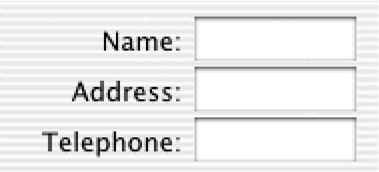
Implements NSCoding (NSResponder)

Package: com.apple.cocoa.application

Companion guides Forms
Matrix Programming Guide for Cocoa

Overview

An NSForm is a vertical NSMatrix of NSFormCells. Here's an example:



NSForm uses NSFormCell (page 701) to implement its user interface.

Tasks

Constructors

`NSForm` (page 695)
Creates an empty NSForm with a zero-sized frame rectangle.

Adding and Removing Entries

`addEntry` (page 695)
Adds a new entry to the end of the receiver and gives it the title *`title`*.

`insertEntryAtIndex` (page 696)
Inserts an entry with the title *`title`* at the position in the receiver specified by *`entryIndex`*.

`removeEntryAtIndex` (page 697)
Removes the entry at *`entryIndex`*.

Changing the Appearance of All the Entries

`setBezeled` (page 697)

`setBordered` (page 697)
Sets whether the entries in the receiver display a border—that is, a thin line—around their editable text fields.

`setEntryWidth` (page 698)
Sets the width (in pixels) of all the entries in the receiver.

`setFrameSize` (page 698)
Sets the receiver's frame size to be *`newSize`*.

`setInterlineSpacing` (page 698)
Sets the number of pixels between entries in the receiver to *`spacing`*.

`setTitleAlignment` (page 698)
Sets the alignment for all of the entry titles.

`setTextAlignment` (page 698)
Sets the alignment for all of the receiver's editable text.

`setTitleFont` (page 699)
Sets the font for all of the entry titles to *`font`*.

`setTextFont` (page 698)
Sets the font for all of the receiver's editable text fields to *`font`*.

Getting Cells and Indices

`indexOfCellWithTag` (page 696)
Returns the index of the entry whose tag is *`tag`*.

`indexOfSelectedItem` (page 696)
Returns the index of the selected entry.

`cellAtIndex` (page 696)
Returns the entry specified by *`entryIndex`*.

Displaying a Cell

`drawCellAtIndex` (page 696)
Displays the entry specified by *`entryIndex`*.

Editing Text

`selectTextAtIndex` (page 697)
Selects the entry at *`entryIndex`*.

Constructors

NSForm

Creates an empty NSForm with a zero-sized frame rectangle.

```
public NSForm()
```

Creates an NSForm with *frameRect* as its frame rectangle.

```
public NSForm(NSRect frameRect)
```

Creates an NSForm, in the frame specified by *frameRect*.

```
public NSForm(NSRect frameRect, int mode, NSCell aCell, int numRows, int numColumns)
```

Discussion
The new NSForm contains *numRows* rows and *numColumns* columns. *mode* is set as the tracking mode for the NSForm and can be one of the modes described in NSMatrix's "Constants" (page 908).

The new form creates cells by copying *aCell*, which should be an instance of a subclass of NSCell.

Creates an NSForm, in the frame specified by *frameRect*.

```
public NSForm(NSRect frameRect, int mode, Class aClass, int numRows, int numColumns)
```

Discussion
The new NSForm contains *numRows* rows and *numColumns* columns. *mode* is set as the tracking mode for the NSForm and can be one of the modes described in NSMatrix's "Constants" (page 908).

The new form creates and uses cells of class *aClass*.

Instance Methods

addEntry

Adds a new entry to the end of the receiver and gives it the title *title*.

```
public NSFormCell addEntry(String title)
```

Discussion
The new entry has no tag, target, or action, but is enabled and editable.

See Also
insertEntryAtIndex (page 696)
setEditable (page 323) (NSCell)
setTag (page 51) (NSActionCell)
setTarget (page 51) (NSActionCell)
setAction (page 48) (NSActionCell)
setEnabled (page 49) (NSActionCell)

cellAtIndex

Returns the entry specified by *entryIndex*.

```
public NSCell cellAtIndex(int entryIndex)
```

See Also

indexOfCellWithTag (page 696)

indexOfSelectedItem (page 696)

drawCellAtIndex

Displays the entry specified by *entryIndex*.

```
public void drawCellAtIndex(int entryIndex)
```

Discussion

Because this method is called automatically whenever a cell needs drawing, you never need to invoke it explicitly. It is included in the API so you can override it if you subclass NSFormCell.

See Also

indexOfCellWithTag (page 696)

indexOfSelectedItem (page 696)

indexOfCellWithTag

Returns the index of the entry whose tag is *tag*.

```
public int indexOfCellWithTag(int tag)
```

See Also

tag (page 334) (NSCell)

indexOfSelectedItem

Returns the index of the selected entry.

```
public int indexOfSelectedItem()
```

Discussion

If no entry is selected, `indexOfSelectedItem` returns –1.

insertEntryAtIndex

Inserts an entry with the title *title* at the position in the receiver specified by *entryIndex*.

```
public NSFormCell insertEntryAtIndex(String title, int entryIndex)
```

Discussion

The new entry has no tag, target, or action, and, as explained in the class description, it won’t appear on the screen automatically.

Returns the newly inserted NSFormCell.

See Also
`addEntry` (page 695)
`removeEntryAtIndex` (page 697)

removeEntryAtIndex

Removes the entry at *entryIndex*.

```
public void removeEntryAtIndex(int entryIndex)
```

Discussion
If *entryIndex* is not a valid position in the receiver, does nothing.

selectTextAtIndex

Selects the entry at *entryIndex*.

```
public void selectTextAtIndex(int entryIndex)
```

Discussion
If *entryIndex* is not a valid position in the receiver, does nothing.

setBezeled

```
public void setBezeled(boolean flag)
```

Discussion
If *flag* is `true`, sets all the entries in the receiver to show a bezel around their editable text; if *flag* is `false`, sets all the entries to show no bezel.

See Also
`setBordered` (page 697)
`isBezeled` (page 314) (NSCell)

setBordered

Sets whether the entries in the receiver display a border—that is, a thin line—around their editable text fields.

```
public void setBordered(boolean flag)
```

Discussion
If *flag* is `true`, they display a border; otherwise, they don't. An entry can have a border or a bezel, but not both.

See Also
`setBezeled` (page 697)
`isBordered` (page 314) (NSCell)

setEntryWidth

Sets the width (in pixels) of all the entries in the receiver.

```
public void setEntryWidth(float width)
```

Discussion

This *width* includes both the title and the text field.

setFrameSize

Sets the receiver’s frame size to be *newSize*.

```
public void setFrameSize(NSSize newSize)
```

Discussion

The width of NSFormCells always match the width of the NSForm. The cell width is always changed to match the view regardless of the value returned by `autosizesCells` (page 884).

setInterlineSpacing

Sets the number of pixels between entries in the receiver to *spacing*.

```
public void setInterlineSpacing(float spacing)
```

setTextAlignment

Sets the alignment for all of the receiver’s editable text.

```
public void setTextAlignment(int alignment)
```

Discussion

alignment can be one of three constants: `NSText.RightTextAlignment`, `NSText.CenterTextAlignment`, or `NSText.LeftTextAlignment` (the default).

See Also

`setTitleAlignment` (page 698)

setTextFont

Sets the font for all of the receiver’s editable text fields to *font*.

```
public void setTextFont(NSFont font)
```

See Also

`setTextFont` (page 698)

setTitleAlignment

Sets the alignment for all of the entry titles.

```
public void setTitleAlignment(int alignment)
```

Discussion

alignment can be one of three constants: `NSText.RightTextAlignment`, `NSText.CenterTextAlignment`, or the default, `NSText.LeftTextAlignment`.

See Also

`setTextAlignment` (page 698)

setTitleFont

Sets the font for all of the entry titles to *font*.

```
public void setTitleFont(NSFont font)
```

See Also

`setTextFont` (page 698)

NSFormCell

Inherits from	NSActionCell : NSCell : NSObject
Implements	NSCoding (NSCell)
Package:	com.apple.cocoa.application
Companion guide	Forms

Overview

The NSFormCell class is used to implement text entry fields in a form. The left part of an NSFormCell is a title. The right part is an editable text entry field.

NSFormCell implements the user interface of NSForm (page 693).

Tasks

Constructors

`NSFormCell` (page 702)
Creates an empty NSFormCell.

Asking About a Cell's Appearance

`isOpaque` (page 703)
Returns `true` if the title is empty and an opaque bezel is set, otherwise `false` is returned.

Asking About a Cell's Title

`attributedTitle` (page 703)
Returns the title as an attributed string.

`title` (page 706)
Returns the receiver's title.

`titleAlignment` (page 706)
Returns the alignment of the title.

`titleFont` (page 706)
Returns the font used to draw the receiver's title.

`titleWidth` (page 706)
Returns the width (in pixels) of the title field.

`titleWidthWithSize` (page 706)
Returns the width (in pixels) of the title field.

Changing the Cell's Title

`setAttributedTitle` (page 704)
Sets the receiver's title and title attributes according to *anAttributedString*.

`setTitle` (page 705)
Sets the receiver's title to *aString*.

`setTitleAlignment` (page 705)
Sets the alignment of the title.

`setTitleFont` (page 705)
Sets the title's font to *font*.

`setTitleWidth` (page 705)
Sets the width in pixels.

Setting a Keyboard Equivalent

`setTitleWithMnemonic` (page 705)
Sets the cell title and a single mnemonic character.

Asking About Placeholder Values

`placeholderAttributedString` (page 703)
Returns the cell's attributed placeholder string.

`placeholderString` (page 704)
Returns the cell's plain text placeholder string.

`setPlaceholderAttributedString` (page 704)
Sets the placeholder text for the cell as an attributed string.

`setPlaceholderString` (page 704)
Sets the placeholder text for the cell as a plain text string.

Constructors

NSFormCell

Creates an empty NSFormCell.

```
public NSFormCell()
```

Creates a new NSFormCell.

```
public NSFormCell(String aString)
```

Discussion
Its title is set to *aString*. The contents of its text entry field are set to the empty string (“”). 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.

Creates an NSFormCell initialized with *anImage* and set to have the cell’s default menu.

```
public NSFormCell(NSImage anImage)
```

Discussion
If *anImage* is `null`, no image is set.

Instance Methods

attributedTitle

Returns the title as an attributed string.

```
public NSAttributedString attributedTitle()
```

isOpaque

Returns `true` if the title is empty and an opaque bezel is set, otherwise `false` is returned.

```
public boolean isOpaque()
```

placeholderAttributedString

Returns the cell’s attributed placeholder string.

```
public NSAttributedString placeholderAttributedString()
```

Discussion
If this method returns `null`, you can also call `placeholderString` to see if the cell has a plain text placeholder string.

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

See Also
`placeholderString` (page 704)
`setPlaceholderAttributedString` (page 704)

placeholderString

Returns the cell's plain text placeholder string.

```
public String placeholderString()
```

Discussion

If this method returns `null`, you can also call `placeholderAttributedString` to see if the cell has an attributed placeholder string.

Availability

Available in Mac OS X v10.4 and later.

See Also

`placeholderAttributedString` (page 703)
`setPlaceholderString` (page 704)

setAttributedTitle

Sets the receiver's title and title attributes according to *anAttributedString*.

```
public void setAttributedTitle(NSAttributedString anAttributedString)
```

setPlaceholderAttributedString

Sets the placeholder text for the cell as an attributed string.

```
public void setPlaceholderAttributedString(NSAttributedString string)
```

Discussion

The string argument is represented by *string*.

Note that invoking this method clears out any plain text string set by `setPlaceholderString`.

Availability

Available in Mac OS X v10.4 and later.

See Also

`placeholderAttributedString` (page 703)
`setPlaceholderString` (page 704)

setPlaceholderString

Sets the placeholder text for the cell as a plain text string.

```
public void setPlaceholderString(String string)
```

Discussion

The text string is represented by the *string* argument.

Note that invoking this method clears out any attributed string set by `setPlaceholderAttributedString` (page 704).

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

See Also
`placeholderString` (page 704)
`setPlaceholderAttributedString` (page 704)

setTitle

Sets the receiver's title to *aString*.

```
public void setTitle(String aString)
```

setTitleAlignment

Sets the alignment of the title.

```
public void setTitleAlignment(int alignment)
```

Discussion
alignment can be one of three constants: `NSText.LeftTextAlignment`, `NSText.RightTextAlignment`, or `NSText.CenterTextAlignment`.

setTitleFont

Sets the title's font to *font*.

```
public void setTitleFont(NSFont font)
```

setTitleWidth

Sets the width in pixels.

```
public void setTitleWidth(float width)
```

Discussion
You usually won't need to invoke this method, because the Application Kit automatically sets the title width whenever the title changes. If, however, the automatic width doesn't suit your needs, you can use `setTitleWidth` to set the width explicitly.

Once you have set the width this way, the Application Kit stops setting the width automatically; you will need to invoke `setTitleWidth` every time the title changes. If you want the Application Kit to resume automatic width assignments, invoke `setTitleWidth` with a negative *width* value.

setTitleWithMnemonic

Sets the cell title and a single mnemonic character.

```
public void setTitleWithMnemonic(String titleWithAmpersand)
```

Discussion

Mnemonics are not supported in Mac OS X.

See Also

`setTitle` (page 705)

title

Returns the receiver's title.

```
public String title()
```

Discussion

The default title is "Field:".

titleAlignment

Returns the alignment of the title.

```
public int titleAlignment()
```

Discussion

The alignment can be one of the following: `NSText.LeftTextAlignment`, `NSText.CenterTextAlignment`, or `NSText.RightTextAlignment` (the default).

titleFont

Returns the font used to draw the receiver's title.

```
public NSFont titleFont()
```

titleWidth

Returns the width (in pixels) of the title field.

```
public float titleWidth()
```

Discussion

If you specified the width using `setTitleWidth` (page 705), this method returns the value you chose. Otherwise, it returns the width calculated automatically by the Application Kit.

See Also

`titleWidthWithSize` (page 706)

titleWidthWithSize

Returns the width (in pixels) of the title field.

```
public float titleWidthWithSize(NSSize aSize)
```

Discussion

If you specified the width using `setTitleWidth` (page 705), this method returns the value you chose. Otherwise, it calculates the width, constrained to *`aSize`*.

See Also

`titleWidth` (page 706)

NSGlyphInfo

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.application
Availability	Available in Mac OS X v10.2 and later.
Companion guide	Font Handling

Overview

NSGlyphInfo is a glyph attribute value (`GlyphInfoAttributeName`) in an attributed string. NSGlyphInfo allows you to override a font's specified mapping from Unicode to the glyph ID. Overriding the mapping allows you to specify a variant glyph for a given character if the font contains multiple variations for that character or to specify a glyph that doesn't have a standard mapping (such as some ligature glyphs).

Tasks

Constructors

`NSGlyphInfo` (page 710)
You should not call this constructor; use the class methods to create NSGlyphInfo objects instead.

Creating an NSGlyphInfo Object

`glyphInfoWithCharacterIdentifierInCollectionAndBaseString` (page 710)
Instantiates and returns an NSGlyphInfo object using a character identifier and a character collection.

`glyphInfoWithGlyphForFontAndBaseString` (page 710)
Instantiates and returns an NSGlyphInfo object using a glyph index and an NSFont.

`glyphInfoWithGlyphNameForFontAndBaseString` (page 711)
Instantiates and returns an NSGlyphInfo object using a glyph name and an NSFont.

Getting Information About an NSGlyphInfo Object

`characterCollection` (page 711)
: Returns a value specifying the glyph–to–character identifier mapping of the receiver, if the receiver was instantiated using `glyphInfoWithCharacterIdentifierInCollectionAndBaseString` (page 710).

`characterIdentifier` (page 711)
: Returns the receiver's character identifier (CID).

`glyphName` (page 712)
: Returns the receiver's glyph name.

Constructors

NSGlyphInfo

You should not call this constructor; use the class methods to create NSGlyphInfo objects instead.

```
public NSGlyphInfo()
```

Availability
Available in Mac OS X v10.2 and later.

Static Methods

glyphInfoWithCharacterIdentifierInCollectionAndBaseString

Instantiates and returns an NSGlyphInfo object using a character identifier and a character collection.

```
public static NSGlyphInfo
    glyphInfoWithCharacterIdentifierInCollectionAndBaseString(int cid, int
    characterCollection, String theString)
```

Discussion
theString is the part of the attributed string the returned instance is intended to override. Possible values for *characterCollection* are described in "Constants" (page 712).

Availability
Available in Mac OS X v10.2 and later.

glyphInfoWithGlyphForFontAndBaseString

Instantiates and returns an NSGlyphInfo object using a glyph index and an NSFont.

```
public static NSGlyphInfo glyphInfoWithGlyphForFontAndBaseString(int glyph, NSFont
     font, String theString)
```

Discussion
`theString` is the part of the attributed string the returned instance is intended to override.

Availability
Available in Mac OS X v10.2 and later.

glyphInfoWithGlyphNameForFontAndBaseString

Instantiates and returns an NSGlyphInfo object using a glyph name and an NSFont.

```
public static NSGlyphInfo glyphInfoWithGlyphNameForFontAndBaseString(String
    glyphName, NSFont font, String theString)
```

Discussion
`theString` is the part of the attributed string the returned instance is intended to override.

Availability
Available in Mac OS X v10.2 and later.

Instance Methods

characterCollection

Returns a value specifying the glyph–to–character identifier mapping of the receiver, if the receiver was instantiated using `glyphInfoWithCharacterIdentifierInCollectionAndBaseString` (page 710).

```
public int characterCollection()
```

Discussion
Returns `IdentityMappingCharacterCollection` if the receiver was instantiated a glyph name. Other possible return values are described in "Constants" (page 712).

Availability
Available in Mac OS X v10.2 and later.

characterIdentifier

Returns the receiver's character identifier (CID).

```
public int characterIdentifier()
```

Discussion
If the receiver was instantiated with a method other than `glyphInfoWithCharacterIdentifierInCollectionAndBaseString` (page 710), this method returns `NULL`.

Availability
Available in Mac OS X v10.2 and later.

glyphName

Returns the receiver's glyph name.

```
public String glyphName()
```

Discussion
If the receiver was instantiated with a method other than `glyphInfoWithGlyphNameForFontAndBaseString` (page 711), this method returns `null`.

Availability
Available in Mac OS X v10.2 and later.

Constants

The following values specify the mapping of character identifiers to glyphs, and is returned by `characterCollection` (page 711):

Constant	Description
`IdentityMappingCharacterCollection`	Indicates that the character identifier is equal to the glyph index.
`AdobeCNS1CharacterCollection`	Indicates the Adobe-CNS1 mapping.
`AdobeGB1CharacterCollection`	Indicates the Adobe-GB1 mapping.
`AdobeJapan1CharacterCollection`	Indicates the Adobe-Japan1 mapping.
`AdobeJapan2CharacterCollection`	Indicates the Adobe-Japan2 mapping.
`AdobeKorea1CharacterCollection`	Indicates the Adobe-Korea1 mapping.

NSGraphics

Inherits from NSObject
Package: com.apple.cocoa.application
Companion guide Basic Drawing

Overview

The NSGraphics class provides static methods for performing some basic drawing operations, view clipping, and inquiries about the application's windows.

Tasks

Constructors

`NSGraphics` (page 716)
Creates and returns an empty NSGraphics object.

Obtaining Device Information

`availableWindowDepths` (page 716)
Returns an array of values specifying which window depths are currently available.

`bestDepth` (page 716)
Returns a window depth deep enough for the given number of colors in *`colorSpace`*, bits per sample specified by *`bps`*, bits per pixel specified by *`bpp`*, and whether planar as specified by *`planar`*.

`bitsPerPixelFromDepth` (page 717)
Returns the number of bits per pixel for the window depth specified by *`depth`*.

`bitsPerSampleFromDepth` (page 717)
Returns the number of bits per sample (bits per pixel in each color component) for the window depth specified by *`depth`*.

`colorSpaceFromDepth` (page 718)
Returns the color space name for the specified *`depth`*.

`numberOfColorComponents` (page 725)
Returns the number of color components in the color space *`colorSpaceName`*.

`planarFromDepth` (page 725)
Returns `true` if the specified window *`depth`* is planar and `false` if it is not.

Working with NSAttributedStrings

`drawAttributedString` (page 720)
: Draws *attributedString* with its font and other display attributes at *aPoint* in the currently focused NSView.

`sizeOfAttributedString` (page 726)
: Returns the bounding box of the marks that *attributedString* draws.

Working with Windows

`convertGlobalToWindowNumber` (page 718)
: Deprecated.

`convertWindowNumberToGlobal` (page 718)
: Deprecated.

`windowCount` (page 726)
: Returns the number of onscreen windows belonging to the application.

`windowList` (page 726)
: Provides an ordered list of the application's onscreen windows.

Working with Bitmap Images

`copyBitmapFromGState` (page 719)
: Deprecated.

`copyBits` (page 719)
: Copies the pixels in the rectangle specified by *srcRect* to the location specified by *destPoint*.

`drawBitmap` (page 720)
: Renders a bitmap image using an appropriate display operator.

`readPixel` (page 726)
: Returns the color of the pixel at the location specified by *aPoint*.

Filling a List of Rectangles

`fillRectList` (page 724)
: Fills a list of rectangles, *rectangles*, with the current color.

`fillRectListInRange` (page 724)
: Fills a subset *aRange* of rectangles from the array *rectangles* with the current color.

`fillRectListWithColors` (page 724)
: Takes a list of rectangles, *rectangles*, and a matching list of color values.

`fillRectListWithColorsInRange` (page 724)
: Fills a subset *aRange* of rectangles from the array *rectangles* with the corresponding subset colors from the array *colors*.

Clipping a List of Rectangles

`clipRectList` (page 717)
: Takes an array of rectangles, *rectangles*, constructs a path that is the graphic union of those rectangles, and intersects that path with the current clipping path.

`clipRectListInRange` (page 718)
: Extracts the subset *aRange* from the array of rectangles, *rectangles*, constructs a path that is the graphic union of those rectangles, and intersects that path with the current clipping path.

Working with Frame Rects

`frameRect` (page 725)
: Draws a frame around the inside of *boundsRect* in the current color.

`frameRectWithWidth` (page 725)
: Draws a frame around the inside of *boundsRect* in the current color.

`frameRectWithWidthUsingOperation` (page 725)
: Draws a frame around the inside of *boundsRect* in the current color, using the compositing operation *operation*.

Drawing

`dottedFrameRect` (page 719)
: Deprecated.

`drawButton` (page 721)
: Draws a gray-filled rectangle, used to signify a user interface button.

`drawColorTiledRects` (page 722)
: Draws rectangles with colored borders.

`drawDarkBezel` (page 722)
: Draws a dark gray–filled rectangle with a bezel border.

`drawGrayBezel` (page 722)
: Draws a gray-filled rectangle with a bezel border.

`drawGroove` (page 723)
: Draws a gray-filled rectangle with a groove border.

`drawLightBezel` (page 723)
: Draws a light gray–filled rectangle with a bezel border.

`drawWhiteBezel` (page 723)
: Draws a white-filled rectangle with a bezel border.

`drawWindowBackground` (page 723)
: Draws the window's default background pattern into the rectangle *aRect* of the currently focused view.

`eraseRect` (page 724)
: Erases the rectangle referred to by *boundsRect*, filling it with white.

`highlightRect` (page 725)
: Deprecated.

Focus Rings

`setFocusRingStyle` (page 726)
Sets the style of the focus ring in the current graphics context in the current locked focus view.

Updating Screen

`disableScreenUpdates` (page 719)
Disables screen updates.

`enableScreenUpdates` (page 723)
Enables screen updates.

Constructors

NSGraphics

Creates and returns an empty NSGraphics object.

```
public NSGraphics()
```

Discussion
All of the NSGraphic methods are static, so there is no need to create instances of the class.

Static Methods

availableWindowDepths

Returns an array of values specifying which window depths are currently available.

```
public static int[] availableWindowDepths()
```

See Also
`bitsPerPixelFromDepth` (page 717)
`bitsPerSampleFromDepth` (page 717)
`colorSpaceFromDepth` (page 718)
`planarFromDepth` (page 725)

bestDepth

Returns a window depth deep enough for the given number of colors in *colorSpace*, bits per sample specified by *bps*, bits per pixel specified by *bpp*, and whether planar as specified by *planar*.

```
public static int bestDepth(String colorSpace, int bps, int bpp, boolean planar)
```

Discussion

This method tries to accommodate all the parameters (match or better); if there are multiple matches, it gives the closest, with matching *colorSpace* first, then *bps*, then *planar*, then *bpp*. *bpp* is "bits per pixel"; 0 indicates default (same as the number of bits per plane, either *bps* or *bps* * `numberOfColorComponents` (page 725)); other values may be used as hints to provide backing stores of different configuration—for instance, 8-bit color.

See Also

`bitsPerPixelFromDepth` (page 717)
`bitsPerSampleFromDepth` (page 717)
`colorSpaceFromDepth` (page 718)
`planarFromDepth` (page 725)

bitsPerPixelFromDepth

Returns the number of bits per pixel for the window depth specified by *depth*.

```
public static int bitsPerPixelFromDepth(int depth)
```

See Also

`availableWindowDepths` (page 716)
`bestDepth` (page 716)

bitsPerSampleFromDepth

Returns the number of bits per sample (bits per pixel in each color component) for the window depth specified by *depth*.

```
public static int bitsPerSampleFromDepth(int depth)
```

See Also

`availableWindowDepths` (page 716)
`bestDepth` (page 716)

clipRectList

Takes an array of rectangles, *rectangles*, constructs a path that is the graphic union of those rectangles, and intersects that path with the current clipping path.

```
public static void clipRectList(NSRect[] rectangles)
```

Discussion

After computing the new clipping path, the current path is reset to empty.

See Also

`clipRectListInRange` (page 718)

clipRectListInRange

Extracts the subset *aRange* from the array of rectangles, *rectangles*, constructs a path that is the graphic union of those rectangles, and intersects that path with the current clipping path.

```
public static void clipRectListInRange(NSRect[] rectangles, NSRange aRange)
```

Discussion

After computing the new clipping path, the current path is reset to empty.

See Also

`clipRectList` (page 717)

colorSpaceFromDepth

Returns the color space name for the specified *depth*.

```
public static String colorSpaceFromDepth(int depth)
```

Discussion

For example, the returned color space name can be `CalibratedRGBColorSpace` or `DeviceCMYKColorSpace`.

See Also

`availableWindowDepths` (page 716)

`bestDepth` (page 716)

convertGlobalToWindowNumber

Deprecated.

```
public static int convertGlobalToWindowNumber(int globalNum)
```

Discussion

Local and global window numbers are identical, so this method returns *globalNum*.

See Also

`convertWindowNumberToGlobal` (page 718)

convertWindowNumberToGlobal

Deprecated.

```
public static int convertWindowNumberToGlobal(int winNum)
```

Discussion

Local and global window numbers are identical, so this method returns *winNum*.

See Also

`convertGlobalToWindowNumber` (page 718)

copyBitmapFromGState

Deprecated.

```
public static void copyBitmapFromGState(int srcGState, NSRect srcRect, NSRect
    destRect)
```

copyBits

Copies the pixels in the rectangle specified by *srcRect* to the location specified by *destPoint*.

```
public static void copyBits(int srcGState, NSRect srcRect, NSPoint destPoint)
```

Discussion

The source rectangle is defined in the graphics state designated by *srcGState*. If *srcGState* is NSNull, the current graphics state is assumed. The *destPoint* destination is defined in the current graphics state.

disableScreenUpdates

Disables screen updates.

```
public static void disableScreenUpdates()
```

Discussion

Prevents drawing operations from being flushed to the screen for all windows belonging to the calling process. When you reenable screen updates (with enableScreenUpdates (page 723)) screen flushing for all windows belonging to the calling process appears to be simultaneous. You typically call this method so that operations on multiple windows appear atomic to the user. This is a technique particularly useful for synchronizing parent and child windows. Make sure that the period after calling this method and before reenabling updates is short; the system only allow updating to be disabled for a limited time (currently one second) before automatically reenabling updates. Multiple disableScreenUpdates calls stack and are popped off the stack by matching enableScreenUpdates calls.

Availability

Available in Mac OS X v10.3 and later.

See Also

enableScreenUpdates (page 723)

dottedFrameRect

Deprecated.

```
public static void dottedFrameRect(NSRect aRect)
```

Discussion

Use a dashed NSBezierPath instead.

drawAttributedString

Draws *attributedString* with its font and other display attributes at *aPoint* in the currently focused NSView.

```
public static void drawAttributedString(NSAttributedString attributedString, NSPoint
      aPoint)
```

Discussion

The width (height for vertical layout) of the rendering area is unlimited, unlike the two parameter version of this method, which uses a bounding rectangle. As a result, this method renders the text in a single line.

If the focus view is flipped, the origin is set at the upper-left corner of the drawing bounding box; otherwise the origin is set at the lower-left corner.

Don't invoke this method while no NSView is focused.

See Also

`sizeOfAttributedString` (page 726)

`lockFocus` (page 1759) (NSView)

Draws *attributedString* with its font and other display attributes within *aRect* in the currently focused NSView, clipping the drawing to this rectangle.

```
public static void drawAttributedString(NSAttributedString attributedString, NSRect
      aRect)
```

Discussion

Text is drawn within *aRect* according to its line sweep direction; for example, Arabic text will begin at the right edge and potentially be clipped on the left.

If the focus view is flipped, the origin is set at the upper-left corner of the drawing bounding box; otherwise the origin is set at the lower-left corner.

Don't invoke this method while no NSView is focused.

See Also

`lockFocus` (page 1759) (NSView)

drawBitmap

Renders a bitmap image using an appropriate display operator.

```
public static void drawBitmap(NSRect rect, int pixelsWide, int pixelsHigh, int
    bitsPerSample, int samplesPerPixel, int bitsPerPixel, int bytesPerRow, boolean
      isPlanar, boolean hasAlpha, String colorSpace, byte[] data)
```

Discussion

The image is put in the rectangular area specified by *rect*, which is specified in the current coordinate system and is located in the current window. *pixelsWide* and *pixelsHigh* give the width and height of the image in pixels. If either of these dimensions is larger or smaller than the corresponding dimension of the destination rectangle, the image is scaled to fit.

The *bitsPerSample* argument is the number of bits per sample for each pixel, and *samplesPerPixel* is the number of samples per pixel. *bitsPerPixel* is based on *samplesPerPixel* and the configuration of the bitmap: if the configuration is planar, then the value of *bitsPerPixel* should equal the value of *bitsPerSample*; if the configuration isn't planar (is meshed instead), *bitsPerPixel* should equal *bitsPerSample* * *samplesPerPixel*.

The *bytesPerRow* argument is calculated in one of two ways, depending on the configuration of the image data (data configuration is described below). If the data is planar, *bytesPerRow* is (7 + (*pixelsWide* * *bitsPerSample*)) / 8. If the data is meshed, *bytesPerRow* is (7 + (*pixelsWide* * *bitsPerSample* * *samplesPerPixel*)) / 8.

A sample is data that describes one component of a pixel. In an RGB color system, the red, green, and blue components of a color are specified as separate samples, as are the cyan, magenta, yellow, and black components in a CMYK system. Color values in grayscale are a single sample. Alpha values that determine transparency and opaqueness are specified as a coverage sample separate from color. In bitmap images with alpha, the color (or gray) components have to be premultiplied with the alpha. Images with alpha are displayed, read back, and stored in TIFFs in this way.

The *isPlanar* argument refers to the way data is configured in the bitmap. This flag should be set `true` if a separate data channel is used for each sample. The function provides for up to five channels, `data1`, `data2`, `data3`, `data4`, and `data5`. It should be set `false` if sample values are interwoven in a single channel (meshed); all values for one pixel are specified before values for the next pixel.

Grayscale windows store pixel data in planar configuration; color windows store it in meshed configuration. This method can render meshed data in a planar window, or planar data in a meshed window. However, it's more efficient if the image has a depth (*bitsPerSample*) and configuration (*isPlanar*) that matches the window.

The *hasAlpha* argument indicates whether the image contains alpha. If it does, the number of samples should be one greater than the number of color components in the model (for example, four for RGB).

The *colorSpace* argument can be `CustomColorSpace`, indicating that the image data is to be interpreted according to the current color space in the graphics state. This interpretation allows for imaging using custom color spaces. The image parameters supplied as the other arguments should match what the color space is expecting.

If the image data is planar, *data*[0] through *data*[*samplesPerPixel*–1] point to the planes; if the data is meshed, only *data*[0] needs to be set.

drawButton

Draws a gray-filled rectangle, used to signify a user interface button.

```
public static void drawButton(NSRect boundsRect, NSRect clipRect)
```

Discussion

For an Aqua button, use an NSButton object.

The *boundsRect* argument specifies the rectangle within which the border is to be drawn in the current coordinate system. Since this method is often used to draw the border of a view, this rectangle is typically that view's bounds rectangle. Only those parts of *boundsRect* that lie within the *clipRect*, a clipping rectangle, are drawn.

The rectangle is filled with light gray. This method is designed for rectangles that are defined in unscaled, unrotated coordinate systems (that is, where the y axis is vertical, the x axis is horizontal, and a unit along either axis is equal to 1 screen pixel). The coordinate system can be either flipped or unflipped. The sides of the rectangle should lie on pixel boundaries.

drawColorTiledRects

Draws rectangles with colored borders.

```
public static void drawColorTiledRects(NSRect boundsRect, NSRect clipRect, int[]
    sides, NSColor[] colors, NSRange aRange)
```

Discussion

The `drawColorTiledRects` method is a generic method that can be used to draw different types of borders. These borders can be used to outline an area or to give rectangles the effect of being recessed from or elevated above the surface of the screen.

The *boundsRect* argument specifies the rectangle within which the border is to be drawn in the current coordinate system. Since this function is often used to draw the border of a view, this rectangle will typically be that view's bounds rectangle. Only those parts of *aRect* that lie within *clipRect*, a clipping rectangle, are drawn.

In addition to its *boundsRect* and *clipRect* arguments, `drawColorTiledRects` takes three more arguments, which determine how thick the border is and what colors are used to form it. `drawColorTiledRects` takes successive 1.0-unit-wide slices from the sides of the rectangle specified by the *sides* argument. Each slice is then drawn using the corresponding color from *colors*. `drawColorTiledRects` makes and draws these slices for each pair of elements in *sides* and *colors* within the range *aRange*. If a side is used more than once, the second slice is made inside the first.

drawDarkBezel

Draws a dark gray–filled rectangle with a bezel border.

```
public static void drawDarkBezel(NSRect boundsRect, NSRect clipRect)
```

Discussion

The *boundsRect* argument specifies the rectangle within which the border is to be drawn in the current coordinate system. Only those parts of *boundsRect* that lie within *clipRect*, a clipping rectangle, are drawn.

drawGrayBezel

Draws a gray-filled rectangle with a bezel border.

```
public static void drawGrayBezel(NSRect boundsRect, NSRect clipRect)
```

Discussion

The *boundsRect* argument specifies the rectangle within which the border is to be drawn in the current coordinate system. Only those parts of *boundsRect* that lie within *clipRect*, a clipping rectangle, are drawn.

drawGroove

Draws a gray-filled rectangle with a groove border.

```
public static void drawGroove(NSRect boundsRect, NSRect clipRect)
```

Discussion

The *boundsRect* argument specifies the rectangle within which the border is to be drawn in the current coordinate system. Only those parts of *boundsRect* that lie within *clipRect*, a clipping rectangle, are drawn.

drawLightBezel

Draws a light gray–filled rectangle with a bezel border.

```
public static void drawLightBezel(NSRect boundsRect, NSRect clipRect)
```

Discussion

The *boundsRect* argument specifies the rectangle within which the border is to be drawn in the current coordinate system. Only those parts of *boundsRect* that lie within *clipRect*, a clipping rectangle, are drawn.

drawWhiteBezel

Draws a white-filled rectangle with a bezel border.

```
public static void drawWhiteBezel(NSRect boundsRect, NSRect clipRect)
```

Discussion

The *boundsRect* argument specifies the rectangle within which the border is to be drawn in the current coordinate system. Only those parts of *boundsRect* that lie within *clipRect*, a clipping rectangle, are drawn.

drawWindowBackground

Draws the window's default background pattern into the rectangle *aRect* of the currently focused view.

```
public static void drawWindowBackground(NSRect aRect)
```

enableScreenUpdates

Enables screen updates.

```
public static void enableScreenUpdates()
```

Discussion

Reenables, for all windows of a process, the flushing of drawing operations to the screen that was previously disabled by `disableScreenUpdates` (page 719). Multiple `enableScreenUpdates` calls pop matching `disableScreenUpdates` calls off a stack.

Availability

Available in Mac OS X v10.3 and later.

See Also
`disableScreenUpdates` (page 719)

eraseRect

Erases the rectangle referred to by *boundsRect*, filling it with white.

```
public static void eraseRect(NSRect boundsRect)
```

Discussion
It does not alter the current color.

fillRectList

Fills a list of rectangles, *rectangles*, with the current color.

```
public static void fillRectList(NSRect[] rectangles)
```

fillRectListInRange

Fills a subset *aRange* of rectangles from the array *rectangles* with the current color.

```
public static void fillRectListInRange(NSRect[] rectangles, NSRange aRange)
```

fillRectListWithColors

Takes a list of rectangles, *rectangles*, and a matching list of color values.

```
public static void fillRectListWithColors(NSRect[] rectangles, NSColor[] colors)
```

Discussion
The first rectangle is filled with the first color, the second rectangle with the second color, and so on. There must be an equal number of rectangles and color values. The rectangles should not overlap; the order in which they are filled cannot be guaranteed. This method alters the current color of the current graphics state, setting it unpredictably to one of the values passed in *colors*.

See Also
`fillRectListWithColorsInRange` (page 724)

fillRectListWithColorsInRange

Fills a subset *aRange* of rectangles from the array *rectangles* with the corresponding subset colors from the array *colors*.

```
public static void fillRectListWithColors(NSRect[] rectangles, NSColor[] colors,
    NSRange aRange)
```

See Also
`fillRectListWithColors` (page 724)

frameRect

Draws a frame around the inside of `boundsRect` in the current color.

```
public static void frameRect(NSRect boundsRect)
```

Discussion

The width is equal to 1.0 in the current coordinate system. Since the frame is drawn inside the rectangle, it will be visible even if drawing is clipped to the rectangle.

frameRectWithWidth

Draws a frame around the inside of `boundsRect` in the current color.

```
public static void frameRectWithWidth(NSRect boundsRect, float width)
```

Discussion

The width is equal to `width` in the current coordinate system. Since the frame is drawn inside the rectangle, it will be visible even if drawing is clipped to the rectangle.

frameRectWithWidthUsingOperation

Draws a frame around the inside of `boundsRect` in the current color, using the compositing operation `operation`.

```
public static void frameRectWithWidthUsingOperation(NSRect boundsRect, float width,
     int operation)
```

Discussion

The width is equal to `width` in the current coordinate system. Since the frame is drawn inside the rectangle, it will be visible even if drawing is clipped to the rectangle.

highlightRect

Deprecated.

```
public static void highlightRect(NSRect aRect)
```

numberOfColorComponents

Returns the number of color components in the color space `colorSpaceName`.

```
public static int numberOfColorComponents(String colorSpaceName)
```

planarFromDepth

Returns `true` if the specified window `depth` is planar and `false` if it is not.

```
public static boolean planarFromDepth(int depth)
```

See Also
`availableWindowDepths` (page 716)
`bestDepth` (page 716)

readPixel

Returns the color of the pixel at the location specified by *aPoint*.

```
public static NSColor readPixel(NSPoint aPoint)
```

Discussion
The location argument is taken in the current coordinate system—in other words, you must lock focus on the view that contains the pixel that you wish to query and then pass the coordinate for the pixel in the view's coordinate system.

setFocusRingStyle

Sets the style of the focus ring in the current graphics context in the current locked focus view.

```
public static void setFocusRingStyle(int style)
```

Discussion
This style affects all rendering until the graphics state is restored. *style* is one of the values discussed in "Constants" (page 727).

Note that the focus ring may actually be drawn outside the view, but will be clipped to any clipping superview or the window content view.

sizeOfAttributedString

Returns the bounding box of the marks that *attributedString* draws.

```
public static NSSize sizeOfAttributedString(NSAttributedString attributedString)
```

See Also
`drawAttributedString` (page 720)

windowCount

Returns the number of onscreen windows belonging to the application.

```
public static int windowCount()
```

windowList

Provides an ordered list of the application's onscreen windows.

```
public static void windowList(int[] list)
```

Discussion

The order of windows in *list* is the same as their order in the window server's screen list (their front-to-back order on the screen).

Constants

The following colors are the standard gray values for the 2-bit-deep grayscale color space:

```
Black
DarkGray
LightGray
White
```

The following color spaces are defined by NSGraphics and are described in "About Color Spaces":

```
CalibratedBlackColorSpace
CalibratedRGBColorSpace
CalibratedWhiteColorSpace
CustomColorSpace
DeviceBlackColorSpace
DeviceCMYKColorSpace
DeviceRGBColorSpace
DeviceWhiteColorSpace
DynamicSystemColorSpace
NamedColorSpace
PatternColorSpace
```

The following device constants are defined by NSGraphics. These are the keys for device description dictionaries, such as those returned by the `deviceDictionary` methods of NSPrinter, NSScreen, and NSWindow.

Constant	Description
`DeviceBitsPerSample`	An int that indicates the bit depth of the device
`DeviceColorSpaceName`	A string describing the color space of the device
`DeviceIsPrinter`	Boolean value that tells whether the device is a printer
`DeviceIsScreen`	Boolean value that tells whether the device is a screen
`DeviceResolution`	An NSSize representing dots per inch
`DeviceSize`	An NSSize representing the device's size in points

NSGraphics defines the following constants to specify where a focus ring should be drawn and is used by:

Constant	Description
`FocusRingOnly`	Use if you don't have an image or text to add a keyboard focus ring to.
`FocusRingBelow`	Draw below text.
`FocusRingAbove`	Draw over an image.

NSGraphics defines the following constants to specify focus ring types; the focus ring type is used by NSCell (and potentially by any NSView object) and to configure if and how a control or other view object should draw its focus ring.

Constant	Description
`FocusRingTypeDefault`	The default focus ring type for an NSView or NSCell object.
`FocusRingTypeNone`	No focus ring. If you set the focus ring type to this value, NSView and NSCell objects do not draw any focus ring
`FocusRingTypeExterior`	The standard Aqua type of focus ring, which is drawn around the NSView or NSCell object.

Notifications

SystemColorsDidChangeNotification

Sent when the system colors have been changed (such as through a system control panel interface).

This notification contains no notification object and no *userInfo* dictionary.

NSGraphicsContext

Inherits from	NSObject
Package:	com.apple.cocoa.application
Companion guide	Basic Drawing

Overview

The NSGraphicsContext class is the programmatic interface to objects that represent graphics contexts. A context can be thought of as a destination to which drawing and graphics state operations are sent for execution. Each graphics context contains its own graphics environment and state.

The NSGraphicsContext class is an abstract superclass for destination-specific graphics contexts. You obtain instances of concrete subclasses with the static methods `currentContext` (page 731), `graphicsContextWithAttributes` (page 732), , , and `graphicsContextWithWindow` (page 732).

At any time there is the notion of the current context. The current context for the current thread may be set using `setCurrentContext` (page 732).

Graphics contexts are maintained on a stack. You push a graphics context onto the stack by sending it a `saveGraphicsState` (page 735) message, and pop it off the stack by sending it a `restoreGraphicsState` (page 734) message. By sending `restoreGraphicsState` (page 734) to an NSGraphicsContext object you remove it from the stack, and the next graphics context on the stack becomes the current graphics context.

Tasks

Constructors

`NSGraphicsContext` (page 731)
NSGraphicsContext is an abstract class, so you should not create your own instances.

Creating a Graphics Context

`graphicsContextWithAttributes` (page 732)
Instantiates a concrete subclass of NSGraphicsContext using the information in *`attributes`*.

`graphicsContextWithWindow` (page 732)
Creates and returns a new graphics context for drawing into *`aWindow`*.

Testing the Drawing Destination

`currentContextDrawingToScreen` (page 732)
Convenience method equivalent to sending `isDrawingToScreen` (page 734) to the result of `currentContext` (page 731).

`isDrawingToScreen` (page 734)
Returns `true` if the drawing destination is the screen or a bitmap, `false` otherwise.

Setting and Identifying the Current Context

`currentContext` (page 731)
Returns the current graphics context of the current thread.

`restoreGraphicsContext` (page 732)
Pops a graphics context from the per-thread stack, makes it current, and sends the context a `restoreGraphicsState` (page 734) message.

`setCurrentContext` (page 732)
Sets the current graphics context of the current thread to *`context`*.

`saveGraphicsContext` (page 732)
Sends the current graphics context a `saveGraphicsState` (page 735) message and pushes the context onto the per-thread stack

Getting Information About a Context

`attributes` (page 733)
Returns the receiver's attributes, if any.

Controlling the Context Flush

`flush` (page 733)
Not implemented.

`flushGraphics` (page 733)
Forces any buffered operations or data to be sent to the receiver's destination.

`restoreGraphicsState` (page 734)
Removes the receiver's graphics state from the top of the graphics state stack and makes the next graphics state the current graphics state.

`saveGraphicsState` (page 735)
Saves the current graphics state and creates a new graphics state on the top of the stack.

`setGraphicsState` (page 733)
Makes the graphics context of *`graphicState`* current, and resets graphics state.

`synchronize` (page 736)
Not implemented.

Rendering Options

`imageInterpolation` (page 733)
: Returns the receiver's interpolation (image smoothing).

`setImageInterpolation` (page 735)
: Sets the receiver's interpolation (image smoothing) to *`interpolation`*.

`shouldAntialias` (page 736)
: Returns whether the receiver uses antialiasing.

`setShouldAntialias` (page 735)
: Sets whether the receiver should use antialiasing, depending on the Boolean value *`antialias`*.

`patternPhase` (page 734)
: Returns the amount to offset the pattern color when filling the receiver.

`setPatternPhase` (page 735)
: Sets the amount to offset the pattern color when filling the receiver.

Constructors

NSGraphicsContext

NSGraphicsContext is an abstract class, so you should not create your own instances.

```
public NSGraphicsContext()
```

Discussion
Use one of the static methods to obtain an instance of the appropriate concrete subclass.

See Also
`currentContext` (page 731)
`currentContext` (page 731)
`graphicsContextWithAttributes` (page 732)
`graphicsContextWithWindow` (page 732)

Static Methods

currentContext

Returns the current graphics context of the current thread.

```
public static NSGraphicsContext currentContext()
```

Discussion
Returns a concrete subclass of NSGraphicsContext.

currentContextDrawingToScreen

Convenience method equivalent to sending `isDrawingToScreen` (page 734) to the result of `currentContext` (page 731).

```
public static boolean currentContextDrawingToScreen()
```

graphicsContextWithAttributes

Instantiates a concrete subclass of NSGraphicsContext using the information in *attributes*.

```
public static NSGraphicsContext graphicsContextWithAttributes(NSDictionary
    attributes)
```

Discussion

Use this method to create a graphics context for a window or bitmap destination. If you want to create a graphics context for a PDF or PostScript destination, do not use this method; instead, use the NSPrintOperation class to set up the printing environment needed to generate that type of information.

See the "Constants" section for the dictionary keys available for *attributes*.

graphicsContextWithWindow

Creates and returns a new graphics context for drawing into *aWindow*.

```
public static NSGraphicsContext graphicsContextWithWindow(NSWindow aWindow)
```

restoreGraphicsContext

Pops a graphics context from the per-thread stack, makes it current, and sends the context a `restoreGraphicsState` (page 734) message.

```
public static void restoreGraphicsContext()
```

saveGraphicsContext

Sends the current graphics context a `saveGraphicsState` (page 735) message and pushes the context onto the per-thread stack

```
public static void saveGraphicsContext()
```

Discussion

.

setCurrentContext

Sets the current graphics context of the current thread to *context*.

```
public static void setCurrentContext(NSGraphicsContext context)
```

Discussion

context must be a concrete subclass of NSGraphicsContext.

setGraphicsState

Makes the graphics context of *graphicState* current, and resets graphics state.

```
public static void setGraphicsState(int graphicsState)
```

Discussion

The *graphicState* must be created in the calling thread.

Instance Methods

attributes

Returns the receiver's attributes, if any.

```
public NSDictionary attributes()
```

Discussion

Screen-based graphics contexts do not store attributes, even if you create them using `graphicsContextWithAttributes` (page 732).

flush

Not implemented.

```
public void flush()
```

Discussion

Use `flushGraphics` (page 733).

flushGraphics

Forces any buffered operations or data to be sent to the receiver's destination.

```
public void flushGraphics()
```

Discussion

Graphics contexts use buffers to queue pending operations but for efficiency reasons may not always empty those buffers immediately. This method forces the buffers to be emptied.

imageInterpolation

Returns the receiver's interpolation (image smoothing).

```
public int imageInterpolation()
```

See Also
setImageInterpolation (page 735)

isDrawingToScreen

Returns true if the drawing destination is the screen or a bitmap, false otherwise.

```
public boolean isDrawingToScreen()
```

Discussion
You can interpret a return value of false to mean that the drawing destination is a printer, although it may also be a PDF or EPS file. The NSColor method set, for example, invokes isDrawingToScreen to determine whether it can apply an alpha value, which is not supported by printing contexts.

If this method returns false, you can call attributes (page 733) to see if additional information is available about the drawing destination.

patternPhase

Returns the amount to offset the pattern color when filling the receiver.

```
public NSPoint patternPhase()
```

Discussion
The pattern phase is a translation (width, height) applied before a pattern is drawn in the current context and is part of the saved graphics state of the context. The default pattern phase is (0,0). Setting the pattern phase has the effect of temporarily changing the pattern matrix of any pattern you decide to draw. For example, setting the pattern phase to (2,3) has the effect of moving the start of pattern cell tiling to the point (2,3) in default user space.

Availability
Available in Mac OS X v10.2 and later.

See Also
setPatternPhase (page 735)

restoreGraphicsState

Removes the receiver's graphics state from the top of the graphics state stack and makes the next graphics state the current graphics state.

```
public void restoreGraphicsState()
```

Discussion
This method must have been preceded with a saveGraphicsState (page 735) message to add the graphics state to the stack. Invocations of saveGraphicsState and restoreGraphicsState methods may be nested.

Restoring the graphics state restores such attributes as the current drawing style, transformation matrix, color, and font of the original graphics state.

saveGraphicsState

Saves the current graphics state and creates a new graphics state on the top of the stack.

```
public void saveGraphicsState()
```

Discussion

The new graphics state is a copy of the previous state that can be modified to handle new drawing operations.

Saving the graphics state saves such attributes as the current drawing style, transformation matrix, color, and font. To set drawing style attributes, use the methods of NSBezierPath. Other attributes are accessed through appropriate objects such as NSAffineTransform, NSColor, and NSFont.

setImageInterpolation

Sets the receiver's interpolation (image smoothing) to *interpolation*.

```
public void setImageInterpolation(int interpolation)
```

Discussion

Note that this value is not part of the graphics state, so it cannot be reset using restoreGraphicsState (page 734).

See Also

imageInterpolation (page 733)

setPatternPhase

Sets the amount to offset the pattern color when filling the receiver.

```
public void setPatternPhase(NSPoint phase)
```

Discussion

Use it when you need to line up the pattern color with another pattern, such as the pattern in a superview.

The pattern phase is a translation (width, height) applied before a pattern is drawn in the current context and is part of the saved graphics state of the context. The default pattern phase is (0,0). Setting the pattern phase has the effect of temporarily changing the pattern matrix of any pattern you decide to draw. For example, setting the pattern phase to (2,3) has the effect of moving the start of pattern cell tiling to the point (2,3) in default user space.

Availability

Available in Mac OS X v10.2 and later.

See Also

patternPhase (page 734)

setShouldAntialias

Sets whether the receiver should use antialiasing, depending on the Boolean value *antialias*.

```
public void setShouldAntialias(boolean antialias)
```

Discussion
This value is part of the graphics state and is restored by restoreGraphicsState (page 734).

See Also
shouldAntialias (page 736)

shouldAntialias

Returns whether the receiver uses antialiasing.

```
public boolean shouldAntialias()
```

See Also
setShouldAntialias (page 735)

synchronize

Not implemented.

```
public void synchronize()
```

Constants

The following constants are defined by NSGraphicsContext and are dictionary keys used by graphicsContextWithAttributes (page 732) and attributes (page 733):

Constant	Description
GraphicsContext-DestinationAttributeName	Can be an instance of NSWindow, NSBitmapImageRep when creating a graphics context. When determining the type of a graphics context, this value can be an NSMutableData, NSString, or URL.
GraphicsContext-RepresentationFormat-AttributeName	Specifies the destination file format. This value should be retrieved only and not used to create a graphics context.

The following constants are possible values for the GraphicsContextRepresentationFormatAttributeName key in a graphic context's attribute dictionary:

Constant	Description
GraphicsContextPDFFormat	Destination file format is PDF.
GraphicsContextPSFormat	Destination file format is PostScript.

The following interpolations are defined by NSGraphicsContext and used by imageInterpolation (page 733) and setImageInterpolation (page 735):

Constant	Description
`ImageInterpolationDefault`	Use the context’s default interpolation.
`ImageInterpolationNone`	No interpolation.
`ImageInterpolationLow`	Fast, low-quality interpolation.
`ImageInterpolationHigh`	Slower, higher-quality interpolation.

NSHelpManager

Inherits from	NSObject
Package:	com.apple.cocoa.application
Companion guide	Online Help

Overview

NSHelpManager provides an approach to displaying online help. An application contains one instance of NSHelpManager. Your application's code rarely needs to access NSHelpManager directly. Instead, you use Interface Builder and Xcode to set up online help for your application.

Tasks

Constructors

`NSHelpManager` (page 740)
Creates an empty NSHelpManager.

Creating an NSHelpManager Instance

`sharedHelpManager` (page 741)
Returns the shared NSHelpManager instance, creating it if it does not already exist.

Getting and Setting Context Help Mode

`setContextHelpModeActive` (page 740)
Controls context-sensitive help mode.

`isContextHelpModeActive` (page 740)
Returns `true` if the application is currently in context-sensitive help mode, `false` otherwise.

Returning Context-sensitive Help

`contextHelpForObject` (page 741)
Returns context-sensitive help for *`object`*.

`showContextHelpForObject` (page 742)
  Displays the context-sensitive help for *object* at or near the point on the screen specified by *point*.

Setting Up Context-sensitive Help

`setContextHelpForObject` (page 742)
  Associates *help* with *object*.

`removeContextHelpForObject` (page 742)
  Removes the association between *object* and its context-sensitive help.

Displaying Application Help

`findString` (page 741)
  Performs a search for the specified string in the specified book.

`openHelpAnchor` (page 742)
  Finds and displays the text at the given anchor location in the given book.

Constructors

NSHelpManager

Creates an empty NSHelpManager.

```
public NSHelpManager()
```

Static Methods

isContextHelpModeActive

Returns `true` if the application is currently in context-sensitive help mode, `false` otherwise.

```
public static boolean isContextHelpModeActive()
```

Discussion
In context-sensitive help mode, when a user clicks a user interface item, help for that item is displayed in a small window just below the cursor.

See Also
`setContextHelpModeActive` (page 740)

setContextHelpModeActive

Controls context-sensitive help mode.

```
public static void setContextHelpModeActive(boolean flag)
```

Discussion

If *flag* is `true`, the application enters context-sensitive help mode. If *flag* is `false`, the application returns to normal operation.

You never send this message directly; instead, the NSApplication method `activateContextHelpMode` (page 106) activates context-sensitive help mode, and the first mouse click after displaying the context-sensitive help window deactivates it.

When the application enters context-sensitive help mode, NSHelpManager posts a `ContextHelpModeDidActivateNotification` (page 743) to the default notification center. When the application returns to normal operation, NSHelpManager posts a `ContextHelpModeDidDeactivateNotification` (page 743).

See Also

`isContextHelpModeActive` (page 740)

sharedHelpManager

Returns the shared NSHelpManager instance, creating it if it does not already exist.

```
public static NSHelpManager sharedHelpManager()
```

Instance Methods

contextHelpForObject

Returns context-sensitive help for *object*.

```
public NSAttributedString contextHelpForObject(Object object)
```

See Also

`setContextHelpForObject` (page 742)

`showContextHelpForObject` (page 742)

findString

Performs a search for the specified string in the specified book.

```
public void findString(String query, String book)
```

Discussion

The *query* parameter specifies the search string, and *book* should be a localized help book name or `null`. If *book* is `null`, all installed help books are searched.

This is a wrapper for `AHRegisterHelpBook` (which is called only once to register the help book specified in the application's main bundle) and `AHSearch`.

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

openHelpAnchor

Finds and displays the text at the given anchor location in the given book.

```
public void openHelpAnchor(String anchor, String book)
```

Discussion
The *anchor* parameter is a string specifying the anchor location, and *book* should be a localized help book name or `null`. If *book* is `null`, all installed help books are searched.

This method is a wrapper for `AHRegisterHelpBook` (which is called only once to register the help book specified in the application's main bundle) and `AHLookupAnchor`.

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

removeContextHelpForObject

Removes the association between *object* and its context-sensitive help.

```
public void removeContextHelpForObject(Object object)
```

Discussion
If *object* does not have context-sensitive help associated with it, this method does nothing. Typically, you use Interface Builder to remove context-sensitive help from an item.

See Also
`setContextHelpForObject` (page 742)

setContextHelpForObject

Associates *help* with *object*.

```
public void setContextHelpForObject(NSAttributedString help, Object object)
```

Discussion
When the application enters context-sensitive help mode, if *object* is clicked, *help* will appear in the context-sensitive help window. Typically, you use Interface Builder to associate context-sensitive help with an object.

See Also
`removeContextHelpForObject` (page 742)

showContextHelpForObject

Displays the context-sensitive help for *object* at or near the point on the screen specified by *point*.

```
public boolean showContextHelpForObject(Object object, NSPoint point)
```

Discussion

This point is usually just under the cursor. Returns `true` if it successfully displays context-sensitive help for the object, `false` if it cannot (for example, if there is no context-sensitive help associated with this object).

See Also

`contextHelpForObject` (page 741)

Notifications

ContextHelpModeDidActivateNotification

Posted when the application enters context-sensitive help mode. This typically happens when the user holds down the Help key.

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

ContextHelpModeDidDeactivateNotification

Posted when the application exits context-sensitive help mode. This happens when the user clicks the mouse button while the cursor is anywhere on the screen after displaying a context-sensitive help topic.

The notification object is the NSHelpManager object. This notification does not contains a *`userInfo`* dictionary.

NSImage

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.application
Companion guide	Drawing and Images

Overview

An NSImage object contains an image that can be composited anywhere without first being drawn in any particular view.

Tasks

Constructors

`NSImage` (page 750)
Creates an empty NSImage with a zero-sized frame rectangle.

Setting the Size of the Image

`setSize` (page 767)
Sets the width and height of the image.

`size` (page 767)
Returns the size of the receiver.

Referring to Images by Name

`imageNamed` (page 752)
Returns the NSImage instance associated with *name*.

`setName` (page 766)
Registers the receiver under the name specified by *aString*, provided that no other NSImage is registered using that name.

`name` (page 762)
: Returns the name assigned to the receiver, or `null` if no name has been assigned.

Specifying the Image

`addRepresentation` (page 754)
: Adds *`imageRep`* to the receiver's list of representations.

`addRepresentations` (page 754)
: Adds each of the representations in *`imageReps`* to the receiver's list of representations.

`lockFocus` (page 761)
: Prepares the current device for drawing the receiver by setting the offscreen window (where the receiver's representation will be cached) as the current window. It also sets the coordinate system of the offscreen window's relevant area to the current coordinate system.

`lockFocusOnRepresentation` (page 761)
: Prepares the current device for drawing the *`imageRepresentation`* receiver by setting the offscreen window (where the representation will be cached) as the current window. It also sets the coordinate system of the offscreen window's relevant area to the current coordinate system.

`unlockFocus` (page 768)
: Balances a previous `lockFocus` (page 761) or `lockFocusOnRepresentation` (page 761) message.

Using the Image

`compositeToPoint` (page 756)
: Composites the image to the location specified by *`aPoint`* using the specified compositing operation, *`op`*.

`compositeToPointFromRect` (page 757)
: Composites the portion of the image enclosed by the *`srcRect`* rectangle to the location specified by *`aPoint`* in the current coordinate system.

`compositeToPointFromRectWithFraction` (page 757)
: Partially composites the *`srcRect`* portion of the image to the location specified by *`aPoint`*, using the specified compositing operation, *`op`*.

`compositeToPointWithFraction` (page 758)
: Partially composites the image to the location specified by *`aPoint`*, using the specified compositing operation, *`op`*.

`dissolveToPoint` (page 758)
: Partially composites the image to the location specified by *`aPoint`*, just as `compositeToPoint` (page 756) does, but uses the `CompositeSourceOver` operator implicitly.

`dissolveToPointFromRect` (page 758)
: Partially composites the *`srcRect`* portion of the image to the location specified by *`aPoint`*, just as `compositeToPointFromRect` (page 757) does, but uses the `CompositeSourceOver` operator implicitly.

Choosing Which Image Representation to Use

`setPrefersColorMatch` (page 766)
: Sets whether color matches are preferred over resolution matches when determining which representation to use.

`prefersColorMatch` (page 762)

`setUsesEPSOnResolutionMismatch` (page 767)
: Sets whether EPS representations are preferred when there are no representations that match the resolution of the device, depending on the Boolean value *`flag`*.

`usesEPSOnResolutionMismatch` (page 768)
: Returns whether EPS representations are preferred when there are no representations that match the resolution of the device.

`setMatchesOnMultipleResolution` (page 765)
: Sets whether image representations with resolutions that are integral multiples of the resolution of the device are considered to match the device, depending on the Boolean value *`flag`*.

`matchesOnMultipleResolution` (page 762)
: Returns `true` if the resolution of the device and the resolution specified for the image are considered to match if one is an integer multiple of the other, and `false` if device and image resolutions are considered to match only if they are exactly the same.

Getting the Representations

`bestRepresentationForDevice` (page 755)
: Returns the best representation for the device described by *`deviceDescription`*.

`representations` (page 763)
: Returns an array containing all the representations of the receiver.

`removeRepresentation` (page 763)
: Removes and releases the *`imageRep`* representation from the receiver's list of representations.

Determining How the Image Is Stored

`setCachedSeparately` (page 764)
: Sets whether each image representation will be cached in its own offscreen window or in a window shared with other images.

`isCachedSeparately` (page 760)
: Returns `true` if each representation of the receiver is cached separately in an offscreen window of its own and `false` if they can be cached in offscreen windows together with other images.

`setDataRetained` (page 765)
: Sets whether the receiver retains the data needed to render the image, depending on the Boolean value *`flag`*.

`isDataRetained` (page 760)
: Returns `true` if the receiver retains the data needed to render the image and `false` if it doesn't.

`setCacheDepthMatchesImageDepth` (page 764)
    Sets whether the application's default depth limit applies to the offscreen windows where the receiver's representations are cached.

`cacheDepthMatchesImageDepth` (page 755)
    Returns `false` if the application's default depth limit applies to the offscreen windows where the receiver's representations are cached.

`cacheMode` (page 756)
    Returns the receiver's caching mode.

`setCacheMode` (page 764)
    Set the receiver's caching mode.

Drawing the Image

`drawAtPoint` (page 759)
    Partially composites the *`srcRect`* portion of the image to the location *`point`* in the current coordinate system, using the specified operation, *`op`*.

`drawInRect` (page 759)
    Partially composites the *`srcRect`* portion of the image inside the *`dstRect`* portion of the current coordinate system, using the specified operation, *`op`*.

`drawRepresentationInRect` (page 759)
    Fills the specified rectangle with the background color, then sends the *`imageRep`* a `drawInRect` (page 789) message to draw itself inside the *`dstRect`* rectangle (if the NSImage is scalable) or a `drawAtPoint` (page 789) message to draw itself at the location of the rectangle (if the NSImage is not scalable).

Determining How the Image Is Drawn

`isValid` (page 761)
    Returns `true` if a representation for the receiver can drawn in the cache and `false` if it can't—for example, because the file from which it was initialized is nonexistent, or the data in that file is invalid.

`setScalesWhenResized` (page 766)
    Sets whether representations with sizes that differ from the size of the receiver will be scaled to fit.

`scalesWhenResized` (page 763)

`setBackgroundColor` (page 763)
    Sets the background color of the image.

`backgroundColor` (page 755)
    Returns the background color of the rectangle where the image is cached.

`setFlipped` (page 765)
    Determines whether the polarity of the y axis is inverted when drawing an image.

`isFlipped` (page 760)
    Returns `true` if a vertically flipped coordinate system is used when locating the position of the receiver and `false` if it isn't.

`recache` (page 762)
    Invalidates the offscreen caches of all representations and frees them.

Assigning a Delegate

setDelegate (page 765)
Makes *anObject* the delegate of the receiver.

delegate (page 758)
Returns the delegate of the receiver, or null if no delegate has been set.

Producing TIFF Data for the Image

TIFFRepresentation (page 768)
Returns a data object containing TIFF for all representations, using their default compressions.

Testing Image Data Sources

canInitWithPasteboard (page 752)
Tests whether the receiver can create an instance of itself from the data represented by *pasteboard*.

imageFileTypes (page 752)
Returns an array of strings representing those file types for which a registered NSImageRep exists.

imageUnfilteredFileTypes (page 753)
Returns an array of strings representing those file types for which a registered NSImageRep exists.

imagePasteboardTypes (page 753)
Returns an array of pasteboard types for which a registered NSImageRep exists.

imageUnfilteredPasteboardTypes (page 754)
Returns an array of pasteboard types for which a registered NSImageRep exists.

Incremental Loading

cancelIncrementalLoad (page 756)
Immediately cancels the download operation if the image is being incrementally loaded.

Loading an image

imageDidLoadRepresentation (page 771) *delegate method*
This method is invoked when *image* has been as fully decompressed as is possible.

imageDidLoadPartOfRepresentation (page 771) *delegate method*
During incremental loading, this method is called repeatedly to inform the delegate that more of the image *rep* is available.

imageDidLoadRepresentationHeader (page 771) *delegate method*
During incremental loading, the image invokes this method once enough data has been read to determine the size of the image.

imageWillLoadRepresentation (page 772) *delegate method*
For incremental loading, this method is invoked when you first draw *image* or otherwise require the bitmap data.

Drawing an image

imageDidNotDraw (page 772) *delegate method*
Implemented by the delegate to respond to a message sent by the sender (an NSImage) when the sender was unable, for whatever reason, to composite or lock focus on its image to draw in *aRect*.

Constructors

NSImage

Creates an empty NSImage with a zero-sized frame rectangle.

```
public NSImage()
```

Creates an NSImage based on *byReferencing*.

```
public NSImage(String filename, boolean byReferencing)
```

Discussion
If *byReferencing* is `true`, the NSImage doesn't actually open *filename* or create image representations from its data until an application attempts to composite or requests information about the NSImage.

The setDataRetained (page 765) is invoked with an argument of `true`, thus enabling the image to hold onto its filename. Note that if an image created by referencing is archived, only the filename will be saved.

If *byReferencing* is `false`, *filename* is opened, and one or more image representations is created from its data.

In both cases, *filename* may be a full or relative pathname and should include an extension that identifies the data type in the file. If *byReferencing* is `true`, the constructor will look for an NSImageRep subclass that handles that data type from among those registered with NSImage. If *byReferencing* is `false`, 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.

If *byReferencing* is `false` and at least one image representation can't be created from the contents of *filename*, `null` is returned.If *byReferencing* is `true` and the new instance can't be initialized, `null` is returned. Since creating by reference doesn't actually create image representations for the data, your application should do error checking before attempting to use the image; one way to do so is by invoking the isValid (page 761) method to check whether the image can be drawn.

Creates a new NSImage, setting its size to *aSize*.

```
public NSImage(NSSize aSize)
```

Discussion
The size should be specified in units of the base coordinate system. Although you can create an NSImage without specifying a size by passing a size of (0.0, 0.0), the NSImage's size must be set before it can be used.

Creates a new NSImage, with the contents of the data object *aData*.

```
public NSImage(NSData aData)
```

Discussion
If unable to create one or more image representations from *aData*, `null` is returned.

Creates a new NSImage, with the contents of the URL *aURL*.

```
public NSImage(java.net.URL aURL)
```

Discussion
If at least one image representation can't be created from the contents of *aURL*, `null` is returned.

Creates a new NSImage instance for the file at *url*.

```
public com.apple.cocoa.application.NSImage(URL url, boolean)
```

Discussion
This constructor initializes lazily: The NSImage doesn't actually open *url* or create image representations from its data until an application attempts to composite or requests information about the NSImage.

The URL should include an extension that identifies the data type in the file. The mechanism that actually creates the image representation for *url* will look for an NSImageRep subclass that handles that data type from among those registered with NSImage.

After finishing the initialization, this method returns the NSImage. However, if the new instance can't be initialized, it's freed and `null` is returned. Since this constructor doesn't actually create image representations for the data, your application should do error checking before attempting to use the image; one way to do so is by invoking the `isValid` (page 761) method to check whether the image can be drawn.

This method invokes `setDataRetained` (page 765) with an argument of `true`, thus enabling it to hold onto its URL. Note that if an image created with this method is archived, only the URL will be saved.

Availability
Available in Mac OS X v10.2 and later.

Creates a new NSImage, with data from *aPasteboard*.

```
public NSImage(NSPasteboard aPasteboard)
```

Discussion
The *aPasteboard* object should contain a type returned by one of the registered NSImageRep's `imageUnfilteredPasteboardTypes` (page 787) methods; the default types supported are `NSPasteboard.PostscriptPboardType` (NSEPSImageRep), `NSPasteboard.PDFPboardType` (NSPDFImageRep), `NSPasteboard.PICTPboardType` (NSPICTImageRep), and `NSPasteboard.TIFFPboardType` (NSBitmapImageRep). If *aPasteboard* contains a `NSPasteboard.FilenamesPboardType`, the filename should have an extension returned by one of the registered NSImageRep's `imageUnfilteredFileTypes` (page 786) methods; the default types supported include "`tiff`", "`gif`", "`jpg`" (all in NSBitmapImageRep), "`pdf`" (NSPDFImageRep), "`pict`" (NSPICTImageRep), and "`eps`" (NSEPSImageRep).

If unable to create one or more image representations, `null` is returned.

Static Methods

canInitWithPasteboard

Tests whether the receiver can create an instance of itself from the data represented by *pasteboard*.

```
public static boolean canInitWithPasteboard(NSPasteboard pasteboard)
```

Discussion

Returns `true` if the receiver's list of registered NSImageReps includes a class that can handle the data represented by *pasteboard*.

NSImage uses the NSImageRep class method `imageUnfilteredPasteboardTypes` (page 787) to find a 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` (page 787) method to notify NSImage of the pasteboard types your class supports.

See Also

`imagePasteboardTypes` (page 753)

imageFileTypes

Returns an array of strings representing those file types for which a registered NSImageRep exists.

```
public static NSArray imageFileTypes()
```

Discussion

This list includes all file types supported by registered subclasses of NSImageRep, plus 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 NSOpenPanel's `runModalForTypes` (page 1024) method.

File types are identified by extension and HFS file types.

When creating a subclass of NSImageRep that accepts image data from nondefault file types, override NSImageRep's `imageUnfilteredFileTypes` (page 786) method to notify NSImage of the file types your class supports.

See Also

`imageUnfilteredFileTypes` (page 753)

imageNamed

Returns the NSImage instance associated with *name*.

```
public static NSImage imageNamed(String name)
```

Discussion

The returned object is one that's been assigned a name with the `setName` (page 766) method.

If there's no known NSImage with *name*, this method tries to create one by searching for image data in the application's main bundle (see NSBundle's class description for a description of how the bundle's contents are searched). If a file contains data for more than one image, a separate representation is created for each. If an image representation can't be found for *name*, no object is created, and `null` is returned.

The preferred way to name an image is to ask for a name without the extension, but to include the extension for a filename.

One particularly useful image is referenced by the string `@"NSApplicationIcon"`. If you supply this string to `imageNamed` (page 752), the returned image will be the application's own icon. Icons for other applications can be obtained through the use of methods declared in the NSWorkspace class.

NSImage keeps a reference to the image in a table until the image name is cleared. Consequently you do not need to retain the returned image object unless its name could be cleared. You clear an image name by sending the associated NSImage object a `setName` (page 766) message with an argument of null. This message removes the image from the table and autoreleases the object. However, if the image has been fetched elsewhere using `imageNamed` (page 752), then those instances could still be used.

See Also

`setName` (page 766)

`name` (page 762)

`iconForFile` (page 1904) (NSWorkspace)

`imageFileTypes` (page 752)

imagePasteboardTypes

Returns an array of pasteboard types for which a registered NSImageRep exists.

```
public static NSArray imagePasteboardTypes()
```

Discussion

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 includes `NSPasteboard.PDFPboardType`, `NSPasteboard.PICTPboardType`, `NSPasteboard.PostScriptPboardType`, and `NSPasteboard.TIFFPboardType`.

When creating a subclass of NSImageRep that accepts image data from nondefault pasteboard types, override NSImageRep's `imageUnfilteredPasteboardTypes` (page 787) method to notify NSImage of the pasteboard types your class supports.

See Also

`imageUnfilteredPasteboardTypes` (page 754)

imageUnfilteredFileTypes

Returns an array of strings representing those file types for which a registered NSImageRep exists.

```
public static NSArray imageUnfilteredFileTypes()
```

Discussion
This list consists of all file types supported by registered subclasses of NSImageRep, but does not include 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 NSOpenPanel's `runModalForTypes` (page 1024) method.

See Also
`imageFileTypes` (page 752)

imageUnfilteredPasteboardTypes

Returns an array of pasteboard types for which a registered NSImageRep exists.

```
public static NSArray imageUnfilteredPasteboardTypes()
```

Discussion
This list consists of all pasteboard types supported by registered subclasses of NSImageRep, but does not include those that can be converted to supported pasteboard types through a user-installed filter service.

See Also
`imagePasteboardTypes` (page 753)

Instance Methods

addRepresentation

Adds *imageRep* to the receiver's list of representations.

```
public void addRepresentation(NSImageRep imageRep)
```

Discussion
After invoking this method, you may need to explicitly set features of the new representation, such as size, number of colors, and so on. This fact is true in particular if the NSImage has multiple image representations to choose from. See NSImageRep (page 779) and its subclasses for the methods you use to complete initialization.

Any representation added by this method is retained by the NSImage. Note that representations can't be shared among NSImages.

See Also
`representations` (page 763)
`removeRepresentation` (page 763)

addRepresentations

Adds each of the representations in *imageReps* to the receiver's list of representations.

```
public void addRepresentations(NSArray imageReps)
```

Discussion

After invoking this method, you may need to explicitly set features of the new representations, such as size, number of colors, and so on. This fact is true in particular if the NSImage has multiple image representations to choose from. See NSImageRep (page 779) and its subclasses for the methods you use to complete initialization.

Representations added by this method are retained by the NSImage. Note that representations can't be shared among NSImages.

See Also

`representations` (page 763)

`removeRepresentation` (page 763)

backgroundColor

Returns the background color of the rectangle where the image is cached.

```
public NSColor backgroundColor()
```

Discussion

If no background color has been specified, NSColor's `clearColor` (page 357) is returned, indicating a 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.

bestRepresentationForDevice

Returns the best representation for the device described by *`deviceDescription`*.

```
public NSImageRep bestRepresentationForDevice(NSDictionary deviceDescription)
```

Discussion

If *`deviceDescription`* is `null`, the current device is assumed. "How an Image Representation Is Chosen" outlines the process NSImage goes through to determine the "best" representation for a given device.

See Also

`representations` (page 763)

`prefersColorMatch` (page 762)

cacheDepthMatchesImageDepth

Returns `false` if the application's default depth limit applies to the offscreen windows where the receiver's representations are cached.

```
public boolean cacheDepthMatchesImageDepth()
```

Discussion

If window depths are instead determined by the specifications of the representations, `cacheDepthMatchesImageDepth` returns `true`.

See Also
`setCacheDepthMatchesImageDepth` (page 764)

cacheMode

Returns the receiver's caching mode.

```
public int cacheMode()
```

Discussion
Possible return values are described in "Constants" (page 769).

Availability
Available in Mac OS X v10.2 and later.

See Also
`setCacheMode` (page 764)

cancelIncrementalLoad

Immediately cancels the download operation if the image is being incrementally loaded.

```
public void cancelIncrementalLoad()
```

Discussion
This call has no effect if the image is not loading.

Availability
Available in Mac OS X v10.2 and later.

compositeToPoint

Composites the image to the location specified by *aPoint* using the specified compositing operation, *op*.

```
public void compositeToPoint(NSPoint aPoint, int op)
```

Discussion
The *aPoint* argument specified in the current coordinate system—the coordinate system of the currently focused NSView—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. *op* should be one of the compositing operations described in "Constants" (page 769).

The image is composited from its offscreen 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. Bitmap representations are not cached unless explicitly rendered, by invoking `lockFocus` (page 761) and `unlockFocus` (page 768), 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

`dissolveToPoint` (page 758)

`drawAtPoint` (page 759)

compositeToPointFromRect

Composites the portion of the image enclosed by the *srcRect* rectangle to the location specified by *aPoint* in the current coordinate system.

```
public void compositeToPointFromRect(NSPoint aPoint, NSRect srcRect, int op)
```

Discussion

The *aPoint* argument is the same as for `compositeToPoint` (page 756). *op* should be one of the compositing operations described in "Constants" (page 769).

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, when multiple representations are cached in a single window, it can conceivably composite areas that don't properly belong to the image, if the *srcRect* rectangle happens to include them. If this inclusion 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` (page 764) 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 just 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

`dissolveToPointFromRect` (page 758)

`drawAtPoint` (page 759)

compositeToPointFromRectWithFraction

Partially composites the *srcRect* portion of the image to the location specified by *aPoint*, using the specified compositing operation, *op*.

```
public void compositeToPointFromRectWithFraction(NSPoint aPoint, NSRect srcRect,
    int op, float delta)
```

Discussion

Behaves the same as `compositeToPointFromRect` (page 757), except that the *delta* argument specifies how much of the resulting composite will come from the NSImage object. By invoking repeatedly with *delta* increasing from 0.0 to 1.0 (refreshing and flushing the destination view between invocations), you gradually fade the original destination into the fully composited image.

See Also

`dissolveToPointFromRect` (page 758)

`drawAtPoint` (page 759)

compositeToPointWithFraction

Partially composites the image to the location specified by `aPoint`, using the specified compositing operation, `op`.

```
public void compositeToPointWithFraction(NSPoint aPoint, int op, float delta)
```

Discussion

Behaves the same as `compositeToPoint` (page 756), except that the `delta` argument specifies how much of the resulting composite will come from the NSImage object. By invoking repeatedly with `delta` increasing from 0.0 to 1.0 (refreshing and flushing the destination view between invocations), you gradually fade the original destination into the fully composited image.

See Also

`dissolveToPoint` (page 758)

delegate

Returns the delegate of the receiver, or `null` if no delegate has been set.

```
public Object delegate()
```

See Also

`setDelegate` (page 765)

dissolveToPoint

Partially composites the image to the location specified by `aPoint`, just as `compositeToPoint` (page 756) does, but uses the `CompositeSourceOver` operator implicitly.

```
public void dissolveToPoint(NSPoint aPoint, float delta)
```

Discussion

`delta` is a fraction from 0.0 to 1.0 that specifies how much of the resulting composite will come from the NSImage. If the source image contains alpha, this operation may promote the destination NSWindow to contain alpha.

To slowly dissolve one image into another, this method (or `dissolveToPointFromRect` (page 758)) needs to be invoked repeatedly with an ever-increasing `delta`. Since `delta` 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 replacement 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` (page 756). The `delta` argument is ignored.

dissolveToPointFromRect

Partially composites the `srcRect` portion of the image to the location specified by `aPoint`, just as `compositeToPointFromRect` (page 757) does, but uses the `CompositeSourceOver` operator implicitly.

```
public void dissolveToPointFromRect(NSPoint aPoint, NSRect srcRect, float delta)
```

Discussion
delta is a fraction from 0.0 to 1.0 that specifies how much of the resulting composite will come from the NSImage. If the source image contains alpha, this operation may promote the destination NSWindow.

When printing, this method is identical to `compositeToPointFromRect` (page 757). The *delta* argument is ignored.

See Also
`dissolveToPoint` (page 758)

drawAtPoint

Partially composites the *srcRect* portion of the image to the location *point* in the current coordinate system, using the specified operation, *op*.

```
public void drawAtPoint(NSPoint point, NSRect srcRect, int op, float delta)
```

Discussion
delta is a fraction from 0.0 to 1.0 that specifies how much of the resulting composite will come from the NSImage. The composite is positioned and oriented according to the current coordinate system. The image is rotated and scaled as needed.

See Also
`compositeToPointFromRectWithFraction` (page 757)
`dissolveToPoint` (page 758)

drawInRect

Partially composites the *srcRect* portion of the image inside the *dstRect* portion of the current coordinate system, using the specified operation, *op*.

```
public void drawInRect(NSRect dstRect, NSRect srcRect, int op, float delta)
```

Discussion
delta is a fraction from 0.0 to 1.0 that specifies how much of the resulting composite will come from the NSImage. The composite is positioned and oriented according to the current coordinate system. The image is rotated and scaled as needed.

See Also
`dissolveToPoint` (page 758)

drawRepresentationInRect

Fills the specified rectangle with the background color, then sends the *imageRep* a `drawInRect` (page 789) message to draw itself inside the *dstRect* rectangle (if the NSImage is scalable) or a `drawAtPoint` (page 789) message to draw itself at the location of the rectangle (if the NSImage is not scalable).

```
public boolean drawRepresentationInRect(NSImageRep imageRep, NSRect dstRect)
```

Discussion
The rectangle is located in the current window and is specified in the current coordinate system. This method returns the value returned by the `drawInRect` (page 789) or `drawAtPoint` (page 789) method, which indicates whether or not the representation was successfully drawn.

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.

If the background color is fully transparent and the image isn't being cached by the NSImage, the rectangle won't be filled before the representation draws.

isCachedSeparately

Returns `true` if each representation of the receiver is cached separately in an offscreen window of its own and `false` if they can be cached in offscreen windows together with other images.

```
public boolean isCachedSeparately()
```

Discussion
A return of `false` doesn't mean that the windows are, in fact, shared, just that they can be. The default is `false`.

isDataRetained

Returns `true` if the receiver retains the data needed to render the image and `false` if it doesn't.

```
public boolean isDataRetained()
```

Discussion
The default is `false`. If the data is available 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.

isFlipped

Returns `true` if a vertically flipped coordinate system is used when locating the position of the receiver and `false` if it isn't.

```
public boolean isFlipped()
```

Discussion
The default is `false`.

See Also
`setFlipped` (page 765)

isValid

Returns true if a representation for the receiver can drawn in the cache and false if it can't—for example, because the file from which it was initialized is nonexistent, or the data in that file is invalid.

```
public boolean isValid()
```

Discussion

If no representations exist for the receiver, isValid first creates a cache with the default depth.

lockFocus

Prepares the current device for drawing the receiver by setting the offscreen window (where the receiver's representation will be cached) as the current window. It also sets the coordinate system of the offscreen window's relevant area to the current coordinate system.

```
public void lockFocus()
```

Discussion

If the receiver has no representations, lockFocus first creates one with the default depth. See "How an Image Representation Is Chosen" for information on how the "best" representation is chosen.

A successful lockFocus message must be balanced by a subsequent unlockFocus (page 768) message to the same NSImage. These messages bracket the code that draws the image.

If lockFocus is unable to focus on the representation, it throws an ImageCacheException.

See Also

bestRepresentationForDevice (page 755)

isValid (page 761)

prefersColorMatch (page 762)

representations (page 763)

lockFocusOnRepresentation

Prepares the current device for drawing the *imageRepresentation* receiver by setting the offscreen window (where the representation will be cached) as the current window. It also sets the coordinate system of the offscreen window's relevant area to the current coordinate system.

```
public void lockFocusOnRepresentation(NSImageRep imageRepresentation)
```

Discussion

If *imageRepresentation* is null, lockFocusOnRepresentation acts like lockFocus (page 761), setting focus to the best representation for the NSImage. Otherwise, *imageRepresentation* must be one of the representations in the NSImage.

A successful lockFocusOnRepresentation message must be balanced by a subsequent unlockFocus (page 768) message to the same NSImage. These messages bracket the code that draws the image.

If lockFocusOnRepresentation is unable to focus on the representation, it throws an ImageCacheException.

See Also
isValid (page 761)

matchesOnMultipleResolution

Returns true if the resolution of the device and the resolution specified for the image are considered to match if one is an integer multiple of the other, and false if device and image resolutions are considered to match only if they are exactly the same.

```
public boolean matchesOnMultipleResolution()
```

Discussion
The default is true.

See Also
setMatchesOnMultipleResolution (page 765)

name

Returns the name assigned to the receiver, or null if no name has been assigned.

```
public String name()
```

See Also
setName (page 766)

prefersColorMatch

```
public boolean prefersColorMatch()
```

Discussion
Returns true if, when selecting the representation it will use, the receiver first looks for one that matches the color capability of the rendering device (choosing a grayscale representation for a monochrome device and a color representation for a color device), then if necessary narrows the selection by looking for one that matches the resolution of the device. If the return is false, 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 true.

See Also
setPrefersColorMatch (page 766)

recache

Invalidates the offscreen caches of all representations and frees them.

```
public void recache()
```

Discussion
The next time any representation is composited, it will first be asked to redraw itself in the cache. NSCachedImageReps aren't destroyed by this method.

If an image is likely not to be used again, it's a good idea to free its caches, since that will reduce the amount of memory consumed by your program and therefore improve performance. If you modify an image represenation of an NSImage object, you must send `recache` (page 762) to the image object to have the changed representation redrawn and recached.

removeRepresentation

Removes and releases the *imageRep* representation from the receiver's list of representations.

```
public void removeRepresentation(NSImageRep imageRep)
```

See Also
`representations` (page 763)

representations

Returns an array containing all the representations of the receiver.

```
public NSArray representations()
```

scalesWhenResized

```
public boolean scalesWhenResized()
```

Discussion
Returns `true` if image representations are scaled to fit the size specified for the receiver. If representations are not scalable, this method returns `false`. The default is `false`.

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` (page 766)
`setSize` (page 767)

setBackgroundColor

Sets the background color of the image.

```
public void setBackgroundColor(NSColor aColor)
```

Discussion
The default is NSColor's `clearColor` (page 357), indicating 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 receiver to recache itself.

See Also
`recache` (page 762)

backgroundColor (page 755)

setCacheDepthMatchesImageDepth

Sets whether the application's default depth limit applies to the offscreen windows where the receiver's representations are cached.

```
public void setCacheDepthMatchesImageDepth(boolean flag)
```

Discussion

If flag is `true`, window depths are determined by the specifications of the representations. If *flag* is `false` (the default), the application's default depth limit applies to the off-screen windows where the receiver's representations are cached. This method doesn't cause the receiver to recache itself.

See Also

cacheDepthMatchesImageDepth (page 755)

lockFocus (page 761)

recache (page 762)

setCachedSeparately

Sets whether each image representation will be cached in its own offscreen window or in a window shared with other images.

```
public void setCachedSeparately(boolean flag)
```

Discussion

If *flag* is `true`, each representation is guaranteed to be in a separate window. If *flag* is `false` (the default), a representation can be cached together with other images, though in practice it might not be.

If an NSImage is to be resized frequently, it's more efficient to cache its representations separately.

This method doesn't invalidate any existing caches.

See Also

recache (page 762)

setCacheMode

Set the receiver's caching mode.

```
public void setCacheMode(int mode)
```

Discussion

Possible values for *mode* are described in "Constants" (page 769).

Availability

Available in Mac OS X v10.2 and later.

See Also

cacheMode (page 756)

setDataRetained

Sets whether the receiver retains the data needed to render the image, depending on the Boolean value *flag*.

```
public void setDataRetained(boolean flag)
```

Discussion

The default is `false`. If the data is available 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 receiver to retain the data. However, if the receiver reads image data from a file that could change, you may want to have it keep the data itself. Generally, this method is useful to redraw the image to a device of different resolution.

setDelegate

Makes *anObject* the delegate of the receiver.

```
public void setDelegate(Object anObject)
```

See Also

`delegate` (page 758)

setFlipped

Determines whether the polarity of the y axis is inverted when drawing an image.

```
public void setFlipped(boolean flag)
```

Discussion

If *flag* is `true`, 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; it doesn't affect the coordinate system for specifying portions of the image for methods like `compositeToPointFromRect` (page 757) or `dissolveToPointFromRect` (page 758). This method doesn't cause the receiver to recache itself.

If the image is cached while *flag* is `true`, then the receiver is cached flipped and changing *flag* does not affect the orientation of the cached image.

See Also

`isFlipped` (page 760)

`recache` (page 762)

setMatchesOnMultipleResolution

Sets whether image representations with resolutions that are integral multiples of the resolution of the device are considered to match the device, depending on the Boolean value *flag*.

```
public void setMatchesOnMultipleResolution(boolean flag)
```

Discussion

The default is `true`.

See Also

`matchesOnMultipleResolution` (page 762)

setName

Registers the receiver under the name specified by *aString*, provided that no other NSImage is registered using that name.

```
public boolean setName(String aString)
```

Discussion

If the receiver is already registered under another name, `setName` (page 766) first unregisters the prior name. `setName` returns `true` unless another NSImage is registered using the name specified by *aString*, in which case `setName` does nothing and returns `false`.

See Also

`imageNamed` (page 752)

`name` (page 762)

setPrefersColorMatch

Sets whether color matches are preferred over resolution matches when determining which representation to use.

```
public void setPrefersColorMatch(boolean flag)
```

Discussion

If *flag* is `true`, the receiver 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), and then if necessary narrows the selection by trying to match the resolution of the representation to the resolution of the device. If *flag* is `false`, 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 `true`.

See Also

`prefersColorMatch` (page 762)

setScalesWhenResized

Sets whether representations with sizes that differ from the size of the receiver will be scaled to fit.

```
public void setScalesWhenResized(boolean flag)
```

Discussion

If *flag* is `true`, representations are scaled to fit. The default is `false`.

A representation added with either `addRepresentation` (page 754) or `addRepresentations` (page 754) 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` (page 763)
`setSize` (page 767)

setSize

Sets the width and height of the image.

```
public void setSize(NSSize aSize)
```

Discussion
aSize should be in units of the base coordinate system.

The size of an NSImage must be set before it can be used. You can change the size of an NSImage 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 offscreen window to recreate the cache.

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.

See Also
`size` (page 767)
`setScalesWhenResized` (page 766)

setUsesEPSOnResolutionMismatch

Sets whether EPS representations are preferred when there are no representations that match the resolution of the device, depending on the Boolean value *flag*.

```
public void setUsesEPSOnResolutionMismatch(boolean flag)
```

Discussion
The default is `false`.

See Also
`usesEPSOnResolutionMismatch` (page 768)
`setMatchesOnMultipleResolution` (page 765)

size

Returns the size of the receiver.

```
public NSSize size()
```

Discussion
If no size has been set, and no size can be determined from any of the receiver's representations, the returned NSSize will have a width and height of 0.0.

See Also
`setSize` (page 767)

TIFFRepresentation

Returns a data object containing TIFF for all representations, using their default compressions.

```
public NSData TIFFRepresentation()
```

Returns a data object containing TIFF for all representations, using the specified compression type and compression factor.

```
public NSData TIFFRepresentation(int comp, float aFloat)
```

Discussion
Legal values for *comp* are described in "TIFF Compression in NSBitmapImageReps". *aFloat* provides a hint for those compression types that implement variable compression ratios; currently only JPEG compression uses a compression factor.

If the specified compression isn't applicable, no compression is used. If a problem is encountered during generation of the TIFF, TIFFRepresentation throws an exception.

See Also
TIFFRepresentation (page 198) (NSBitmapImageRep)
representationUsingType (page 196) (NSBitmapImageRep)

unlockFocus

Balances a previous lockFocus (page 761) or lockFocusOnRepresentation (page 761) message.

```
public void unlockFocus()
```

Discussion
All successful lockFocus and lockFocusOnRepresentation messages (those that don't throw an ImageCacheException) must be followed by a subsequent unlockFocus message. Those that throw should never be followed by unlockFocus.

usesEPSOnResolutionMismatch

Returns whether EPS representations are preferred when there are no representations that match the resolution of the device.

```
public boolean usesEPSOnResolutionMismatch()
```

Discussion
The default is false.

See Also
setUsesEPSOnResolutionMismatch (page 767)
matchesOnMultipleResolution (page 762)

Constants

The following compositing operators are defined by NSImage and used by compositeToPointFromRect (page 757), compositeToPoint (page 756), compositeToPointFromRectWithFraction (page 757), compositeToPointWithFraction (page 758), drawAtPoint (page 759), and drawInRect (page 759).

The constants are described in terms of having source and destination images, each having an opaque and transparent region. The destination image after the operation is defined in terms of the source and destination before images as follows:

Constant	Description
CompositeClear	Transparent.
CompositeCopy	Source image.
CompositeDestinationAtop	Destination image wherever both images are opaque, source image wherever source image is opaque but destination image is transparent, and transparent elsewhere.
CompositeDestinationIn	Destination image wherever both images are opaque, and transparent elsewhere.
CompositeDestinationOut	Destination image wherever destination image is opaque but source image is transparent, and transparent elsewhere.
CompositeDestinationOver	Destination image wherever destination image is opaque, and source image elsewhere.
CompositeHighlight	Deprecated. Mapped to CompositeSourceOver.
CompositePlusDarker	Sum of source and destination images, with color values approaching 0 as a limit.
CompositePlusLighter	Sum of source and destination images, with color values approaching 1 as a limit.
CompositeSourceAtop	Source image wherever both images are opaque, destination image wherever destination image is opaque but source image is transparent, and transparent elsewhere.
CompositeSourceIn	Source image wherever both images are opaque, and transparent elsewhere.
CompositeSourceOut	Source image wherever source image is opaque but destination image is transparent, and transparent elsewhere.
CompositeSourceOver	Source image wherever source image is opaque, and destination image elsewhere.
CompositeXOR	Exclusive OR of source and destination images. Works only with black and white images and is not recommended for color contexts.

The following constants are status values passed to the incremental loading delegate method `imageDidLoadRepresentation` (page 771).

Constant	Description
`ImageLoadStatusCompleted`	Enough data has been provided to completely decompress the image.
`ImageLoadStatusCancelled`	Image loading was canceled. The image contains the portions of the data that have already been successfully decompressed, if any.
`ImageLoadStatus-InvalidData`	An error occurred during image decompression. The image data is probably corrupt. The image contains the portions of the data that have already been successfully decompressed, if any.
`ImageLoadStatus-UnexpectedEOF`	Not enough data was available for full decompression of the image. The image contains the portions of the data that have already been successfully decompressed, if any.
`ImageLoadStatusReadError`	Not enough data was available for full decompression of the image. The image contains the portions of the data that have already been successfully decompressed, if any.

The follow constants specify the caching policy on a per NSImage basis. The caching policy is set using `cacheMode` (page 756) and `setCacheMode` (page 764).

Constant	Description
`ImageCacheDefault`	Caching is unspecified. Use the image rep's default.
`ImageCacheAlways`	Always generate a cache when drawing.
`ImageCacheBySize`	Cache if cache size is smaller than the original data.
`ImageCacheNever`	Never cache; always draw direct.

For the various types of image reps, the default caching policy is:

Image Rep Class	Default caching policy
NSBitmapImageRep	`ImageCacheBySize`. Cache if bitmap is 32-bits in 16-bit world or greater than 72 dpi.
NSPICTImageRep	`ImageCacheBySize`. Same reasoning as NSBitmapImageRep in the event the PICT contains a bitmap.
NSPDFImageRep	`ImageCacheAlways`

Delegate Methods

imageDidLoadPartOfRepresentation

During incremental loading, this method is called repeatedly to inform the delegate that more of the image *rep* is available.

```
public abstract void imageDidLoadPartOfRepresentation(NSImage image, NSImageRep
    rep, int rows)
```

Discussion
This method is optional; incremental loading will continue if the delegate does not implement it.

Availability
Available in Mac OS X v10.2 and later.

imageDidLoadRepresentation

This method is invoked when *image* has been as fully decompressed as is possible.

```
public abstract void imageDidLoadRepresentation(NSImage image, NSImageRep rep, int
     status)
```

Discussion
The image invokes this method on the delegate to notify it that the NSImageRep *rep* has finished downloading with a final status of *status*.

If an error occurs during downloading or decompression, *status* will be `ImageLoadStatusInvalidData`, `ImageLoadStatusUnexpectedEOF`, or `ImageLoadStatusReadError`. If you cancel the download, *status* will be `ImageLoadStatusCancelled`.

This method is required for incremental loading; the delegate must implement it if incremental loading is desired. You must also set up the instance to be loaded lazily, by initializing it using either Constructor that initializes a file or URL lazily.

Availability
Available in Mac OS X v10.2 and later.

imageDidLoadRepresentationHeader

During incremental loading, the image invokes this method once enough data has been read to determine the size of the image.

```
public abstract void imageDidLoadRepresentationHeader(NSImage image, NSImageRep
    rep)
```

Discussion
At this point, the NSBitmapImageRep *rep* is valid and has storage for the bitmap. The bitmap is filled with the image's background color. This method is optional; incremental loading will continue if the delegate does not implement it.

Availability
Available in Mac OS X v10.2 and later.

imageDidNotDraw

Implemented by the delegate to respond to a message sent by the sender (an NSImage) when the sender was unable, for whatever reason, to composite or lock focus on its image to draw in *aRect*.

```
public abstract NSImage imageDidNotDraw(Object sender, NSRect aRect)
```

Discussion
The delegate can do one of the following:

- Return another NSImage to draw in the sender's place.
- Draw the image itself and return `null`,.
- Simply return `null` to indicate that *sender* should give up the attempt at drawing the image.

imageWillLoadRepresentation

For incremental loading, this method is invoked when you first draw *image* or otherwise require the bitmap data.

```
public abstract void imageWillLoadRepresentation(NSImage image, NSImageRep rep)
```

Discussion
The image download begins immediately after this method returns. This method is optional; incremental loading will continue if the delegate does not implement it.

Availability
Available in Mac OS X v10.2 and later.

NSImageCell

Inherits from	NSCell : NSObject
Implements	NSCoding (NSCell)
Package:	com.apple.cocoa.application
Companion guides	Image Views Matrix Programming Guide for Cocoa Table View Programming Guide

Overview

An NSImageCell displays a single NSImage in a frame. This class provides methods for choosing the frame and for aligning and scaling the image to fit the frame.

The object value of an NSImageCell must be an NSImage, so if you use NSCell's `setObjectValue` (page 328) method, be sure to supply an NSImage as an argument. Because an NSImage doesn't need to be converted for display, you won't use the NSCell methods relating to formatters.

An NSImageCell is usually associated with some kind of NSControl (page 441)—an NSImageView (page 795), an NSMatrix (page 875), or an NSTableView (page 1437).

Tasks

Constructors

`NSImageCell` (page 774)
Creates an empty NSImageCell.

Aligning and Scaling the Image

`imageAlignment` (page 774)
Returns the position of the receiver's image in the frame.

`setImageAlignment` (page 775)
Lets you specify the position of the image in the frame.

`imageScaling` (page 775)
Returns the way the receiver's image alters to fit the frame.

`setImageScaling` (page 775)
   Lets you specify the way the image alters to fit the frame.

Choosing the Frame

`imageFrameStyle` (page 774)
   Returns the style of frame that appears around the image.

`setImageFrameStyle` (page 775)
   Lets you specify the kind of frame that borders the image.

Constructors

NSImageCell

Creates an empty NSImageCell.

```
public NSImageCell()
```

Creates an NSImageCell initialized with *aString* and set to have the cell's default menu.

```
public NSImageCell(String aString)
```

Creates an NSImageCell initialized with *anImage* and set to have the cell's default menu.

```
public NSImageCell(NSImage anImage)
```

Discussion
If *anImage* is `null`, no image is set.

Instance Methods

imageAlignment

Returns the position of the receiver's image in the frame.

```
public int imageAlignment()
```

Discussion
For a list of possible alignments, see "Constants" (page 776).

See Also
`setImageAlignment` (page 775)

imageFrameStyle

Returns the style of frame that appears around the image.

```
public int imageFrameStyle()
```

Discussion

For a list of frame styles, see "Constants" (page 776).

See Also

`setImageFrameStyle` (page 775)

imageScaling

Returns the way the receiver's image alters to fit the frame.

```
public int imageScaling()
```

Discussion

For a list of possible values, see "Constants" (page 776).

See Also

`setImageScaling` (page 775)

setImageAlignment

Lets you specify the position of the image in the frame.

```
public void setImageAlignment(int alignment)
```

Discussion

The possible alignments are listed in "Constants" (page 776).

The default *`alignment`* is `ImageAlignCenter`.

See Also

`imageAlignment` (page 774)

setImageFrameStyle

Lets you specify the kind of frame that borders the image.

```
public void setImageFrameStyle(int frameStyle)
```

Discussion

The possible styles are listed in "Constants" (page 776).

The default *`frameStyle`* is `ImageFrameNone`.

See Also

`imageFrameStyle` (page 774)

setImageScaling

Lets you specify the way the image alters to fit the frame.

```
public void setImageScaling(int scaling)
```

Discussion

The possible values are listed in "Constants" (page 776).

The default *scaling* is `ScaleProportionally`.

See Also

`imageScaling` (page 775)

Constants

These constants allow you to specify the location of the image in the frame and are used by `imageAlignment` (page 774) and `setImageAlignment` (page 775):

- `ImageAlignLeft`
- `ImageAlignRight`
- `ImageAlignCenter`
- `ImageAlignTop`
- `ImageAlignBottom`
- `ImageAlignTopLeft`
- `ImageAlignTopRight`
- `ImageAlignBottomLeft`
- `ImageAlignBottomRight`

These constants allow you to specify the kind of frame bordering the image and are used by `imageFrameStyle` (page 774) and `setImageFrameStyle` (page 775). These constants are obsolete, and are not compliant with the Apple Human Interface Guidelines:

Constant	Description
`ImageFrameNone`	An invisible frame
`ImageFramePhoto`	A thin black outline and a dropped shadow
`ImageFrameGrayBezel`	A gray, concave bezel that makes the image look sunken
`ImageFrameGroove`	A thin groove that looks etched around the image
`ImageFrameButton`	A convex bezel that makes the image stand out in relief, like a button

These constants allow you to specify the way the image alters to fit the frame. They are used by `imageScaling` (page 775) and `setImageScaling` (page 775).

Constant	Description
`ScaleProportionally`	If the image is too large, it shrinks to fit inside the frame. The proportions of the image are preserved.
`ScaleToFit`	The image shrinks or expands, and its proportions distort, until it exactly fits the frame.
`ScaleNone`	The size and proportions of the image don't change. If the frame is too small to display the whole image, the edges of the image are trimmed off.

NSImageRep

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.application
Companion guide	Drawing and Images

Overview

NSImageRep is a semiabstract superclass ("semi" because it has some instance variables and implementation of its own); 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.

Tasks

Constructors

`NSImageRep` (page 782)
Creates an empty NSImageRep with a zero-sized frame rectangle.

Creating an NSImageRep

`imageRepsWithContentsOfFile` (page 783)

`imageRepsWithPasteboard` (page 784)

`imageRepsWithContentsOfURL` (page 784)

`imageRepWithContentsOfFile` (page 785)

`imageRepWithPasteboard` (page 786)

`imageRepWithContentsOfURL` (page 786)

Checking Data Types

canInitWithData (page 782)
Should be overridden in subclasses to return true if the receiver can initialize itself from *data*, and false if it cannot.

canInitWithPasteboard (page 782)
Returns true if the NSImageRep can handle the data represented by *pasteboard*, otherwise returns false.

imageFileTypes (page 782)
Returns an array of Strings representing all file types supported by NSImageRep or one of its subclasses.

imagePasteboardTypes (page 783)
Returns an array of Strings representing all pasteboard types supported by NSImageRep or one of its subclasses.

imageUnfilteredFileTypes (page 786)
Returns an array of Strings representing all file types (extensions) supported by the NSImageRep.

imageUnfilteredPasteboardTypes (page 787)
Returns an array representing all pasteboard types supported by the NSImageRep.

Setting the Size of the Image

setSize (page 792)
Sets the size of the image to *aSize* in units of the base coordinate system.

size (page 792)
Returns the size of the image in units of the base coordinate system.

Specifying Information About the Representation

bitsPerSample (page 788)
Returns the number of bits per sample—that is, the number of bits used to specify each component of data in a pixel. If the receiver is a planar image, this method returns the number of bits per sample per plane.

colorSpaceName (page 788)
Returns the name if the image's color space, or NSGraphics.CalibratedRGBColorSpace if no name has been assigned.

hasAlpha (page 789)
Returns true if the receiver has been informed that the image has a coverage component (alpha), and false if not.

isOpaque (page 790)
Returns true if the receiver is opaque, false otherwise.

pixelsHigh (page 790)
Returns the height of the image in pixels, as specified in the image data.

pixelsWide (page 790)
Returns the width of the image in pixels, as specified in the image data.

`setAlpha` (page 790)
: Informs the receiver whether the image has an alpha component.

`setBitsPerSample` (page 791)
: Informs the receiver that the image has *anInt* bits of data for each pixel in each component.

`setColorSpaceName` (page 791)
: Informs the receiver of the image's color space.

`setOpaque` (page 791)
: Sets opacity of the receiver's image.

`setPixelsHigh` (page 792)
: Informs the receiver that the data specifies an image *anInt* pixels high.

`setPixelsWide` (page 792)
: Informs the receiver that the data specifies an image *anInt* pixels wide.

Drawing the Image

`draw` (page 788)
: Implemented by subclasses to draw the image at location (0.0, 0.0) in the current coordinate system.

`drawAtPoint` (page 789)
: Sets the current coordinates to *aPoint*, invokes the receiver's `draw` method to draw the image at that point, then restores the current coordinates to their original setting.

`drawInRect` (page 789)
: Draws the image so it fits inside *rect*.

Managing NSImageRep Subclasses

`imageRepClassForData` (page 783)
: Returns the NSImageRep subclass that handles data of type *data*, or `null` if the NSImage class registry contains no subclasses that handle data of the specified type.

`imageRepClassForFileType` (page 783)
: Returns the NSImageRep subclass that handles files of type *type*, or `null` if the NSImage class registry contains no subclasses that handle files of the specified type.

`imageRepClassForPasteboardType` (page 783)
: Returns the NSImageRep subclass that handles pasteboard data of type *type*, or `null` if the NSImage class registry contains no subclasses that handle pasteboard data of the specified type.

`registeredImageRepClasses` (page 787)
: Returns an array containing the registered NSImageRep classes.

`registerImageRepClass` (page 787)
: Adds *imageRepClass* to the registry of available NSImageRep classes.

`unregisterImageRepClass` (page 788)
: Removes *imageRepClass* from the registry of available NSImageRep classes.

Constructors

NSImageRep

Creates an empty NSImageRep with a zero-sized frame rectangle.

```
public NSImageRep()
```

Static Methods

canInitWithData

Should be overridden in subclasses to return `true` if the receiver can initialize itself from *`data`*, and `false` if it cannot.

```
public static boolean canInitWithData(NSData data)
```

Discussion
Note that this method doesn't need to do a comprehensive check; it should return `false` only if it knows the receiver can't initialize itself from *`data`*.

See Also
`canInitWithPasteboard` (page 782)

canInitWithPasteboard

Returns `true` if the NSImageRep can handle the data represented by *`pasteboard`*, otherwise returns `false`.

```
public static boolean canInitWithPasteboard(NSPasteboard pasteboard)
```

Discussion
This method invokes the `imageUnfilteredPasteboardTypes` (page 787) class method and checks the list of types returned by that method against the data types in *`pasteboard`*. If it finds a match, it returns `true`. When creating a subclass of NSImageRep that accepts image data from a non-default pasteboard type, override the `imageUnfilteredPasteboardTypes` (page 787) method to assure this method returns the correct response.

See Also
`canInitWithData` (page 782)

imageFileTypes

Returns an array of Strings representing all file types supported by NSImageRep or one of its subclasses.

```
public static NSArray imageFileTypes()
```

Discussion

The list includes both those types returned by the `imageUnfilteredFileTypes` (page 786) class method and those that can be converted to a supported type by a user-installed filter service. The file types returned can include encoded HFS file types as well as filename extensions.

Don't override this method when subclassing NSImageRep—it always returns a valid list for any subclass of NSImageRep that correctly overrides the `imageUnfilteredFileTypes` (page 786) method.

imagePasteboardTypes

Returns an array of Strings representing all pasteboard types supported by NSImageRep or one of its subclasses.

```
public static NSArray imagePasteboardTypes()
```

Discussion

The list includes both those types returned by the `imageUnfilteredPasteboardTypes` (page 787) 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 any subclass of NSImageRep that correctly overrides the `imageUnfilteredPasteboardTypes` (page 787) method.

imageRepClassForData

Returns the NSImageRep subclass that handles data of type *data*, or `null` if the NSImage class registry contains no subclasses that handle data of the specified type.

```
public static Class imageRepClassForData(NSData data)
```

imageRepClassForFileType

Returns the NSImageRep subclass that handles files of type *type*, or `null` if the NSImage class registry contains no subclasses that handle files of the specified type.

```
public static Class imageRepClassForFileType(String type)
```

Discussion

type may be either an encoded HFS file type or a filename extension.

imageRepClassForPasteboardType

Returns the NSImageRep subclass that handles pasteboard data of type *type*, or `null` if the NSImage class registry contains no subclasses that handle pasteboard data of the specified type.

```
public static Class imageRepClassForPasteboardType(String type)
```

imageRepsWithContentsOfFile

```
public static NSArray imageRepsWithContentsOfFile(String filename)
```

Discussion

If sent to the NSImageRep class object, this method returns an array of objects (all newly allocated instances of a subclass of NSImageRep, chosen through the use of `imageRepClassForFileType` (page 783)) that have been initialized with the contents of the file *filename*. If sent to a subclass of NSImageRep that recognizes the type of file specified by *filename*, it returns an array of objects (all instances of that subclass) that have been initialized with the contents of the file *filename*.

The `imageRepsWithContentsOfFile` (page 783) method returns `null` in any of the following cases:

- The message is sent to the NSImageRep class object, and there are no subclasses in the NSImageRep class registry that handle data of the type indicated by *filename*.
- The message is sent to a subclass of NSImageRep, and that subclass doesn't handle data of the type indicated by *filename*.
- The NSImageRep subclass is unable to initialize itself with the contents of *filename*.

The *filename* argument may be a full or relative pathname and should include an extension that identifies the data type in the file. By default, the files handled include those with the extensions "`tiff`", "`gif`", "`jpg`", "`pict`", "`pdf`", and "`eps`".

The NSImageRep subclass is initialized by creating an NSData object based on the contents of the file, then passing it to `imageRepsWithData`.

See Also

`imageFileTypes` (page 782)

imageRepsWithContentsOfURL

```
public static NSArray imageRepsWithContentsOfURL(java.net.URL aURL)
```

Discussion

If sent to the NSImageRep class object, this method returns an array of objects (all newly allocated instances of a subclass of NSImageRep) that have been initialized with the contents of *aURL*. If sent to a subclass of NSImageRep that recognizes the type of data contained in *aURL*, it returns an array of objects (all instances of that subclass) that have been initialized with the contents of *aURL*.

The `imageRepsWithContentsOfURL` (page 784) method returns `null` in any of the following cases:

- The message is sent to the NSImageRep class object, and there are no subclasses in the NSImageRep class registry that handle data of the type contained in *aURL*.
- The message is sent to a subclass of NSImageRep, and that subclass doesn't handle data of the type contained in *aURL*.
- The NSImageRep subclass is unable to initialize itself with the contents of *aURL*.

The NSImageRep subclass is initialized by creating an NSData object based on the contents of *aURL*, then passing it to `imageRepsWithData`.

imageRepsWithPasteboard

```
public static NSArray imageRepsWithPasteboard(NSPasteboard pasteboard)
```

Discussion

If sent to the NSImageRep class object, this method returns an array of objects (all newly-allocated instances of a subclass of NSImageRep) that have been initialized with the data in *pasteboard*. If sent to a subclass of NSImageRep that recognizes the type of data contained in *pasteboard*, it returns an array of objects (all instances of that subclass) initialized with the data in *pasteboard*.

`imageRepsWithPasteboard` returns `null` in any of the following cases:

- The message is sent to the NSImageRep class object, and there are no subclasses in the NSImageRep class registry that handle data of the type contained in *pasteboard*.
- The message is sent to a subclass of NSImageRep, and that subclass doesn't handle data of the type contained in *pasteboard*.
- The NSImageRep subclass is unable to initialize itself with the contents of *pasteboard*.

The NSImageRep subclass is initialized by creating an NSData object based on the data in *pasteboard*, then passing it to `imageRepsWithData`.

See Also

`imagePasteboardTypes` (page 783)

imageRepWithContentsOfFile

```
public static NSImageRep imageRepWithContentsOfFile(String filename)
```

Discussion

If sent to the NSImageRep class object, this method returns a newly allocated instance of a subclass of NSImageRep (chosen through the use of `imageRepClassForFileType` (page 783)) initialized with the contents of the file *filename*. If sent to a subclass of NSImageRep that recognizes the type of file specified by *filename*, it returns an instance of that subclass initialized with the contents of the file *filename*.

The `imageRepWithContentsOfFile` method returns `null` in any of the following cases:

- The message is sent to the NSImageRep class object, and there are no subclasses in the NSImageRep class registry that handle data of the type indicated by *filename*.
- The message is sent to a subclass of NSImageRep, and that subclass doesn't handle data of the type indicated by *filename*.
- The NSImageRep subclass is unable to initialize itself with the contents of *filename*.

The *filename* argument may be a full or relative pathname and should include an extension that identifies the data type in the file. By default, the files handled include those with the extensions "`tiff`", "`gif`", "`jpg`", "`pict`", "`pdf`", and "`eps`".

The NSImageRep subclass is initialized by creating an NSData object based on the contents of the file, then passing it to `imageRepWithData`.

See Also

`imageFileTypes` (page 782)

imageRepWithContentsOfURL

```
public static Object imageRepWithContentsOfURL(java.net.URL aURL)
```

Discussion
If sent to the NSImageRep class object, this method returns a newly allocated instance of a subclass of NSImageRep initialized with the contents of *aURL*. If sent to a subclass of NSImageRep that recognizes the type of data contained in *aURL*, it returns an instance of that subclass initialized with the contents of *aURL*.

The `imageRepWithContentsOfURL` method returns `null` in any of the following cases:

- The message is sent to the NSImageRep class object, and there are no subclasses in the NSImageRep class registry that handle data of the type contained in *aURL*.
- The message is sent to a subclass of NSImageRep, and that subclass doesn't handle data of the type contained in *aURL*.
- The NSImageRep subclass is unable to initialize itself with the contents of *aURL*.

The NSImageRep subclass is initialized by creating an NSData object based on the contents of the file, then passing it to `imageRepWithData`.

imageRepWithPasteboard

```
public static NSImageRep imageRepWithPasteboard(NSPasteboard pasteboard)
```

Discussion
If sent to the NSImageRep class object, this method returns a newly allocated instance of a subclass of NSImageRep initialized with the data in *pasteboard*. If sent to a subclass of NSImageRep that recognizes the type of data contained in *pasteboard*, it returns an instance of that subclass initialized with the data in *pasteboard*.

The `imageRepWithPasteboard` method returns `null` in any of the following cases:

- The message is sent to the NSImageRep class object, and there are no subclasses in the NSImageRep class registry that handle data of the type contained in *pasteboard*.
- The message is sent to a subclass of NSImageRep, and that subclass doesn't handle data of the type contained in *pasteboard*.
- The NSImageRep subclass is unable to initialize itself with the contents of *pasteboard*.

The NSImageRep subclass is initialized by creating an NSData object based on the data in *pasteboard*, then passing it to `imageRepWithData`.

See Also
`imagePasteboardTypes` (page 783)

imageUnfilteredFileTypes

Returns an array of Strings representing all file types (extensions) supported by the NSImageRep.

```
public static NSArray imageUnfilteredFileTypes()
```

Discussion
The file types returned can include encoded HFS file types as well as filename extensions. By default, the returned array is empty.

When creating a subclass of NSImageRep, override this method to return a list of strings representing the supported file 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
`imageFileTypes` (page 782)
`imageUnfilteredFileTypes` (page 753) (NSImage)

imageUnfilteredPasteboardTypes

Returns an array representing all pasteboard types supported by the NSImageRep.

```
public static NSArray imageUnfilteredPasteboardTypes()
```

Discussion
By default, the returned array is empty.

When creating a subclass of NSImageRep, override this method to return a list representing the supported pasteboard 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
`imagePasteboardTypes` (page 783)
`imageUnfilteredPasteboardTypes` (page 754) (NSImage)

registeredImageRepClasses

Returns an array containing the registered NSImageRep classes.

```
public static NSArray registeredImageRepClasses()
```

registerImageRepClass

Adds *imageRepClass* to the registry of available NSImageRep classes.

```
public static void registerImageRepClass(Class imageRepClass)
```

Discussion
This method posts an `ImageRepRegistryDidChangeNotification` (page 793), along with the receiving object, to the default notification center.

See Also
`unregisterImageRepClass` (page 788)

unregisterImageRepClass

Removes *imageRepClass* from the registry of available NSImageRep classes.

```
public static void unregisterImageRepClass(Class imageRepClass)
```

Discussion
This method posts the `ImageRepRegistryDidChangeNotification` (page 793), along with the receiving object, to the default notification center.

See Also
`registerImageRepClass` (page 787)

Instance Methods

bitsPerSample

Returns the number of bits per sample—that is, the number of bits used to specify each component of data in a pixel. If the receiver is a planar image, this method returns the number of bits per sample per plane.

```
public int bitsPerSample()
```

See Also
`setBitsPerSample` (page 791)
`bitsPerPixel` (page 193) (NSBitmapImageRep)
`samplesPerPixel` (page 197) (NSBitmapImageRep)
`isPlanar` (page 196) (NSBitmapImageRep)

colorSpaceName

Returns the name if the image's color space, or `NSGraphics.CalibratedRGBColorSpace` if no name has been assigned.

```
public String colorSpaceName()
```

See Also
`setColorSpaceName` (page 791)

draw

Implemented by subclasses to draw the image at location (0.0, 0.0) in the current coordinate system.

```
public boolean draw()
```

Discussion
Subclass methods return `true` if the image is successfully drawn, and `false` if it isn't. This version of the method simply returns `true`.

The standard Application Kit subclasses all draw the image using the `NSImage.CompositeCopy` composite operation defined in NSImage's "Constants" (page 769) section. The image data overwrites the destination without any blending effects; any transparent (alpha) regions in the source image appear black. To use other composite operations, you must place the representation into an NSImage object and use its `drawAtPoint` (page 759) or `drawInRect` (page 759) methods.

drawAtPoint

Sets the current coordinates to *aPoint*, invokes the receiver's `draw` method to draw the image at that point, then restores the current coordinates to their original setting.

```
public boolean drawAtPoint(NSPoint aPoint)
```

Discussion

If *aPoint* is (0.0, 0.0), `drawAtPoint` simply invokes `draw` (page 788).

This method returns `false` without translating, scaling, 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.

See Also

`setSize` (page 792)
`drawInRect` (page 789)

drawInRect

Draws the image so it fits inside *rect*.

```
public boolean drawInRect(NSRect rect)
```

Discussion

The current coordinates are set to the point specified in the rectangle and are scaled so the image will fit within the rectangle. The receiver's `draw` method is then invoked to draw the image. After `draw` has been invoked, the current coordinates and scale factors are restored to their original settings.

This method returns `false` without translating, scaling, 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.

See Also

`setSize` (page 792)
`drawAtPoint` (page 789)

hasAlpha

Returns `true` if the receiver has been informed that the image has a coverage component (alpha), and `false` if not.

```
public boolean hasAlpha()
```

See Also

`setAlpha` (page 790)

isOpaque

Returns `true` if the receiver is opaque, `false` otherwise.

```
public boolean isOpaque()
```

Discussion

Use this method to test whether an NSImageRep completely covers the area within the rectangle returned by `size` (page 792).

This value does not indicate if the image has an alpha channel and if there is partial or complete transparency when drawing the image rep. Use `hasAlpha` (page 789) to determine if the image has an alpha channel.

See Also

`setOpaque` (page 791)

pixelsHigh

Returns the height of the image in pixels, as specified in the image data.

```
public int pixelsHigh()
```

See Also

`setPixelsHigh` (page 792)

`pixelsWide` (page 790)

`size` (page 792)

pixelsWide

Returns the width of the image in pixels, as specified in the image data.

```
public int pixelsWide()
```

See Also

`setPixelsWide` (page 792)

`pixelsHigh` (page 790)

`size` (page 792)

setAlpha

Informs the receiver whether the image has an alpha component.

```
public void setAlpha(boolean flag)
```

Discussion

`flag` should be `true` if it does, and `false` if it doesn't.

See Also
`hasAlpha` (page 789)

setBitsPerSample

Informs the receiver that the image has *`anInt`* bits of data for each pixel in each component.

```
public void setBitsPerSample(int anInt)
```

See Also
`bitsPerSample` (page 788)

setColorSpaceName

Informs the receiver of the image's color space.

```
public void setColorSpaceName(String string)
```

Discussion
By default, an NSImageRep's color space name is `NSGraphics.CalibratedRGBColorSpace`. Color space names are defined as part of the NSGraphics class. The following are valid color space names:

```
NSGraphics.CalibratedWhiteColorSpace
NSGraphics.CalibratedBlackColorSpace
NSGraphics.CalibratedRGBColorSpace
NSGraphics.DeviceWhiteColorSpace
NSGraphics.DeviceBlackColorSpace
NSGraphics.DeviceRGBColorSpace
NSGraphics.DeviceCMYKColorSpace
NSGraphics.NamedColorSpace
NSGraphics.CustomColorSpace
```

See Also
`colorSpaceName` (page 788)

setOpaque

Sets opacity of the receiver's image.

```
public void setOpaque(boolean flag)
```

Discussion
If *`flag`* is `true`, the image is opaque.

See Also
`isOpaque` (page 790)

setPixelsHigh

Informs the receiver that the data specifies an image *anInt* pixels high.

```
public void setPixelsHigh(int anInt)
```

See Also

`pixelsHigh` (page 790)

`setPixelsWide` (page 792)

`setSize` (page 792)

setPixelsWide

Informs the receiver that the data specifies an image *anInt* pixels wide.

```
public void setPixelsWide(int anInt)
```

See Also

`pixelsWide` (page 790)

`setPixelsHigh` (page 792)

`setSize` (page 792)

setSize

Sets the size of the image to *aSize* in units of the base coordinate system.

```
public void setSize(NSSize aSize)
```

Discussion

This method 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. You must set the image size before you can render it.

See Also

`size` (page 792)

`draw` (page 788)

`setPixelsHigh` (page 792)

`setPixelsWide` (page 792)

size

Returns the size of the image in units of the base coordinate system.

```
public NSSize size()
```

Discussion

This size is 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

`setSize` (page 792)

`pixelsHigh` (page 790)
`pixelsWide` (page 790)

Constants

The following constant is defined as a convenience by NSImageRep:

Constant	Description
`ImageRepMatchesDevice`	Indicates that the value of certain attributes, such as the number of colors or bits per sample, will change to match the display device.

Notifications

ImageRepRegistryDidChangeNotification

Posted whenever the NSImageRep class registry changes.

The notification object is the image class that is registered or unregistered. This notification does not contain a *userInfo* dictionary.

NSImageView

Inherits from NSControl : NSView : NSResponder : NSObject

Implements NSCoding (NSResponder)

Package: com.apple.cocoa.application

Companion guide Image Views

Overview

An NSImageView displays a single NSImage in a frame and can optionally allow a user to drag an image to it.

Tasks

Constructors

`NSImageView` (page 796)
Creates an NSImageView with a zero-sized frame rectangle.

Choosing the Image

`image` (page 797)
Returns the `NSImage` displayed by the receiver.

`setImage` (page 799)
Lets you specify the image the receiver displays as *`image`*.

Choosing the Frame

`imageFrameStyle` (page 798)
Returns the style of frame that appears around the image.

`setImageFrameStyle` (page 800)
Lets you specify the kind of frame that borders the image.

Aligning and Scaling the Image

`imageAlignment` (page 797)
Returns the position of the cell's image in the frame.

`setImageAlignment` (page 799)
Lets you specify the position of the image in the frame.

`imageScaling` (page 798)
Returns the way the cell's image alters to fit the frame.

`setImageScaling` (page 800)
Lets you specify the way the image alters to fit the frame.

Responding to User Events

`isEditable` (page 798)
Returns whether the user can drag a new image into the frame.

`setEditable` (page 799)
Specifies whether the user can drag a new image into the frame, depending on the Boolean value *flag*.

Animating Image Playback

`animates` (page 797)
Returns whether the receiver automatically plays an animated image that is assigned to it.

`setAnimates` (page 798)
Specifies whether the receiver automatically plays an animated image that is assigned to it.

Pasteboard Support

`setAllowsCutCopyPaste` (page 798)
Specifies whether the receiver allows the user to cut, copy and paste the image contents.

`allowsCutCopyPaste` (page 797)
Returns whether the receiver allows the user to cut, copy and paste of the image contents.

Constructors

NSImageView

Creates an NSImageView with a zero-sized frame rectangle.

```
public NSImageView()
```

Creates an NSImageView with *frameRect* as its frame rectangle.

```
public NSImageView(NSRect frameRect)
```

Instance Methods

allowsCutCopyPaste

Returns whether the receiver allows the user to cut, copy and paste of the image contents.

```
public boolean allowsCutCopyPaste()
```

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

See Also
`setAllowsCutCopyPaste` (page 798)

animates

Returns whether the receiver automatically plays an animated image that is assigned to it.

```
public boolean animates()
```

Discussion
If this method returns `true`, the receiver automatically plays animated images, with the timing and looping characteristics specified by the image data.

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

See Also
`setAnimates` (page 798)

image

Returns the `NSImage` displayed by the receiver.

```
public NSImage image()
```

See Also
`setImage` (page 799)

imageAlignment

Returns the position of the cell's image in the frame.

```
public int imageAlignment()
```

Discussion
For a list of possible alignments, see `setImageAlignment` (page 799).

imageFrameStyle

Returns the style of frame that appears around the image.

```
public int imageFrameStyle()
```

Discussion

For a list of frame styles, see `setImageFrameStyle` (page 800).

imageScaling

Returns the way the cell's image alters to fit the frame.

```
public int imageScaling()
```

Discussion

For a list of possible values, see `setImageScaling` (page 800).

isEditable

Returns whether the user can drag a new image into the frame.

```
public boolean isEditable()
```

Discussion

The default is `true`.

See Also

`setEditable` (page 799)

setAllowsCutCopyPaste

Specifies whether the receiver allows the user to cut, copy and paste the image contents.

```
public void setAllowsCutCopyPaste(boolean allow)
```

Availability

Available in Mac OS X v10.4 and later.

See Also

`allowsCutCopyPaste` (page 797)

setAnimates

Specifies whether the receiver automatically plays an animated image that is assigned to it.

```
public void setAnimates(boolean flag)
```

Discussion

If *flag* is set to true, the receiver automatically plays animated images, with the timing and looping characteristics specified by the image data. If *flag* is set to false, the receiver displays the first frame of the animation. The default is true for newly created instances of NSImageView and false for previously-created NSImageView objects loaded from nib files.

Availability

Available in Mac OS X v10.3 and later.

See Also

animates (page 797)

setEditable

Specifies whether the user can drag a new image into the frame, depending on the Boolean value *flag*.

```
public void setEditable(boolean flag)
```

See Also

isEditable (page 798)

setImage

Lets you specify the image the receiver displays as *image*.

```
public void setImage(NSImage image)
```

See Also

image (page 797)

setImageAlignment

Lets you specify the position of the image in the frame.

```
public void setImageAlignment(int alignment)
```

Discussion

The possible values for *alignment* are:

- NSImageCell.ImageAlignLeft
- NSImageCell.ImageAlignRight
- NSImageCell.ImageAlignCenter
- NSImageCell.ImageAlignTop
- NSImageCell.ImageAlignBottom
- NSImageCell.ImageAlignTopLeft
- NSImageCell.ImageAlignTopRight
- NSImageCell.ImageAlignBottomLeft

- `NSImageCell.ImageAlignBottomRight`

The default alignment is `NSImageCell.ImageAlignCenter`.

See Also
`imageAlignment` (page 797)

setImageFrameStyle

Lets you specify the kind of frame that borders the image.

```
public void setImageFrameStyle(int frameStyle)
```

Discussion
The possible values for *`frameStyle`* are:

- `NSImageCell.ImageFrameNone`—an invisible frame
- `NSImageCell.ImageFramePhoto`—a thin black outline and a dropped shadow
- `NSImageCell.ImageFrameGrayBezel`—a gray, concave bezel that makes the image look sunken
- `NSImageCell.ImageFrameGroove`—a thin groove that looks etched around the image
- `NSImageCell.ImageFrameButton`—a convex bezel that makes the image stand out in relief, like a button

The default *`frameStyle`* is `NSImageCell.ImageFrameNone`.

See Also
`imageFrameStyle` (page 798)

setImageScaling

Lets you specify the way the image alters to fit the frame.

```
public void setImageScaling(int scaling)
```

Discussion
The possible values for *`scaling`* are:

- `NSImageCell.ScaleProportionally`. If the image is too large, it shrinks to fit inside the frame. The proportions of the image are preserved. The image is never scaled up to fit a larger frame.
- `NSImageCell.ScaleToFit`. The image shrinks or expands, and its proportions distort, until it exactly fits the frame.
- `NSImageCell.ScaleNone`. The size and proportions of the image don't change. If the frame is too small to display the whole image, the edges of the image are trimmed off.

The default scaling is `NSImageCell.ScaleProportionally`.

See Also
`imageScaling` (page 798)

NSInputManager

Inherits from	NSObject
Implements	NSTextInput
Package:	com.apple.cocoa.application
Companion guide	Text Input Management

Overview

NSInputManager is one of the three players in the Cocoa text input management system. The input manager acts as a proxy between "NSInputServer" (page 809) and text views. You never have to instantiate or subclass NSInputManager, and unless you are implementing a text view that does not inherit from NSTextView, you never have to directly access its methods either.

If an application needs more complex text handling than the standard Cocoa text view classes can provide, then it will use its own text view class that implements the "NSTextInput" (page 2025) interface, and that class will call NSInputManager's methods. The current input manager's input server will call some of the text view's NSTextInput interface methods.

The `wantsToDelayTextChangeNotifications` (page 807), `wantsToHandleMouseEvents` (page 808), and `wantsToInterpretAllKeystrokes` (page 808) methods call methods of the same names on the input manager's current input server and return the result.

Interfaces Implemented

NSTextInput
- `characterIndexForPoint` (page 2026)
- `conversationIdentifier` (page 2027)
- `doCommandBySelector` (page 2027)
- `firstRectForCharacterRange` (page 2027)
- `hasMarkedText` (page 2027)
- `insertText` (page 2028)
- `markedRange` (page 2028)
- `selectedRange` (page 2028)
- `setMarkedTextAndSelectedRange` (page 2028)
- `unmarkText` (page 2029)
- `validAttributesForMarkedText` (page 2029)

Tasks

Constructors

NSInputManager (page 804)
Cocoa constructs an NSInputManager automatically. This method is an implementation detail. You will never call this method.

Input Server Selection

currentInputManager (page 804)
The current input manager is the one that has been chosen to handle keyboard events at the time this method is called.

cycleToNextInputLanguage (page 804)
Deprecated.

cycleToNextInputServerInLanguage (page 804)
Deprecated.

Marked Text

markedTextSelectionChanged (page 806)
The user clicked at the beginning of marked text, the end of marked text, or in between, or the user made a selection within the marked text in the *client* text view.

markedTextAbandoned (page 806)
The input server must abandon whatever it was doing with marked text.

Event Handling

handleMouseEvent (page 805)

Input Server Information

image (page 805)
Deprecated.

language (page 806)
Returns the Language property from the input server's Info file, or null if none is specified there.

localizedInputManagerName (page 806)
Returns the name of the input server as it appears in the Edit>Input submenu.

server (page 807)
Deprecated.

Input Server Options

`wantsToDelayTextChangeNotifications` (page 807)
A `true` return value tells the sender that only a call to its `insertText` (page 1980) method constitutes a modification to its text storage.

`wantsToHandleMouseEvents` (page 808)
Returns `true` if the sender should forward all mouse events within the text view to the input server.

`wantsToInterpretAllKeystrokes` (page 808)
Returns `true` if the server wants all keystrokes to be sent to it as characters.

NSTextInput Interface (used Internally Only)

`attributedSubstringWithRange` (page 804)
From NSTextInput interface; called internally only.

`characterIndexForPoint` (page 805)
From NSTextInput interface; called internally only.

`conversationIdentifier` (page 805)
From NSTextInput interface; called internally only.

`doCommandBySelector` (page 805)
From NSTextInput interface; called internally only.

`firstRectForCharacterRange` (page 805)
From NSTextInput interface; called internally only.

`hasMarkedText` (page 805)
From NSTextInput interface; called internally only.

`insertText` (page 806)
From NSTextInput interface; called internally only.

`markedRange` (page 806)
From NSTextInput interface; called internally only.

`selectedRange` (page 807)
From NSTextInput interface; called internally only.

`setMarkedTextAndSelectedRange` (page 807)
From NSTextInput interface; called internally only.

`unmarkText` (page 807)
From NSTextInput interface; called internally only.

`validAttributesForMarkedText` (page 807)
From NSTextInput interface; called internally only.

Constructors

NSInputManager

Cocoa constructs an NSInputManager automatically. This method is an implementation detail. You will never call this method.

```
public NSInputManager(String inputServerName, String hostName)
```

Static Methods

currentInputManager

The current input manager is the one that has been chosen to handle keyboard events at the time this method is called.

```
public static NSInputManager currentInputManager()
```

Discussion
Don’t cache the return value, because the user can switch to a different input manager at any time.

cycleToNextInputLanguage

Deprecated.

```
public static void cycleToNextInputLanguage(Object sender)
```

cycleToNextInputServerInLanguage

Deprecated.

```
public static void cycleToNextInputServerInLanguage(Object sender)
```

Instance Methods

attributedSubstringWithRange

From NSTextInput interface; called internally only.

```
public NSAttributedString attributedSubstringWithRange(NSRange theRange)
```

characterIndexForPoint

From NSTextInput interface; called internally only.

```
public int characterIndexForPoint(NSPoint thePoint)
```

conversationIdentifier

From NSTextInput interface; called internally only.

```
public int conversationIdentifier()
```

doCommandBySelector

From NSTextInput interface; called internally only.

```
public void doCommandBySelector(NSSelector aSelector)
```

firstRectForCharacterRange

From NSTextInput interface; called internally only.

```
public NSRect firstRectForCharacterRange(NSRange theRange)
```

handleMouseEvent

```
public boolean handleMouseEvent(NSEvent theMouseEvent)
```

Discussion

Forwards a mouse event passed in *theMouseEvent* to the input server. If `wantsToHandleMouseEvents` (page 808) returns `true`, then the text view must forward all mouse events that occur within it. As usual, a return value of `false` means that the text view should proceed with handling the event.

hasMarkedText

From NSTextInput interface; called internally only.

```
public boolean hasMarkedText()
```

image

Deprecated.

```
public NSImage image()
```

insertText

From NSTextInput interface; called internally only.

```
public void insertText(Object aString)
```

language

Returns the `Language` property from the input server's `Info` file, or `null` if none is specified there.

```
public String language()
```

Discussion

For additional information, see "Deploying Input Servers".

localizedInputManagerName

Returns the name of the input server as it appears in the Edit>Input submenu.

```
public String localizedInputManagerName()
```

Discussion

This value comes from the input server's `Info` file.For additional information, see "Deploying Input Servers".

markedRange

From NSTextInput interface; called internally only.

```
public NSRange markedRange()
```

markedTextAbandoned

The input server must abandon whatever it was doing with marked text.

```
public void markedTextAbandoned(Object client)
```

Discussion

The `NSTextView` object *`client`* calls this when the user clicks outside the marked text (anywhere other than the beginning of marked text, the end of marked text, or in between), then `NSTextView` promotes the marked text to normal text as if it had been inserted. A custom text view is free to choose not to keep the marked text.

See Also

`markedTextSelectionChanged` (page 806)

`markedTextAbandoned` (page 1981) (NSInputServiceProvider)

markedTextSelectionChanged

The user clicked at the beginning of marked text, the end of marked text, or in between, or the user made a selection within the marked text in the *`client`* text view.

```
public void markedTextSelectionChanged(NSRange newSel, Object client)
```

Discussion

The range `newSel` is relative to the beginning of the marked text.

See Also

`markedTextAbandoned` (page 806)

`markedTextSelectionChanged` (page 1981) (NSInputServiceProvider)

selectedRange

From NSTextInput interface; called internally only.

```
public NSRange selectedRange()
```

server

Deprecated.

```
public NSInputServer server()
```

setMarkedTextAndSelectedRange

From NSTextInput interface; called internally only.

```
public void setMarkedTextAndSelectedRange(Object aString, NSRange selRange)
```

unmarkText

From NSTextInput interface; called internally only.

```
public void unmarkText()
```

validAttributesForMarkedText

From NSTextInput interface; called internally only.

```
public NSArray validAttributesForMarkedText()
```

wantsToDelayTextChangeNotifications

A `true` return value tells the sender that only a call to its `insertText` (page 1980) method constitutes a modification to its text storage.

```
public boolean wantsToDelayTextChangeNotifications()
```

Discussion

A `false` return value tells the sender that all text given to it, either by insertion or as part of marked text, should constitute a modification to its text storage. The sender may for example want to filter all text that is part of a modification but leave marked text unfiltered.

See Also

`wantsToDelayTextChangeNotifications` (page 807) (NSInputServiceProvider)

wantsToHandleMouseEvents

Returns `true` if the sender should forward all mouse events within the text view to the input server.

```
public boolean wantsToHandleMouseEvents()
```

See Also

`wantsToHandleMouseEvents` (page 1982) (NSInputServiceProvider)

wantsToInterpretAllKeystrokes

Returns `true` if the server wants all keystrokes to be sent to it as characters.

```
public boolean wantsToInterpretAllKeystrokes()
```

Discussion

This method is needed only by the inner workings of Cocoa. You will probably not need to call this method.

See Also

`wantsToInterpretAllKeystrokes` (page 1982) (NSInputServiceProvider)

NSInputServer

Inherits from	NSObject
Implements	NSInputServiceProvider NSInputServerMouseTracker
Package:	com.apple.cocoa.application
Companion guide	Text Input Management

Overview

NSInputServer and NSInputManager (page 801) are central to the mechanism that interprets keystrokes and delivers text characters to text view objects. NSInputServer objects provide the direct interface between the user and the text management system, communicating to text views via NSInputManager.

Interfaces Implemented

NSInputServiceProvider

- `activeConversationChanged` (page 1978)
- `activeConversationWillChange` (page 1978)
- `canBeDisabled` (page 1979)
- `doCommandBySelector` (page 1979)
- `inputClientBecomeActive` (page 1979)
- `inputClientDisabled` (page 1980)
- `inputClientEnabled` (page 1980)
- `inputClientResignActive` (page 1980)
- `insertText` (page 1980)
- `markedTextAbandoned` (page 1981)
- `markedTextSelectionChanged` (page 1981)
- `terminate` (page 1981)
- `wantsToDelayTextChangeNotifications` (page 1981)
- `wantsToHandleMouseEvents` (page 1982)
- `wantsToInterpretAllKeystrokes` (page 1982)

NSInputServerMouseTracker

- `mouseDownOnCharacterIndex` (page 1975)
- `mouseDraggedOnCharacterIndex` (page 1976)

`mouseUpOnCharacterIndex` (page 1976)

Tasks

Constructors

`NSInputServer` (page 811)
Constructs an input server with optional *`delegate`* and with *`name`*.

NSInputServiceProvider Interface Implementations

`activeConversationChanged` (page 811)
Forwards to the delegate if provided.

`activeConversationWillChange` (page 811)
Forwards to the delegate if provided.

`canBeDisabled` (page 812)
Returns `false` or forwards to the delegate if provided.

`doCommandBySelector` (page 812)
Forwards to the delegate if provided.

`inputClientBecameActive` (page 812)
Forwards to the delegate if provided.

`inputClientDisabled` (page 812)
Forwards to the delegate if provided.

`inputClientEnabled` (page 812)
Forwards to the delegate if provided.

`inputClientResignActive` (page 812)
Forwards to the delegate if provided.

`insertText` (page 813)
Forwards to the delegate if provided.

`markedTextAbandoned` (page 813)
Forwards to the delegate if provided.

`markedTextSelectionChanged` (page 813)
Forwards to the delegate if provided.

`terminate` (page 814)
Forwards to the delegate if provided.

`wantsToDelayTextChangeNotifications` (page 814)
Returns `false` or forwards to the delegate if provided.

`wantsToHandleMouseEvents` (page 814)
Returns `false` or forwards to the delegate if provided.

`wantsToInterpretAllKeystrokes` (page 814)
Returns `false` or forwards to the delegate if provided.

NSInputServerMouseTracker Interface Implementations

`mouseDownOnCharacterIndex` (page 813)
: Returns `false` or forwards to the delegate if provided.

`mouseDraggedOnCharacterIndex` (page 813)
: Returns `false` or forwards to the delegate if provided.

`mouseUpOnCharacterIndex` (page 814)
: Returns `false` or forwards to the delegate if provided.

Constructors

NSInputServer

Constructs an input server with optional *delegate* and with *name*.

```
public NSInputServer(Object delegate, String name)
```

Discussion

The given *name* identifies this service in the IPC mechanism, so NSInputManager (page 801) can find it. If *delegate* is `null`, then the methods in this class must be overridden in a subclass of NSInputServer. If *delegate* is non-`null`, then all methods forward to the *delegate*, which must implement the NSInputServiceProvider (page 1977) interface and which may need to implement the NSInputServerMouseTracker (page 1975) interface.

Instance Methods

activeConversationChanged

Forwards to the delegate if provided.

```
public void activeConversationChanged(Object anObject, int anInt)
```

See Also

`activeConversationChanged` (page 1978) (NSInputServiceProvider)

activeConversationWillChange

Forwards to the delegate if provided.

```
public void activeConversationWillChange(Object anObject, int anInt)
```

See Also

`activeConversationWillChange` (page 1978) (NSInputServiceProvider)

canBeDisabled

Returns `false` or forwards to the delegate if provided.

```
public boolean canBeDisabled()
```

See Also

`canBeDisabled` (page 1979) (NSInputServiceProvider)

doCommandBySelector

Forwards to the delegate if provided.

```
public void doCommandBySelector(NSSelector aSelector, Object anObject)
```

See Also

`doCommandBySelector` (page 1979) (NSInputServiceProvider)

inputClientBecameActive

Forwards to the delegate if provided.

```
public void inputClientBecameActive(Object anObject)
```

See Also

`inputClientBecomeActive` (page 1979) (NSServiceProvider)

inputClientDisabled

Forwards to the delegate if provided.

```
public void inputClientDisabled(Object anObject)
```

See Also

`inputClientDisabled` (page 1980) (NSInputServiceProvider)

inputClientEnabled

Forwards to the delegate if provided.

```
public void inputClientEnabled(Object anObject)
```

See Also

`inputClientEnabled` (page 1980) (NSInputServiceProvider)

inputClientResignActive

Forwards to the delegate if provided.

```
public void inputClientResignActive(Object anObject)
```

See Also

inputClientResignActive (page 1980) (NSInputServiceProvider)

insertText

Forwards to the delegate if provided.

```
public void insertText(Object anObject, Object anObject)
```

See Also

insertText (page 1980) (NSInputServiceProvider)

markedTextAbandoned

Forwards to the delegate if provided.

```
public void markedTextAbandoned(Object anObject)
```

See Also

markedTextAbandoned (page 1981) (NSInputServiceProvider)

markedTextSelectionChanged

Forwards to the delegate if provided.

```
public void markedTextSelectionChanged(NSRange aRange, Object anObject)
```

See Also

markedTextSelectionChanged (page 1981) (NSInputServiceProvider)

mouseDownOnCharacterIndex

Returns `false` or forwards to the delegate if provided.

```
public boolean mouseDownOnCharacterIndex(int anInt, NSPoint aPoint, int anInt,
    Object anObject)
```

See Also

mouseDownOnCharacterIndex (page 1975) (NSInputServiceMouseTracker)

mouseDraggedOnCharacterIndex

Returns `false` or forwards to the delegate if provided.

```
public boolean mouseDraggedOnCharacterIndex(int anInt, NSPoint aPoint, int anInt,
     Object anObject)
```

See Also

mouseDraggedOnCharacterIndex (page 1976) (NSInputServiceMouseTracker)

mouseUpOnCharacterIndex

Returns `false` or forwards to the delegate if provided.

```
public void mouseUpOnCharacterIndex(int anInt, NSPoint aPoint, int anInt, Object
    anObject)
```

See Also

`mouseUpOnCharacterIndex` (page 1976) (NSInputServerMouseTracker)

terminate

Forwards to the delegate if provided.

```
public void terminate(Object sender)
```

See Also

`terminate` (page 1981) (NSInputServiceProvider)

wantsToDelayTextChangeNotifications

Returns `false` or forwards to the delegate if provided.

```
public boolean wantsToDelayTextChangeNotifications()
```

See Also

`wantsToDelayTextChangeNotifications` (page 1981) (NSInputServiceProvider)

wantsToHandleMouseEvents

Returns `false` or forwards to the delegate if provided.

```
public boolean wantsToHandleMouseEvents()
```

See Also

`wantsToHandleMouseEvents` (page 1982) (NSInputServiceProvider)

wantsToInterpretAllKeystrokes

Returns `false` or forwards to the delegate if provided.

```
public boolean wantsToInterpretAllKeystrokes()
```

See Also

`wantsToInterpretAllKeystrokes` (page 1982) (NSInputServiceProvider)

NSInterfaceStyle

Inherits from NSObject

Package: com.apple.cocoa.application

Overview

This class defines parameterized values that allow applications to adhere to the UI conventions of the current operating-system environment. Only the Macintosh interface style is supported.

Tasks

Constructors

`NSInterfaceStyle` (page 815)
Constructor. Instantiating an NSInterfaceStyle is useless, because all methods are static.

Getting an Interface Style

`interfaceStyleForKey` (page 816)

Constructors

NSInterfaceStyle

Constructor. Instantiating an NSInterfaceStyle is useless, because all methods are static.

```
public NSInterfaceStyle()
```

Static Methods

interfaceStyleForKey

```
public static int interfaceStyleForKey(String aString, NSResponder aResponder)
```

Discussion

Responders can use this function to parameterize their drawing and behavior. If the responder has specific defaults to control various aspects of its interface individually, the keys for those special settings can be passed in *aString*; otherwise pass `null` to get the global setting. The responder should always be passed in *aResponder*, but in situations where a responder is not available, pass `null`.

Constants

You can pass the following style defaults to `interfaceStyleForKey` (page 816):

Constant	Description
`InterfaceStyleDefault`	Pass this to retrieve the default interface style.
`AlertInterfaceStyleDefault`	Deprecated.
`BrowserInterfaceStyleDefault`	Deprecated.
`ControlInterfaceStyleDefault`	Deprecated.
`MenuInterfaceStyleDefault`	Deprecated.
`MiniWindowStyleDefault`	Deprecated.
`PopUpButtonInterfaceStyleDefault`	Deprecated.
`ScrollerArrowPositioningDefault`	Deprecated.
`ScrollerInterfaceStyleDefault`	Deprecated.
`ScrollerPagingBehaviorDefault`	Deprecated.
`SystemColorInterfaceStyleDefault`	Deprecated.
`TableViewInterfaceStyleDefault`	Deprecated.
`WindowInterfaceStyleDefault`	Deprecated.
`MacintoshInterfaceStyleDefaultValue`	Macintosh.
`Windows95InterfaceStyleDefaultValue`	Windows 95.

The following values are returned by `interfaceStyleForKey` (page 816):

Constant	Description
`MacintoshInterfaceStyle`	Conforms to Mac OS interface standards.
`NoInterfaceStyle`	No particular interface style specified.
`Windows95InterfaceStyle`	Conforms to Microsoft Windows 95 interface standards.

NSLayoutManager

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.application
Companion guides	Text System Overview Text Layout Programming Guide for Cocoa

Overview

An NSLayoutManager coordinates the layout and display of characters held in an NSTextStorage object. It maps Unicode character codes to glyphs, sets the glyphs in a series of NSTextContainers, and displays them in a series of NSTextViews. In addition to its core function of laying out text, an NSLayoutManager coordinates its NSTextViews, provides services to those NSTextViews to support NSRulerViews for editing paragraph styles, and handles the layout and display of text attributes not inherent in glyphs (such as underline or strikethrough). You can create a subclass of NSLayoutManager to handle additional text attributes, whether inherent or not.

NSLayoutManager now provides the threshold for text antialiasing. It looks at the `AppleAntiAliasingThreshold` default value. If the font size is smaller than or equal to this threshold size, the text is rendered aliased by NSLayoutManager. You can change the threshold value from the System Preferences application's General pane.

Tasks

Constructors

`NSLayoutManager` (page 827)
Creates an empty NSLayoutManager.

Setting the Text Storage

`setTextStorage` (page 850)
Sets the receiver's NSTextStorage to *textStorage*.

`textStorage` (page 853)
Returns the receiver's NSTextStorage.

`replaceTextStorage` (page 844)
: Replaces the NSTextStorage for the group of text-system objects containing the receiver with *newTextStorage*.

Setting Text Containers

`textContainers` (page 853)
: Returns the receiver's NSTextContainers.

`addTextContainer` (page 827)
: Appends *aTextContainer* to the series of NSTextContainers where the receiver arranges text.

`insertTextContainerAtIndex` (page 837)
: Inserts *aTextContainer* into the series of text containers at *index* and invalidates layout for all subsequent NSTextContainer's.

`removeTextContainerAtIndex` (page 843)
: Removes the NSTextContainer at *index* and invalidates the layout as needed.

Invalidating Glyphs and Layout

`invalidateGlyphsForCharacterRange` (page 838)
: Invalidates the cached glyphs for the characters in *charRange* and adjusts the remaining glyph-to-character mapping according to *lengthChange*, which indicates the number of characters added to or removed from the text store.

`invalidateLayoutForCharacterRange` (page 838)
: Invalidates the layout information for the glyphs mapped to the characters in *charRange*.

`invalidateDisplayForCharacterRange` (page 837)
: Invalidates display for *charRange*.

`invalidateDisplayForGlyphRange` (page 838)
: Marks the glyphs in *glyphRange* as needing display, as well as the appropriate regions of the NSTextViews that display those glyphs (using NSView's `setNeedsDisplay` (page 1779)).

`textContainerChangedGeometry` (page 852)
: Invalidates the layout information, and possibly glyphs, for *aTextContainer* and all subsequent NSTextContainers.

`textContainerChangedTextView` (page 853)
: Updates information needed to manage NSTextView objects in *aTextContainer*.

`textStorageChanged` (page 854)
: Invalidates glyph and layout information for a portion of the text in *aTextStorage*.

Turning Background Layout On/off

`setBackgroundLayoutEnabled` (page 845)
: Sets whether the receiver generates glyphs and lays them out when the application's run loop is idle according to *flag*.

`backgroundLayoutEnabled` (page 827)
: Returns `true` if the receiver generates glyphs and lays out text when the application's run loop is idle, `false` if it performs glyph generation and layout only when necessary.

Accessing Glyphs

`insertGlyphAtGlyphIndex` (page 837)
: Inserts *`aGlyph`* into the glyph cache at *`glyphIndex`* and maps it to the character at *`charIndex`*.

`isValidGlyphIndex` (page 839)
: Returns `true` if the specified *`glyphIndex`* refers to a valid glyph, otherwise `false`.

`glyphAtIndex` (page 834)
: Returns the glyph at *`glyphIndex`*.

`replaceGlyphAtIndex` (page 843)
: Replaces the glyph at *`glyphIndex`* with *`newGlyph`*.

`glyphsInRange` (page 836)
: Returns displayable glyphs from *`glyphRange`*.

`deleteGlyphsInRange` (page 830)
: Deletes the glyphs in *`glyphRange`*.

`numberOfGlyphs` (page 841)
: Returns the number of glyphs in the receiver, performing glyph generation if needed to determine this number.

Mapping Characters to Glyphs

`setCharacterIndexForGlyphAtIndex` (page 845)
: Maps the character at *`charIndex`* to the glyph at *`glyphIndex`*.

`characterIndexForGlyphAtIndex` (page 829)
: Returns the index in the NSTextStorage for the first character mapped to the glyph at *`glyphIndex`* within the receiver.

`characterRangeForGlyphRange` (page 829)
: Returns the range for the characters in the receiver's text store that are mapped to the glyphs in *`glyphRange`*.

`glyphRangeForCharacterRange` (page 836)
: Returns the range for the glyphs mapped to the characters of the text store in *`charRange`*.

Setting Glyph Attributes

`setGlyphAttributeForGlyphAtIndex` (page 847)
: Sets a custom attribute value for the glyph at *`glyphIndex`*.

`glyphAttributeForGlyphAtIndex` (page 834)
: Returns the value of the attribute identified by *`attributeTag`* for the glyph at *`glyphIndex`*.

`setDefaultAttachmentScaling` (page 846)
: Sets the default scaling behavior to *`scaling`* if an attachment image is too large to fit in a text container.

`defaultAttachmentScaling` (page 829)
: Returns the default behavior desired if an attachment image is too large to fit in a text container.

`showAttachmentCell` (page 851)
: Actually draws an attachment cell.

Handling Layout for Text Containers

`setTextContainerForGlyphRange` (page 850)
: Sets to *aTextContainer* the NSTextContainer where the glyphs in *glyphRange* are laid out.

`glyphRangeForTextContainer` (page 836)
: Returns the range for glyphs laid out within *aTextContainer*.

`textContainerForGlyphAtIndex` (page 853)
: Returns the NSTextContainer where the glyph at *glyphIndex* is laid out.

`usedRectForTextContainer` (page 855)
: Returns the bounding rectangle for the glyphs laid out in *aTextContainer*, which tells "how full" it is.

Handling Line Fragment Rectangles

`setLineFragmentRectForGlyphAtIndex` (page 848)
: Sets the line fragment rectangle where the glyphs in *glyphRange* are laid out to *fragmentRect*.

`lineFragmentRectForGlyphAtIndex` (page 840)
: Returns the line fragment rectangle containing the glyph at *glyphIndex*.

`lineFragmentUsedRectForGlyphAtIndex` (page 840)
: Returns the portion of the line fragment rectangle containing *glyphIndex* that actually contains glyphs (such as for a partial or wrapped line), plus the line fragment padding defined by the NSTextContainer where the glyphs reside.

`setExtraLineFragmentRect` (page 846)
: Sets a line fragment rectangle for displaying an empty last line in a body of text.

`extraLineFragmentRect` (page 832)
: Returns the rectangle defining the extra line fragment for the insertion point at the end of a text (either in an empty text or after a final paragraph separator).

`extraLineFragmentUsedRect` (page 833)
: Returns the rectangle enclosing the insertion point drawn in the extra line fragment rectangle.

`extraLineFragmentTextContainer` (page 833)
: Returns the NSTextContainer that contains the extra line fragment rectangle, or `null` if there is no extra line fragment rectangle.

`setDrawsOutsideLineFragmentForGlyphAtIndex` (page 846)
: Sets according to *flag* whether the glyph at *glyphIndex* exceeds the bounds of the line fragment where it's laid out.

`drawsOutsideLineFragmentForGlyphAtIndex` (page 831)
: Returns `true` if the glyph at *glyphIndex* exceeds the bounds of the line fragment where it's laid out, `false` otherwise.

Layout of Glyphs

`setLocationForStartOfGlyphRange` (page 848)
: Sets the location where the glyphs in *glyphRange* are laid out to *aPoint*, which is expressed relative to the origin of the line fragment rectangle for *glyphRange*.

`locationForGlyphAtIndex` (page 841)
: Returns the location, in terms of its line fragment rectangle, for the glyph at *glyphIndex*.

`rangeOfNominallySpacedGlyphsContainingIndex` (page 841)
: Returns the range for the glyphs around *glyphIndex* that can be displayed using only their advancements from the font, without pairwise kerning or other adjustments to spacing.

`rectArrayForCharacterRange` (page 842)
: Returns a rectangle for the glyphs in *aTextContainer* that correspond to *charRange*.

`rectArrayForGlyphRange` (page 842)
: Returns a rectangle for the glyphs in *aTextContainer* in *glyphRange*.

`boundingRectForGlyphRange` (page 828)
: Returns a single bounding rectangle (in container coordinates) enclosing all glyphs and other marks drawn in *aTextContainer* for *glyphRange*, including glyphs that draw outside their line fragment rectangles and text attributes such as underlining.

`glyphRangeForBoundingRect` (page 835)
: Returns the smallest contiguous range for glyphs that are laid out wholly or partially within *aRect* in *aTextContainer*.

`glyphRangeForBoundingRectWithoutAdditionalLayout` (page 835)
: Returns the smallest contiguous range for glyphs that are laid out wholly or partially within *bounds* in *aTextContainer*.

`fractionOfDistanceThroughGlyphForPoint` (page 834)
: Returns the ratio of the distance into the glyph relative to the next glyph (in the appropriate sweep direction).

`glyphIndexForPoint` (page 835)
: Returns the index for the glyph nearest *aPoint* within *aTextContainer*.

Handling Layout for Text Blocks

`setLayoutRect` (page 847)
: Sets the layout rectangle enclosing a text block *block* containing the glyph range *glyphRange* to *rect*.

`layoutRectForTextBlock` (page 839)
: Returns the layout rectangle within which the text block *block* containing the glyph range *glyphRange* is to be laid out.

`setBoundsRect` (page 845)
: Sets the bounding rectangle enclosing a text block *block* containing the glyph range *glyphRange* to *rect*.

`boundsRectForTextBlock` (page 828)
: Returns the bounding rectangle enclosing a text block *block* containing the glyph range *glyphRange*.

`layoutRectForTextBlockAtIndex` (page 839)
: Returns the layout rectangle within which the text block *block* containing the glyph at *glyphIndex* is to be laid out.

`boundsRectForTextBlockAtIndex` (page 828)
: Returns the bounding rectangle enclosing a text block *block* containing the glyph at *glyphIndex*.

Display of Special Glyphs

`setNotShownForGlyphAtIndex` (page 848)
: Sets according to *flag* whether the glyph at *glyphIndex* is one that isn't shown.

`notShownAttributeForGlyphAtIndex` (page 841)
: Returns `true` if the glyph at *glyphIndex* isn't shown (in the sense of the PostScript `show` operator), `false` if it is.

`setShowsInvisibleCharacters` (page 849)
: Controls whether the receiver makes whitespace and other typically nonvisible characters visible in layout where possible.

`showsInvisibleCharacters` (page 851)
: Returns `true` if the receiver substitutes visible glyphs for invisible characters if the font and script support it, `false` if it doesn't.

`setShowsControlCharacters` (page 849)
: Controls whether the receiver makes control characters visible in layout where possible.

`showsControlCharacters` (page 851)
: Returns `true` if the receiver substitutes visible glyphs for control characters if the font and script support it, `false` if it doesn't.

Controlling Hyphenation

`setHyphenationFactor` (page 847)
: Sets the threshold as to when hyphenation will be done.

`hyphenationFactor` (page 837)

Finding Unlaid Characters and Glyphs

`firstUnlaidCharacterIndex` (page 833)
: Returns the index for the first unlaid character in the layout manager.

`firstUnlaidGlyphIndex` (page 834)
: Returns the index for the first unlaid glyph in the layout manager.

Using Screen Fonts

`setUsesScreenFonts` (page 851)
: Sets according to *flag* whether the receiver calculates layout and displays text using screen fonts when possible.

`usesScreenFonts` (page 856)
: Returns `true` if the receiver calculates layout and displays text using screen fonts when possible, `false` otherwise.

`substituteFontForFont` (page 852)
: Returns a screen font suitable for use in place of *`originalFont`*, or simply returns *`originalFont`* if a screen font can't be used or isn't available.

Handling Rulers

`rulerAccessoryViewForTextView` (page 844)
: Returns the accessory NSView for *`aRulerView`* in *`aTextView`*.

`rulerMarkersForTextView` (page 844)
: Returns the NSRulerMarkers for *`aRulerView`* in *`aTextView`*, based on *`paraStyle`*.

Managing the Responder Chain

`layoutManagerOwnsFirstResponderInWindow` (page 839)
: Returns `true` if the first responder in *`aWindow`* is an NSTextView associated with the receiver, `false` otherwise.

`firstTextView` (page 833)
: Returns the first NSTextView in the receiver's series of text views.

`textViewForBeginningOfSelection` (page 854)
: Returns the NSTextView containing the first glyph in the selection, or `null` if there's no selection or there isn't enough layout information to determine the text view.

Drawing

`drawBackgroundForGlyphRange` (page 830)
: Draws background marks for *`glyphRange`*, which must lie completely within a single NSTextContainer.

`drawGlyphsForGlyphRange` (page 831)
: Draws the glyphs in *`glyphRange`*, which must lie completely within a single NSTextContainer.

`drawUnderlineForGlyphRange` (page 832)
: Draws underlining for the glyphs in *`glyphRange`*, which must belong to a single line fragment rectangle (as returned by `lineFragmentRectForGlyphAtIndex` (page 840)).

`underlineGlyphRange` (page 855)
: Calculates and draws underlining for the glyphs in *`glyphRange`*, which must belong to a single line fragment rectangle (as returned by `lineFragmentRectForGlyphAtIndex` (page 840)).

`drawStrikethroughForGlyphRange` (page 831)
: Draws strikethrough for the glyphs in *`glyphRange`*, which must belong to a single line fragment rectangle (as returned by `lineFragmentRectForGlyphAtIndex` (page 840)).

`strikethroughGlyphRange` (page 851)
: Calculates and draws strikethrough for the glyphs in *`glyphRange`*, which must belong to a single line fragment rectangle (as returned by `lineFragmentRectForGlyphAtIndex` (page 840)).

Setting the Delegate

`setDelegate` (page 846)
: Sets the receiver's delegate to *anObject*.

`delegate` (page 830)
: Returns the receiver's delegate.

Typesetter Compatibility

`defaultLineHeightForFont` (page 830)
: Returns the default line height for a line of text drawn using *theFont*.

`setTypesetterBehavior` (page 850)
: Sets according to *theBehavior* the default typesetter behavior, which affects glyph spacing and line height.

`typesetterBehavior` (page 854)
: Returns the current typesetter behavior value.

Temporary Attribute Support

`addTemporaryAttributes` (page 827)
: Appends one or more temporary attributes to the attributes dictionary of the specified character range.

`removeTemporaryAttribute` (page 843)
: Removes a temporary attribute *name* from the list of attributes for the specified character range *charRange*.

`setTemporaryAttributes` (page 849)
: Sets one or more temporary attributes passed in *attrs* for the character range specified in *charRange*.

`temporaryAttributesAtCharacterIndex` (page 852)
: Returns the dictionary of temporary attributes for the character range specified in *effectiveCharRange* at character index *charIndex*.

Laying out text

`layoutManagerDidCompleteLayoutForTextContainer` (page 857) *delegate method*
: Informs the delegate that *aLayoutManager* has finished laying out text in *aTextContainer*.

`layoutManagerDidInvalidateLayout` (page 857) *delegate method*
: Informs the delegate that *aLayoutManager* has invalidated layout information (not glyph information).

Constructors

NSLayoutManager

Creates an empty NSLayoutManager.

```
public NSLayoutManager()
```

Instance Methods

addTemporaryAttributes

Appends one or more temporary attributes to the attributes dictionary of the specified character range.

```
public void addTemporaryAttributes(NSDictionary attrs, NSRange charRange)
```

Discussion
Temporary attributes are used only for onscreen drawing and are not persistent in any way. NSTextView uses them to color misspelled words when continuous spell checking is enabled. Currently the only temporary attributes that will be recognized are those related to colors and underlines.

See Also
`setTemporaryAttributes` (page 849)
`removeTemporaryAttribute` (page 843)
`temporaryAttributesAtCharacterIndex` (page 852)

addTextContainer

Appends *aTextContainer* to the series of NSTextContainers where the receiver arranges text.

```
public void addTextContainer(NSTextContainer aTextContainer)
```

Discussion
Invalidates glyphs and layout as needed, but doesn't perform glyph generation or layout.

See Also
`insertTextContainerAtIndex` (page 837)
`removeTextContainerAtIndex` (page 843)
`textContainers` (page 853)
`invalidateGlyphsForCharacterRange` (page 838)
`invalidateLayoutForCharacterRange` (page 838)

backgroundLayoutEnabled

Returns `true` if the receiver generates glyphs and lays out text when the application's run loop is idle, `false` if it performs glyph generation and layout only when necessary.

```
public boolean backgroundLayoutEnabled()
```

See Also
`setBackgroundLayoutEnabled` (page 845)

boundingRectForGlyphRange

Returns a single bounding rectangle (in container coordinates) enclosing all glyphs and other marks drawn in *aTextContainer* for *glyphRange*, including glyphs that draw outside their line fragment rectangles and text attributes such as underlining.

```
public NSRect boundingRectForGlyphRange(NSRange glyphRange, NSTextContainer
    aTextContainer)
```

Discussion
This method is useful for determining the area that needs to be redrawn when a range of glyphs changes.

Performs glyph generation and layout if needed.

See Also
`glyphRangeForTextContainer` (page 836)
`drawsOutsideLineFragmentForGlyphAtIndex` (page 831)

boundsRectForTextBlock

Returns the bounding rectangle enclosing a text block *block* containing the glyph range *glyphRange*.

```
public NSRect boundsRectForTextBlock(NSTextBlock block, NSRange glyphRange)
```

Discussion
This method causes glyph generation but not layout. It returns `ZeroRect` if no rectangle has been set for the specified block since the last invalidation.

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

See Also
`setBoundsRect` (page 845)

boundsRectForTextBlockAtIndex

Returns the bounding rectangle enclosing a text block *block* containing the glyph at *glyphIndex*.

```
public NSRect boundsRectForTextBlockAtIndex(NSTextBlock block, int glyphIndex,
    NSMutableRange effectiveGlyphRange)
```

Discussion
The *effectiveGlyphRange* is set to contain the range for all glyphs in the text block. This method causes glyph generation but not layout. It returns `ZeroRect` if no rectangle has been set for the specified block since the last invalidation.

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

See Also
`setBoundsRect` (page 845)

characterIndexForGlyphAtIndex

Returns the index in the NSTextStorage for the first character mapped to the glyph at *`glyphIndex`* within the receiver.

```
public int characterIndexForGlyphAtIndex(int glyphIndex)
```

Discussion
In many cases it’s better to use the range-mapping methods, `characterRangeForGlyphRange` (page 829) and `glyphRangeForCharacterRange` (page 836), which provide more comprehensive information.

Performs glyph generation if needed.

characterRangeForGlyphRange

Returns the range for the characters in the receiver’s text store that are mapped to the glyphs in *`glyphRange`*.

```
public NSRange characterRangeForGlyphRange(NSRange glyphRange, NSMutableRange
    actualGlyphRange)
```

Discussion
If *`actualGlyphRange`* is non-`null`, expands the requested range as needed so that it identifies all glyphs mapped to those characters and returns the new range by reference in *`actualGlyphRange`*.

Suppose the text store begins with the character “Ö” and the glyph cache contains “O” and “¨”. If you get the character range for the glyph range {0, 1} or {1, 1}, *`actualGlyphRange`* is returned as {0, 2}, indicating that both glyphs are mapped to the character “Ö”.

Performs glyph generation if needed.

See Also
`characterIndexForGlyphAtIndex` (page 829)
`glyphRangeForCharacterRange` (page 836)

defaultAttachmentScaling

Returns the default behavior desired if an attachment image is too large to fit in a text container.

```
public int defaultAttachmentScaling()
```

Discussion
Note that attachment cells control their own size and drawing, so this setting can only be advisory for them, but Application Kit–supplied attachment cells will respect it.

See Also
`setDefaultAttachmentScaling` (page 846)

defaultLineHeightForFont

Returns the default line height for a line of text drawn using *theFont*.

```
public float defaultLineHeightForFont(NSFont theFont)
```

Discussion
This value may vary according to the typesetter behavior.

Availability
Available in Mac OS X v10.2 and later.

See Also
`setTypesetterBehavior` (page 850)

delegate

Returns the receiver's delegate.

```
public Object delegate()
```

See Also
`setDelegate` (page 846)

deleteGlyphsInRange

Deletes the glyphs in *glyphRange*.

```
public void deleteGlyphsInRange(NSRange glyphRange)
```

Discussion
This method is for use by the glyph-generation mechanism and doesn't perform any invalidation or generation of the glyphs or layout. You should never directly invoke this method.

See Also
`insertGlyphAtGlyphIndex` (page 837)

drawBackgroundForGlyphRange

Draws background marks for *glyphRange*, which must lie completely within a single NSTextContainer.

```
public void drawBackgroundForGlyphRange(NSRange glyphRange, NSPoint containerOrigin)
```

Discussion
containerOrigin indicates the position of the NSTextContainer in the coordinate system of the NSView being drawn. This method must be invoked with the graphics focus locked on that NSView.

Background marks are such things as selection highlighting, text background color, and any background for marked text.

Performs glyph generation and layout if needed.

See Also

`drawGlyphsForGlyphRange` (page 831)
`glyphRangeForTextContainer` (page 836)
`textContainerOrigin` (page 1659) (NSTextView)

drawGlyphsForGlyphRange

Draws the glyphs in *glyphRange*, which must lie completely within a single NSTextContainer.

```
public void drawGlyphsForGlyphRange(NSRange glyphRange, NSPoint containerOrigin)
```

Discussion

containerOrigin indicates the position of the NSTextContainer in the coordinate system of the NSView being drawn. This method expects the coordinate system of the view to be flipped. This method must be invoked with the graphics focus locked on that NSView.

Performs glyph generation and layout if needed.

See Also

`drawBackgroundForGlyphRange` (page 830)
`glyphRangeForTextContainer` (page 836)
`textContainerOrigin` (page 1659) (NSTextView)

drawsOutsideLineFragmentForGlyphAtIndex

Returns `true` if the glyph at *glyphIndex* exceeds the bounds of the line fragment where it's laid out, `false` otherwise.

```
public boolean drawsOutsideLineFragmentForGlyphAtIndex(int glyphIndex)
```

Discussion

Exceeding bounds can happen when text is set at a fixed line height. For example, if the user specifies a fixed line height of 12 points and sets the font size to 24 points, the glyphs will exceed their layout rectangles.

Glyphs that draw outside their line fragment rectangles aren't considered when calculating enclosing rectangles with the `rectArrayForCharacterRange` (page 842) and `rectArrayForGlyphRange` (page 842) methods. They are, however, considered by `boundingRectForGlyphRange` (page 828).

Performs glyph generation and layout if needed.

drawStrikethroughForGlyphRange

Draws strikethrough for the glyphs in *glyphRange*, which must belong to a single line fragment rectangle (as returned by `lineFragmentRectForGlyphAtIndex` (page 840)).

```
public void drawStrikethroughForGlyphRange(NSRange glyphRange, int strikethroughVal,
     float baselineOffset, NSRect lineRect, NSRange lineGlyphRange, NSPoint
    containerOrigin)
```

Discussion

strikethroughVal indicates the style of strikethrough to draw. *baselineOffset* indicates how far above the text baseline the underline should be drawn (usually a negative value). *lineRect* is the line fragment rectangle containing the glyphs to draw strikethrough for, and *lineGlyphRange* is the range of all glyphs within that line fragment rectangle. *containerOrigin* is the origin of the line fragment rectangle's NSTextContainer in its NSTextView.

This method is invoked automatically by `strikethroughGlyphRange` (page 851); you should rarely need to invoke it directly.

Availability

Available in Mac OS X v10.3 and later.

drawUnderlineForGlyphRange

Draws underlining for the glyphs in *glyphRange*, which must belong to a single line fragment rectangle (as returned by `lineFragmentRectForGlyphAtIndex` (page 840)).

```
public void drawUnderlineForGlyphRange(NSRange glyphRange, int underlineType, float
     baselineOffset, NSRect lineRect, NSRange lineGlyphRange, NSPoint
    containerOrigin)
```

Discussion

underlineType indicates the style of underlining to draw; NSLayoutManager accepts only `NSAttributedString.SingleUnderlineStyle`, but subclasses can define their own underline styles. *baselineOffset* indicates how far below the text baseline the underline should be drawn; it's usually a positive value. *lineRect* is the line fragment rectangle containing the glyphs to draw underlining for, and *lineGlyphRange* is the range of all glyphs within that line fragment rectangle. *containerOrigin* is the origin of the line fragment rectangle's NSTextContainer in its NSTextView.

This method is invoked automatically by `underlineGlyphRange` (page 855); you should rarely need to invoke it directly.

See Also

`textContainerForGlyphAtIndex` (page 853)
`textContainerOrigin` (page 1659) (NSTextView)

extraLineFragmentRect

Returns the rectangle defining the extra line fragment for the insertion point at the end of a text (either in an empty text or after a final paragraph separator).

```
public NSRect extraLineFragmentRect()
```

Discussion

The rectangle is defined in the coordinate system of its NSTextContainer. Returns `NSRect.ZeroRect` if there is no such rectangle.

See Also

`extraLineFragmentUsedRect` (page 833)
`extraLineFragmentTextContainer` (page 833)
`setExtraLineFragmentRect` (page 846)

extraLineFragmentTextContainer

Returns the NSTextContainer that contains the extra line fragment rectangle, or `null` if there is no extra line fragment rectangle.

```
public NSTextContainer extraLineFragmentTextContainer()
```

Discussion

This rectangle is used to display the insertion point at the end of a text (either in an empty text or after a final paragraph separator).

See Also

`extraLineFragmentRect` (page 832)
`extraLineFragmentUsedRect` (page 833)
`setExtraLineFragmentRect` (page 846)

extraLineFragmentUsedRect

Returns the rectangle enclosing the insertion point drawn in the extra line fragment rectangle.

```
public NSRect extraLineFragmentUsedRect()
```

Discussion

The rectangle is defined in the coordinate system of its NSTextContainer. Returns `NSRect.ZeroRect` if there is no extra line fragment rectangle.

The extra line fragment used rectangle is twice as wide (or tall) as the NSTextContainer's line fragment padding, with the insertion point itself in the middle.

See Also

`extraLineFragmentRect` (page 832)
`extraLineFragmentTextContainer` (page 833)
`setExtraLineFragmentRect` (page 846)

firstTextView

Returns the first NSTextView in the receiver's series of text views.

```
public NSTextView firstTextView()
```

Discussion

This NSTextView is the object of various NSText and NSTextView notifications posted.

firstUnlaidCharacterIndex

Returns the index for the first unlaid character in the layout manager.

```
public int firstUnlaidCharacterIndex()
```

firstUnlaidGlyphIndex

Returns the index for the first unlaid glyph in the layout manager.

```
public int firstUnlaidGlyphIndex()
```

fractionOfDistanceThroughGlyphForPoint

Returns the ratio of the distance into the glyph relative to the next glyph (in the appropriate sweep direction).

```
public float fractionOfDistanceThroughGlyphForPoint(NSPoint aPoint, NSTextContainer
     aTextContainer)
```

Discussion

NSLayoutManager currently supports only left-to-right sweep. *aPoint* is expressed in the coordinate system of *aTextContainer*.

For purposes such as dragging out a selection or placing the insertion point, a return value less than or equal to 0.5 indicates that *aPoint* should be considered as falling before the glyph index returned by glyphIndexForPoint (page 835) for the specified point. A return value greater than 0.5 indicates that it should be considered as falling after the glyph index returned by glyphIndexForPoint (page 835). If the nearest glyph doesn't lie under *aPoint* at all (for example, if *aPoint* is beyond the beginning or end of a line), this ratio will be 0 or 1.

Suppose the glyph stream contains the glyphs "A" and "b", with the width of "A" being 13 points. If the user clicks at a location 8 points into "A", the return value is 8/13, or 0.615. In this case, the point given should be considered as falling between "A" and "b" for purposes such as dragging out a selection or placing the insertion point.

Performs glyph generation and layout if needed.

See Also

glyphIndexForPoint (page 835)

glyphAtIndex

Returns the glyph at *glyphIndex*.

```
public int glyphAtIndex(int glyphIndex)
```

Discussion

Throws a RangeException if *glyphIndex* is out of bounds.

Performs glyph generation if needed. To avoid an exception with glyphAtIndex you must first check the glyph index against the number of glyphs, which requires generating all glyphs.

glyphAttributeForGlyphAtIndex

Returns the value of the attribute identified by *attributeTag* for the glyph at *glyphIndex*.

```
public int glyphAttributeForGlyphAtIndex(int attributeTag, int glyphIndex)
```

Discussion

Subclasses that define their own custom attributes must override this method to access their own storage for the attribute values. Nonnegative tags are reserved by Apple; you can define your own attributes with negative tags and set values using `setGlyphAttributeForGlyphAtIndex` (page 847).

glyphIndexForPoint

Returns the index for the glyph nearest *aPoint* within *aTextContainer*.

```
public int glyphIndexForPoint(NSPoint aPoint, NSTextContainer aTextContainer)
```

Discussion

aPoint is expressed in the coordinate system of *aTextContainer*.

Performs glyph generation and layout if needed.

See Also

`fractionOfDistanceThroughGlyphForPoint` (page 834)

glyphRangeForBoundingRect

Returns the smallest contiguous range for glyphs that are laid out wholly or partially within *aRect* in *aTextContainer*.

```
public NSRange glyphRangeForBoundingRect(NSRect aRect, NSTextContainer
    aTextContainer)
```

Discussion

The range returned can include glyphs that don't fall inside or intersect *aRect*, though the first and last glyphs in the range always do. This method is used to determine which glyphs need to be displayed within a given rectangle.

Performs glyph generation and layout if needed.

See Also

`glyphRangeForBoundingRectWithoutAdditionalLayout` (page 835)

glyphRangeForBoundingRectWithoutAdditionalLayout

Returns the smallest contiguous range for glyphs that are laid out wholly or partially within *bounds* in *aTextContainer*.

```
public NSRange glyphRangeForBoundingRectWithoutAdditionalLayout(NSRect bounds,
    NSTextContainer container)
```

Discussion

The range returned can include glyphs that don't fall inside or intersect *aRect*, though the first and last glyphs in the range always do.

Unlike `glyphRangeForBoundingRect` (page 835), this method doesn't perform glyph generation or layout. Its results, though faster, can be incorrect. This method is primarily for use by NSTextView; you should rarely need to use it yourself.

See Also

`glyphRangeForBoundingRect` (page 835)

glyphRangeForCharacterRange

Returns the range for the glyphs mapped to the characters of the text store in *charRange*.

```
public NSRange glyphRangeForCharacterRange(NSRange charRange, NSMutableRange
    actualCharRange)
```

Discussion

If *actualCharRange* is non-`null`, expands the requested range as needed so that it identifies all characters mapped to those glyphs and returns the new range by reference in *actualCharRange*.

Suppose the text store contains the characters "`n~`", and the glyph cache contains "`ñ`". If you get the glyph range for the character range {0, 1} or {1, 1}, *actualCharRange* is returned as {0, 2}, indicating both of the characters mapped to the glyph "`ñ`".

Performs glyph generation if needed.

See Also

`characterIndexForGlyphAtIndex` (page 829)

glyphRangeForTextContainer

Returns the range for glyphs laid out within *aTextContainer*.

```
public NSRange glyphRangeForTextContainer(NSTextContainer aTextContainer)
```

Discussion

Performs glyph generation and layout if needed.

glyphsInRange

Returns displayable glyphs from *glyphRange*.

```
public int [] glyphsInRange(NSRange glyphRange)
```

Discussion

`Throws` a `RangeException` if the range specified exceeds the bounds of the actual glyph range for the receiver.

Performs glyph generation if needed.

See Also

`glyphAtIndex` (page 834)

`notShownAttributeForGlyphAtIndex` (page 841)

hyphenationFactor

```
public float hyphenationFactor()
```

Discussion

Whenever (width of the real contents of the line) / (the line fragment width) is less than `hyphenationFactor`, hyphenation will be attempted when laying out the line. The range of this factor is from 0.0 to 1.0. By default, the value is 0.0, meaning hyphenation is off. A value of 1.0 causes hyphenation to be attempted always. Note that hyphenation will slow down text layout and increase memory usage, so it should be used sparingly.

See Also

`setHyphenationFactor` (page 847)

insertGlyphAtGlyphIndex

Inserts *aGlyph* into the glyph cache at *glyphIndex* and maps it to the character at *charIndex*.

```
public void insertGlyphAtGlyphIndex(int aGlyph, int glyphIndex, int charIndex)
```

Discussion

If the glyph is mapped to several characters, *charIndex* should indicate the first character it's mapped to.

This method is for use by the glyph-generation mechanism and doesn't perform any invalidation or generation of the glyphs or layout. You should never directly invoke this method.

See Also

`deleteGlyphsInRange` (page 830)

`replaceGlyphAtIndex` (page 843)

insertTextContainerAtIndex

Inserts *aTextContainer* into the series of text containers at *index* and invalidates layout for all subsequent NSTextContainer's.

```
public void insertTextContainerAtIndex(NSTextContainer aTextContainer, int index)
```

Discussion

Also invalidates glyph information as needed.

See Also

`addTextContainer` (page 827)

`removeTextContainerAtIndex` (page 843)

`textContainers` (page 853)

invalidateDisplayForCharacterRange

Invalidates display for *charRange*.

```
public void invalidateDisplayForCharacterRange(NSRange charRange)
```

Discussion

Unlaid parts of the range are remembered and will definitely be redisplayed at some point later when the layout is available. Does not actually cause layout.

invalidateDisplayForGlyphRange

Marks the glyphs in *glyphRange* as needing display, as well as the appropriate regions of the NSTextViews that display those glyphs (using NSView's `setNeedsDisplay` (page 1779)).

```
public void invalidateDisplayForGlyphRange(NSRange glyphRange)
```

Discussion

You should rarely need to invoke this method.

invalidateGlyphsForCharacterRange

Invalidates the cached glyphs for the characters in *charRange* and adjusts the remaining glyph-to-character mapping according to *lengthChange*, which indicates the number of characters added to or removed from the text store.

```
public void invalidateGlyphsForCharacterRange(NSRange charRange, int lengthChange,
     NSMutableRange actualCharRange)
```

Discussion

If non-`null`, *actualCharRange* is set to the range of characters mapped to the glyphs just invalidated. This range can be larger than the range of characters given due to the effect of context on glyphs and layout.

You should rarely need to invoke this method. It only invalidates glyph information, and performs no glyph generation or layout. Because invalidating glyphs also invalidates layout, after invoking this method you should also invoke `invalidateLayoutForCharacterRange` (page 838), passing *charRange* as the first argument and `false` as the *flag*.

invalidateLayoutForCharacterRange

Invalidates the layout information for the glyphs mapped to the characters in *charRange*.

```
public void invalidateLayoutForCharacterRange(NSRange charRange, boolean flag,
    NSMutableRange actualCharRange)
```

Discussion

If *flag* is `true`, attempts to save some layout information to avoid recalculation; if *flag* is `false`, saves no layout information. You should typically pass `false` for *flag.* If non-`null`, *actualCharRange* is set to the range of characters mapped to the glyphs whose layout information has been invalidated. This range can be larger than the range of characters given due to the effect of context on glyphs and layout.

This method only invalidates information; it performs no glyph generation or layout. You should rarely need to invoke this method.

See Also

`invalidateGlyphsForCharacterRange` (page 838)

isValidGlyphIndex

Returns `true` if the specified *`glyphIndex`* refers to a valid glyph, otherwise `false`.

```
public boolean isValidGlyphIndex(int glyphIndex)
```

layoutManagerOwnsFirstResponderInWindow

Returns `true` if the first responder in *`aWindow`* is an NSTextView associated with the receiver, `false` otherwise.

```
public boolean layoutManagerOwnsFirstResponderInWindow(NSWindow aWindow)
```

layoutRectForTextBlock

Returns the layout rectangle within which the text block *`block`* containing the glyph range *`glyphRange`* is to be laid out.

```
public NSRect layoutRectForTextBlock(NSTextBlock block, NSRange glyphRange)
```

Discussion
This method causes glyph generation but not layout. It returns `ZeroRect` if no rectangle has been set for the specified block since the last invalidation.

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

See Also
`setLayoutRect` (page 847)

layoutRectForTextBlockAtIndex

Returns the layout rectangle within which the text block *`block`* containing the glyph at *`glyphIndex`* is to be laid out.

```
public NSRect layoutRectForTextBlockAtIndex(NSTextBlock block, int glyphIndex,
    NSMutableRange effectiveGlyphRange)
```

Discussion
The *`effectiveGlyphRange`* is set to contain the range for all glyphs in the text block. This method causes glyph generation but not layout. It returns `ZeroRect` if no rectangle has been set for the specified block since the last invalidation.

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

See Also
`setLayoutRect` (page 847)

lineFragmentRectForGlyphAtIndex

```
public NSRect lineFragmentRectForGlyphAtIndex(int glyphIndex, NSMutableRange
    effectiveGlyphRange)
```

Returns the line fragment rectangle containing the glyph at *glyphIndex*.

```
public NSRect lineFragmentRectForGlyphAtIndex(int glyphIndex, NSMutableRange
    effectiveGlyphRange, boolean flag)
```

Discussion

The rectangle is defined in the coordinate system of its NSTextContainer. If non-`null`, *effectiveGlyphRange* is set to contain the range for all glyphs in that line fragment.

Performs glyph generation if needed. For the variant method with the Boolean argument *flag*, if *flag* is `true`, the method performs no additional layout so as to avoid an infinite recursion.

Availability

Available in Mac OS X v10.4 and later.

See Also

`setLineFragmentRectForGlyphAtIndex` (page 848)
`lineFragmentUsedRectForGlyphAtIndex` (page 840)

lineFragmentUsedRectForGlyphAtIndex

Returns the portion of the line fragment rectangle containing *glyphIndex* that actually contains glyphs (such as for a partial or wrapped line), plus the line fragment padding defined by the NSTextContainer where the glyphs reside.

```
public NSRect lineFragmentUsedRectForGlyphAtIndex(int glyphIndex, NSMutableRange
    effectiveGlyphRange)
```

Returns the portion of the line fragment rectangle containing *glyphIndex* that actually contains glyphs (such as for a partial or wrapped line), plus the line fragment padding defined by the NSTextContainer where the glyphs reside. If *flag* is true, performs no additional layout so as to avoid an infinite recursion.

```
public NSRect lineFragmentUsedRectForGlyphAtIndex(int glyphIndex, NSMutableRange
    effectiveGlyphRange, boolean flag)
```

Discussion

This rectangle is defined in the coordinate system of its NSTextContainer, and is based on line calculation only—that is, it isn't a bounding box for the glyphs in the line fragment.

If non-`null`, *effectiveGlyphRange* is set to contain the range for all glyphs in the line fragment.

Performs glyph generation if needed.

Availability

Available in Mac OS X v10.4 and later.

See Also

`setLineFragmentRectForGlyphAtIndex` (page 848)
`lineFragmentRectForGlyphAtIndex` (page 840)

locationForGlyphAtIndex

Returns the location, in terms of its line fragment rectangle, for the glyph at *glyphIndex*.

```
public NSPoint locationForGlyphAtIndex(int glyphIndex)
```

Discussion

The line fragment rectangle in turn is defined in the coordinate system of the text container where it resides.

Performs glyph generation and layout if needed.

See Also

lineFragmentRectForGlyphAtIndex (page 840)

lineFragmentUsedRectForGlyphAtIndex (page 840)

notShownAttributeForGlyphAtIndex

Returns `true` if the glyph at *glyphIndex* isn't shown (in the sense of the PostScript `show` operator), `false` if it is.

```
public boolean notShownAttributeForGlyphAtIndex(int glyphIndex)
```

Discussion

For example, a tab, newline, or attachment glyph doesn't get shown; it just affects the layout of following glyphs or locates the attachment graphic. Space characters, however, typically are shown as glyphs with a displacement, though they leave no visible marks. Throws a `RangeException` if *glyphIndex* is out of bounds.

Performs glyph generation and layout if needed.

See Also

setNotShownForGlyphAtIndex (page 848)

numberOfGlyphs

Returns the number of glyphs in the receiver, performing glyph generation if needed to determine this number.

```
public int numberOfGlyphs()
```

rangeOfNominallySpacedGlyphsContainingIndex

Returns the range for the glyphs around *glyphIndex* that can be displayed using only their advancements from the font, without pairwise kerning or other adjustments to spacing.

```
public NSRange rangeOfNominallySpacedGlyphsContainingIndex(int glyphIndex)
```

Discussion

Performs glyph generation and layout if needed.

rectArrayForCharacterRange

Returns a rectangle for the glyphs in *aTextContainer* that correspond to *charRange*.

```
public NSRect rectArrayForCharacterRange(NSRange charRange, NSRange selCharRange,
     NSTextContainer aTextContainer)
```

Discussion

This rectangles can be used to draw the background or highlight for the given range of characters. *selCharRange* indicates selected characters, which can affect the size of the rectangles; it must be equal to or contain *charRange*. To calculate the rectangles for drawing the background, use a selected character range whose location is `NSArray.NotFound`. To calculate the rectangles for drawing highlighting for *charRange*, use a selected character range that contains *charRange*.

The number of rectangles returned isn't necessarily the number of lines enclosing the specified range. Contiguous lines can share an enclosing rectangle, and lines broken into several fragments have a separate enclosing rectangle for each fragment.

The rectangle returned is owned by the receiver, and is overwritten by various NSLayoutManager methods. You should never free it and should copy it if you need to keep the values or use them after sending other messages to the layout manager.

The purpose of this method is to calculate line rectangles for drawing the text background and highlighting. These rectangles don't necessarily enclose glyphs that draw outside their line fragment rectangles; use `boundingRectForGlyphRange` (page 828) to determine the area that contains all drawing performed for a range of glyphs.

Performs glyph generation and layout if needed.

See Also

`glyphRangeForTextContainer` (page 836)
`characterRangeForGlyphRange` (page 829)
`drawsOutsideLineFragmentForGlyphAtIndex` (page 831)

rectArrayForGlyphRange

Returns a rectangle for the glyphs in *aTextContainer* in *glyphRange*.

```
public NSRect rectArrayForGlyphRange(NSRange glyphRange, NSRange selGlyphRange,
    NSTextContainer aTextContainer)
```

Discussion

This rectangles can be used to draw the background or highlight for the given range of glyphs. *selGlyphRange* indicates selected glyphs. To calculate the rectangles for drawing the background, use a selected glyph range whose location is `NSArray.NotFound`. To calculate the rectangles for highlighting, use a selected glyph range that contains *glyphRange*.

The number of rectangles returned isn't necessarily the number of lines enclosing the specified range. Contiguous lines can share an enclosing rectangle, and lines broken into several fragments have a separate enclosing rectangle for each fragment.

The rectangle returned is owned by the receiver and is overwritten by various NSLayoutManager methods. You should never free it and should copy it if you need to keep the values or use them after sending other messages to the layout manager.

The purpose of this method is to calculate line rectangles for drawing the text background and highlighting. These rectangles don't necessarily enclose glyphs that draw outside their line fragment rectangles; use `boundingRectForGlyphRange` (page 828) to determine the area that contains all drawing performed for a range of glyphs.

Performs glyph generation and layout if needed.

See Also

`glyphRangeForTextContainer` (page 836)

`drawsOutsideLineFragmentForGlyphAtIndex` (page 831)

removeTemporaryAttribute

Removes a temporary attribute *name* from the list of attributes for the specified character range *charRange*.

```
public void removeTemporaryAttribute(String name, NSRange charRange)
```

Discussion

Temporary attributes are used only for onscreen drawing and are not persistent in any way. NSTextView uses them to color misspelled words when continuous spell checking is enabled. Currently the only temporary attributes that will be recognized are those related to colors and underlines.

See Also

`setTemporaryAttributes` (page 849)

`addTemporaryAttributes` (page 827)

`temporaryAttributesAtCharacterIndex` (page 852)

removeTextContainerAtIndex

Removes the NSTextContainer at *index* and invalidates the layout as needed.

```
public void removeTextContainerAtIndex(int index)
```

Discussion

Also invalidates glyph information as needed.

See Also

`addTextContainer` (page 827)

`insertTextContainerAtIndex` (page 837)

`textContainers` (page 853)

`invalidateGlyphsForCharacterRange` (page 838)

`invalidateLayoutForCharacterRange` (page 838)

replaceGlyphAtIndex

Replaces the glyph at *glyphIndex* with *newGlyph*.

```
public void replaceGlyphAtIndex(int glyphIndex, int newGlyph)
```

Discussion
Doesn't alter the glyph-to-character mapping or invalidate layout information.

This method is for use by the glyph-generation mechanism and doesn't perform any invalidation or generation of the glyphs or layout. You should never directly invoke this method.

See Also
`setCharacterIndexForGlyphAtIndex` (page 845)
`invalidateGlyphsForCharacterRange` (page 838)
`invalidateLayoutForCharacterRange` (page 838)

replaceTextStorage

Replaces the NSTextStorage for the group of text-system objects containing the receiver with *newTextStorage*.

```
public void replaceTextStorage(NSTextStorage newTextStorage)
```

Discussion
All NSLayoutManagers sharing the original NSTextStorage then share the new one. This method makes all the adjustments necessary to keep these relationships intact, unlike `setTextStorage` (page 850).

rulerAccessoryViewForTextView

Returns the accessory NSView for *aRulerView* in *aTextView*.

```
public NSView rulerAccessoryViewForTextView(NSTextView aTextView, NSParagraphStyle
     paraStyle, NSRulerView aRulerView, boolean flag)
```

Discussion
This accessory contains tab wells, text alignment buttons, and so on. *paraStyle* is used to set the state of the controls in the accessory NSView; it must not be `null`. If *flag* is `true` the accessory view is enabled and accepts mouse and keyboard events; if `false` it's disabled.

This method is invoked automatically by the NSTextView object using the layout manager. You should rarely need to invoke it, but you can override it to customize ruler support. If you do this method directly, note that it neither installs the ruler accessory view nor sets the markers for the NSRulerView. You must install the accessory view into the ruler using NSRulerView's `setAccessoryView` (page 1219) method. To set the markers, use `rulerMarkersForTextView` (page 844) to get the markers needed and then send `setMarkers` (page 1220) to the ruler.

See Also
`horizontalRulerView` (page 1275) (NSScrollView)

rulerMarkersForTextView

Returns the NSRulerMarkers for *aRulerView* in *aTextView*, based on *paraStyle*.

```
public NSArray rulerMarkersForTextView(NSTextView aTextView, NSParagraphStyle
    paraStyle, NSRulerView aRulerView)
```

Discussion
These markers represent such things as left and right margins, first-line indent, and tab stops. You can set these markers immediately with NSRulerView's `setMarkers` (page 1220) method.

This method is invoked automatically by the NSTextView object using the layout manager. You should rarely need to invoke it, but you can override it to add new kinds of markers or otherwise customize ruler support.

See Also
`rulerAccessoryViewForTextView` (page 844)

setBackgroundLayoutEnabled

Sets whether the receiver generates glyphs and lays them out when the application's run loop is idle according to *flag*.

```
public void setBackgroundLayoutEnabled(boolean flag)
```

See Also
`backgroundLayoutEnabled` (page 827)

setBoundsRect

Sets the bounding rectangle enclosing a text block *block* containing the glyph range *glyphRange* to *rect*.

```
public void setBoundsRect(NSRect rect, NSTextBlock block, NSRange glyphRange)
```

Discussion
This method causes glyph generation but not layout.

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

See Also
`boundingRectForGlyphRange` (page 828)
`boundsRectForTextBlockAtIndex` (page 828)
`boundsRectForTextBlock` (page 828)

setCharacterIndexForGlyphAtIndex

Maps the character at *charIndex* to the glyph at *glyphIndex*.

```
public void setCharacterIndexForGlyphAtIndex(int charIndex, int glyphIndex)
```

Discussion
This method is for use by the glyph-generation mechanism and doesn't perform any invalidation or generation of the glyphs or layout. You should never directly invoke this method.

See Also
`characterIndexForGlyphAtIndex` (page 829)
`characterRangeForGlyphRange` (page 829)
`glyphRangeForCharacterRange` (page 836)

setDefaultAttachmentScaling

Sets the default scaling behavior to *scaling* if an attachment image is too large to fit in a text container.

```
public void setDefaultAttachmentScaling(int scaling)
```

Discussion

Note that attachment cells control their own size and drawing, so this setting can only be advisory for them, but Application Kit–supplied attachment cells will respect it.

See Also

`defaultAttachmentScaling` (page 829)

setDelegate

Sets the receiver's delegate to *anObject*.

```
public void setDelegate(Object anObject)
```

See Also

`delegate` (page 830)

setDrawsOutsideLineFragmentForGlyphAtIndex

Sets according to *flag* whether the glyph at *glyphIndex* exceeds the bounds of the line fragment where it's laid out.

```
public void setDrawsOutsideLineFragmentForGlyphAtIndex(boolean flag, int glyphIndex)
```

Discussion

This can happen when text is set at a fixed line height. For example, if the user specifies a fixed line height of 12 points and sets the font size to 24 points, the glyphs will exceed their layout rectangles.

This method is used by the layout mechanism; you should never invoke it directly.

See Also

`drawsOutsideLineFragmentForGlyphAtIndex` (page 831)

setExtraLineFragmentRect

Sets a line fragment rectangle for displaying an empty last line in a body of text.

```
public void setExtraLineFragmentRect(NSRect aRect, NSRect usedRect, NSTextContainer
      aTextContainer)
```

Discussion

aRect is the rectangle to set, and *aTextContainer* is the NSTextContainer where the rectangle should be laid out. *usedRect* indicates where the insertion point is drawn.

This method is used by the layout mechanism; you should never invoke it directly.

See Also

`extraLineFragmentRect` (page 832)

`extraLineFragmentUsedRect` (page 833)
`textContainers` (page 853)

setGlyphAttributeForGlyphAtIndex

Sets a custom attribute value for the glyph at *glyphIndex*.

```
public void setGlyphAttributeForGlyphAtIndex(int attributeTag, int anInt, int
    glyphIndex)
```

Discussion

attributeTag identifies the custom attribute, and *anInt* is its new value.

Subclasses that define their own custom attributes must override this method and provide their own storage for the attribute values. Nonnegative tags are reserved; you can define your own attributes with negative tags and set values using this method.

This method doesn't perform glyph generation or layout. The glyph at *glyphIndex* must already have been generated.

See Also

`glyphAttributeForGlyphAtIndex` (page 834)

setHyphenationFactor

Sets the threshold as to when hyphenation will be done.

```
public void setHyphenationFactor(float factor)
```

Discussion

factor is in the range of 0.0 to 1.0. Whenever (width of the real contents of the line) / (the line fragment width) is below *factor*, hyphenation will be attempted when laying out the line. By default, the value is 0.0, meaning hyphenation is off. A *factor* of 1.0 causes hyphenation to be attempted always. Note that hyphenation will slow down text layout and increase memory usage, so it should be used sparingly.

See Also

`hyphenationFactor` (page 837)

setLayoutRect

Sets the layout rectangle enclosing a text block *block* containing the glyph range *glyphRange* to *rect*.

```
public void setLayoutRect(NSRect rect, NSTextBlock block, NSRange glyphRange)
```

Discussion

This method causes glyph generation but not layout.

Availability

Available in Mac OS X v10.4 and later.

See Also

`layoutRectForTextBlockAtIndex` (page 839) `layoutRectForTextBlock` (page 839)

setLineFragmentRectForGlyphAtIndex

Sets the line fragment rectangle where the glyphs in *glyphRange* are laid out to *fragmentRect*.

```
public void setLineFragmentRectForGlyphAtIndex(NSRect fragmentRect, NSRange
    glyphRange, NSRect usedRect)
```

Discussion

The text container must be specified first with `setTextContainerForGlyphRange` (page 850), and the exact positions of the glyphs must be set after the line fragment rectangle with `setLocationForStartOfGlyphRange` (page 848). *usedRect* indicates the portion of *fragmentRect*, in the NSTextContainer's coordinate system, that actually contains glyphs or other marks that are drawn (including the text container's line fragment padding). *usedRect* must be equal to or contained within *fragmentRect*.

This method is used by the layout mechanism; you should never invoke it directly.

See Also

`lineFragmentRectForGlyphAtIndex` (page 840)

`lineFragmentUsedRectForGlyphAtIndex` (page 840)

setLocationForStartOfGlyphRange

Sets the location where the glyphs in *glyphRange* are laid out to *aPoint*, which is expressed relative to the origin of the line fragment rectangle for *glyphRange*.

```
public void setLocationForStartOfGlyphRange(NSPoint aPoint, NSRange glyphRange)
```

Discussion

glyphRange defines a series of glyphs that can be displayed with a single PostScript `show` operation (a nominal range). Setting the location for a series of glyphs implies that the glyphs preceding it can't be included in a single `show` operation.

Before setting the location for a glyph range, you must specify the text container with `setTextContainerForGlyphRange` (page 850) and the line fragment rectangle with `setLineFragmentRectForGlyphAtIndex` (page 848).

This method is used by the layout mechanism; you should never invoke it directly.

See Also

`rangeOfNominallySpacedGlyphsContainingIndex` (page 841)

setNotShownForGlyphAtIndex

Sets according to *flag* whether the glyph at *glyphIndex* is one that isn't shown.

```
public void setNotShownForGlyphAtIndex(boolean flag, int glyphIndex)
```

Discussion
For example, a tab or newline character doesn't leave any marks; it just indicates where following glyphs are laid out. Throws a `RangeException` if *`glyphIndex`* is out of bounds.

This method is used by the layout mechanism; you should never invoke it directly.

See Also
`notShownAttributeForGlyphAtIndex` (page 841)

setShowsControlCharacters

Controls whether the receiver makes control characters visible in layout where possible.

```
public void setShowsControlCharacters(boolean flag)
```

Discussion
If *`flag`* is `true`, it substitutes visible glyphs for control characters if the font and script support it; if *`flag`* is `false` it doesn't.

See Also
`setShowsInvisibleCharacters` (page 849)
`showsControlCharacters` (page 851)

setShowsInvisibleCharacters

Controls whether the receiver makes whitespace and other typically nonvisible characters visible in layout where possible.

```
public void setShowsInvisibleCharacters(boolean flag)
```

Discussion
If *`flag`* is `true`, it substitutes visible glyphs for invisible characters if the font and script support it; if *`flag`* is `false` it doesn't.

See Also
`setShowsControlCharacters` (page 849)
`showsInvisibleCharacters` (page 851)

setTemporaryAttributes

Sets one or more temporary attributes passed in *`attrs`* for the character range specified in *`charRange`*.

```
public void setTemporaryAttributes(NSDictionary attrs, NSRange charRange)
```

Discussion
Temporary attributes are used only for onscreen drawing and are not persistent in any way. NSTextView uses them to color misspelled words when continuous spell checking is enabled. Currently the only temporary attributes that will be recognized are those related to colors and underlines.

See Also
`addTemporaryAttributes` (page 827)

`removeTemporaryAttribute` (page 843)
`temporaryAttributesAtCharacterIndex` (page 852)

setTextContainerForGlyphRange

Sets to *aTextContainer* the NSTextContainer where the glyphs in *glyphRange* are laid out.

```
public void setTextContainerForGlyphRange(NSTextContainer aTextContainer, NSRange
     glyphRange)
```

Discussion
You specify the layout within the container with the `setLineFragmentRectForGlyphAtIndex` (page 848) and `setLocationForStartOfGlyphRange` (page 848) methods.

This method is used by the layout mechanism; you should never invoke it directly.

See Also
`textContainerForGlyphAtIndex` (page 853)

setTextStorage

Sets the receiver's NSTextStorage to *textStorage*.

```
public void setTextStorage(NSTextStorage textStorage)
```

Discussion
This method is invoked automatically when you add an NSLayoutManager to an NSTextStorage object; you should never need to invoke it directly, but might want to override it. If you want to replace the NSTextStorage for an established group of text-system objects containing the receiver, use `replaceTextStorage` (page 844).

See Also
`addLayoutManager` (page 1588) (NSTextStorage)

setTypesetterBehavior

Sets according to *theBehavior* the default typesetter behavior, which affects glyph spacing and line height.

```
public void setTypesetterBehavior(int theBehavior)
```

Discussion
Possible values for *theBehavior* are described in "Constants" (page 856).

If the application was linked on a system prior to Mac OS X version 10.2, NSLayoutManager uses `TypesetterOriginalBehavior` by default.

Availability
Available in Mac OS X v10.2 and later.

See Also
`typesetterBehavior` (page 854)

setUsesScreenFonts

Sets according to `flag` whether the receiver calculates layout and displays text using screen fonts when possible.

```
public void setUsesScreenFonts(boolean flag)
```

See Also

`usesScreenFonts` (page 856)

`substituteFontForFont` (page 852)

showAttachmentCell

Actually draws an attachment cell.

```
public void showAttachmentCell(NSCell cell, NSRect rect, int attachmentIndex)
```

Discussion

The attachment passed in `cell` should be drawn within the given `rect`. `attachmentIndex` is provided for those cells that alter their appearance based on their location.

showsControlCharacters

Returns `true` if the receiver substitutes visible glyphs for control characters if the font and script support it, `false` if it doesn't.

```
public boolean showsControlCharacters()
```

See Also

`showsInvisibleCharacters` (page 851)

`setShowsControlCharacters` (page 849)

showsInvisibleCharacters

Returns `true` if the receiver substitutes visible glyphs for invisible characters if the font and script support it, `false` if it doesn't.

```
public boolean showsInvisibleCharacters()
```

See Also

`showsControlCharacters` (page 851)

`setShowsInvisibleCharacters` (page 849)

strikethroughGlyphRange

Calculates and draws strikethrough for the glyphs in `glyphRange`, which must belong to a single line fragment rectangle (as returned by `lineFragmentRectForGlyphAtIndex` (page 840)).

```
public void strikethroughGlyphRange(NSRange glyphRange, int strikethroughVal, NSRect
      lineRect, NSRange lineGlyphRange, NSPoint containerOrigin)
```

Discussion

strikethroughVal indicates the style of strikethrough to draw. *lineRect* is the line fragment rectangle containing the glyphs to draw strikethrough for, and *lineGlyphRange* is the range of all glyphs within that line fragment rectangle. *containerOrigin* is the origin of the line fragment rectangle's NSTextContainer in its NSTextView.

This method determines which glyphs actually need to be strikethrough based on *strikethroughVal*. After determining which glyphs to draw strikethrough on, this method invokes `drawStrikethroughForGlyphRange` (page 831) for each contiguous range of glyphs that requires it.

Availability

Available in Mac OS X v10.3 and later.

substituteFontForFont

Returns a screen font suitable for use in place of *originalFont*, or simply returns *originalFont* if a screen font can't be used or isn't available.

```
public NSFont substituteFontForFont(NSFont originalFont)
```

Discussion

A screen font can be substituted if the receiver is set to use screen fonts and if no NSTextView associated with the receiver is scaled or rotated.

See Also

`usesScreenFonts` (page 856)

temporaryAttributesAtCharacterIndex

Returns the dictionary of temporary attributes for the character range specified in *effectiveCharRange* at character index *charIndex*.

```
public NSDictionary temporaryAttributesAtCharacterIndex(int charIndex, NSMutableRange
     effectiveCharRange)
```

Discussion

Temporary attributes are used only for onscreen drawing and are not persistent in any way. NSTextView uses them to color misspelled words when continuous spell checking is enabled. Currently the only temporary attributes that will be recognized are those related to colors and underlines.

See Also

`addTemporaryAttributes` (page 827)

`removeTemporaryAttribute` (page 843)

`setTemporaryAttributes` (page 849)

textContainerChangedGeometry

Invalidates the layout information, and possibly glyphs, for *aTextContainer* and all subsequent NSTextContainers.

```
public void textContainerChangedGeometry(NSTextContainer aTextContainer)
```

Discussion
This method is invoked automatically by other components of the text system; you should rarely need to invoke it directly. Subclasses of NSTextContainer, however, must invoke this method any time their size of shape changes (a text container that dynamically adjusts its shape to wrap text around placed graphics, for example, must do so when a graphic is added, moved, or removed).

textContainerChangedTextView

Updates information needed to manage NSTextView objects in *aTextContainer*.

```
public void textContainerChangedTextView(NSTextContainer aTextContainer)
```

Discussion
This method is invoked automatically by other components of the text system; you should rarely need to invoke it directly.

textContainerForGlyphAtIndex

Returns the NSTextContainer where the glyph at *glyphIndex* is laid out.

```
public NSTextContainer textContainerForGlyphAtIndex(int glyphIndex, NSMutableRange
     effectiveGlyphRange)
```

Discussion
If non-null, *effectiveGlyphRange* is set to the range for all glyphs laid out in that text container.

Performs glyph generation and layout if needed.

See Also
setTextContainerForGlyphRange (page 850)

textContainers

Returns the receiver's NSTextContainers.

```
public NSArray textContainers()
```

See Also
addTextContainer (page 827)
insertTextContainerAtIndex (page 837)
removeTextContainerAtIndex (page 843)

textStorage

Returns the receiver's NSTextStorage.

```
public NSTextStorage textStorage()
```

See Also
setTextStorage (page 850)

replaceTextStorage (page 844)

textStorageChanged

Invalidates glyph and layout information for a portion of the text in *aTextStorage*.

```
public void textStorageChanged(NSTextStorage aTextStorage, int mask, NSRange range,
     int lengthChange, NSRange invalidatedCharRange)
```

Discussion

This message is sent from NSTextStorage's processEditing (page 1591) method to indicate that its characters or attributes have been changed. This method invalidates glyphs and layout for the affected characters, and performs a soft invalidation of the layout information for all subsequent characters. *mask* specifies the nature of the changes. Its value is made by combining these options with the C bitwise OR operator:

Option	Meaning
TextStorageEditedAttributes	Attributes were added, removed, or changed.
TextStorageEditedCharacters	Characters were added, removed, or replaced.

The *range* argument indicates the extent of characters resulting from the edits. If the TextStorageEditedCharacters bit of *mask* is set, *lengthChange* gives the number of characters added to or removed from the original range (otherwise its value is irrelevant). For example, after replacing "The" with "Several" to produce the string "Several files couldn't be saved", *range* is {0, 7} and *lengthChange* is 4. The receiver uses this information to update its character-to-glyph mapping and to update the selection range based on the change.

The *invalidatedCharRange* argument represents the range of characters affected after attributes have been fixed. For example, deleting a paragraph separator character invalidates the layout information for all characters in the paragraphs that precede and follow the separator.

The textStorageChanged messages are sent in a series to each NSLayoutManager associated with the text storage object, so the NSLayoutManagers receiving them shouldn't edit *aTextStorage*. If one of them does, the *range*, *lengthChange*, and *invalidatedCharRange* arguments will be incorrect for all following NSLayoutManagers that receive the message.

See Also

invalidateLayoutForCharacterRange (page 838)

textViewForBeginningOfSelection

Returns the NSTextView containing the first glyph in the selection, or null if there's no selection or there isn't enough layout information to determine the text view.

```
public NSTextView textViewForBeginningOfSelection()
```

typesetterBehavior

Returns the current typesetter behavior value.

```
public int typesetterBehavior()
```

Discussion

Possible return values are described in "Constants" (page 856).

Availability

Available in Mac OS X v10.2 and later.

See Also

`setTypesetterBehavior` (page 850)

underlineGlyphRange

Calculates and draws underlining for the glyphs in *glyphRange*, which must belong to a single line fragment rectangle (as returned by `lineFragmentRectForGlyphAtIndex` (page 840)).

```
public void underlineGlyphRange(NSRange glyphRange, int underlineType, NSRect
    lineRect, NSRange lineGlyphRange, NSPoint containerOrigin)
```

Discussion

underlineType indicates the style of underlining to draw; NSLayoutManager accepts only `NSAttributedString.SingleUnderlineStyle`, but subclasses can define their own underline styles. *lineRect* is the line fragment rectangle containing the glyphs to draw underlining for, and *lineGlyphRange* is the range of all glyphs within that line fragment rectangle. *containerOrigin* is the origin of the line fragment rectangle's NSTextContainer in its NSTextView.

This method determines which glyphs actually need to be underlined based on *underlineType*. With `NSAttributedString.SingleUnderlineStyle`, for example, leading and trailing whitespace isn't underlined, but whitespace between visible glyphs is. A potential word-underline style would omit underlining on any whitespace. After determining which glyphs to draw underlining on, this method invokes `drawUnderlineForGlyphRange` (page 832) for each contiguous range of glyphs that requires it.

See Also

`textContainerForGlyphAtIndex` (page 853)

`textContainerOrigin` (page 1659) (NSTextView)

usedRectForTextContainer

Returns the bounding rectangle for the glyphs laid out in *aTextContainer*, which tells "how full" it is.

```
public NSRect usedRectForTextContainer(NSTextContainer aTextContainer)
```

Discussion

This rectangle is given in the coordinate system of *aTextContainer*.

See Also

`containerSize` (page 1557) (NSTextContainer)

usesScreenFonts

Returns true if the receiver calculates layout and displays text using screen fonts when possible, false otherwise.

```
public boolean usesScreenFonts()
```

See Also

setUsesScreenFonts (page 851)

substituteFontForFont (page 852)

Constants

These constants specify how a glyph is laid out relative to the previous glyph. The glyph inscription constants are possible values for the glyph attribute GlyphAttributeInscribe. The only constants that the text system currently uses are GlyphInscribeBase (for most glyphs) and GlyphInscribeOverstrike (for nonbase glyphs). Nonbase glyphs occur when diacritical marks are applied to a base character, and the font does not have a single glyph to represent the combination. For example, if a font did not contain a single glyph for ü, but did contain separate glyphs for u and ¨, then it could be rendered with a base glyph u followed by a nonbase glyph ¨. In that case the nonbase glyph would have the value GlyphInscribeOverstrike for the inscribe attribute.

Glyph inscriptions are set during glyph generation.

Constant	Description
GlyphInscribeBase	A base glyph; a character that the font can represent with a single glyph.
GlyphInscribeBelow	Glyph was rendered below the previous glyph.
GlyphInscribeAbove	Glyph was rendered above the previous glyph.
GlyphInscribeOverstrike	Glyph was rendered on top of the previous glyph.
GlyphInscribeOverBelow	Glyph was rendered on top and below the previous glyph.

The following constants are used by setTypesetterBehavior (page 850) and typesetterBehavior (page 854) to control the compatibility level of the typesetter.

Constant	Description
TypesetterLatestBehavior	Use the most current typesetter behavior in the current system version. For Mac OS X v10.2, this behavior is identical to TypesetterBehavior_10_2. If you use this behavior setting, you cannot necessarily rely on line width and height metrics remaining the same across different versions of Mac OS X.
TypesetterOriginalBehavior	Use the original typesetter behavior, as shipped with Mac OS X v10.1 and earlier.

Constant	Description
`TypesetterBehavior_10_2_WithCompatibility`	Perform typesetting as with `TypesetterBehavior_10_2`, but using line widths and height metric calculations that are the same as with `TypesetterOriginalBehavior`.
`TypesetterBehavior_10_2`	Use the typesetter behavior introduced in Mac OS X version 10.2. This typesetter behavior provides enhanced line and character spacing accuracy and supports more languages than the original typesetter behavior.
`TypesetterBehavior_10_3`	Use the typesetter behavior introduced in Mac OS X version 10.3.
`TypesetterBehavior_10_4`	Use the typesetter behavior introduced in Mac OS X version 10.4.

Delegate Methods

layoutManagerDidCompleteLayoutForTextContainer

Informs the delegate that *aLayoutManager* has finished laying out text in *aTextContainer*.

```
public abstract void layoutManagerDidCompleteLayoutForTextContainer(NSLayoutManager
     aLayoutManager, NSTextContainer aTextContainer, boolean flag)
```

Discussion
aTextContainer is `null` if there aren't enough containers to hold all the text; the delegate can use this information as a cue to add another container. If *flag* is `true`, *aLayoutManager* is finished laying out its text—this also means that *aTextContainer* is the final text container used by the layout manager. Delegates can use this information to show an indicator or background or to enable or disable a button that forces immediate layout of text.

layoutManagerDidInvalidateLayout

Informs the delegate that *aLayoutManager* has invalidated layout information (not glyph information).

```
public abstract void layoutManagerDidInvalidateLayout(NSLayoutManager aLayoutManager)
```

Discussion
This method is invoked only when layout was complete and then became invalidated for some reason. Delegates can use this information to show an indicator or background layout or to enable a button that forces immediate layout of text.

NSLevelIndicator

Inherits from	NSControl : NSView : NSResponder : NSObject
Package:	com.apple.cocoa.application
Availability	Available in Mac OS X v10.4 and later.

Overview

NSLevelIndicator is a subclass of NSControl that displays a value on a linear scale. Level indicators provide a visual representation of a level or amount of something, using discrete values. While similar to NSSlider, it provides a more customized visual feedback to the user. Level indicators do not have a "knob" indicating the current setting or allowing the user to adjust settings. The supported indicator styles include:

- A capacity style level indicator. The continuous mode for this style is often used to indicate conditions such as how much data is on hard disk. The discrete mode is similar to audio level indicators in audio playback applications. You can specify both a warning value and a critical value that provides additional visual feedback to the user.
- A ranking style level indicator. This is similar to the star ranking displays provided in iTunes and iPhoto. You can also specify your own ranking image.
- A relevancy style level indicator. This style is used to display the relevancy of a search result, for example in Mail.

NSLevelIndicator uses an NSLevelIndicatorCell (page 867) to implement much of the control's functionality. NSLevelIndicator provides cover methods for most of NSLevelIndicatorCell's methods, which invoke the corresponding cell method.

Tasks

Constructors

`NSLevelIndicator` (page 860)

Specifying Value Range

`minValue` (page 861)
Returns the receiver's minimum value.

`setMinValue` (page 863)
Sets the minimum value the receiver can represent to *`minValue`*.

`maxValue` (page 861)
Returns the receiver's maximum value.

`setMaxValue` (page 863)
Sets the maximum value the receiver can represent to *`maxValue`*.

`warningValue` (page 865)
Returns the receiver's warning value.

`setWarningValue` (page 864)
Sets the receiver's warning value to *`warningValue`*.

`criticalValue` (page 861)
Returns the receiver's critical value.

`setCriticalValue` (page 862)
Sets the receiver's critical value to *`criticalValue`*.

Managing Tick Marks

`tickMarkPosition` (page 864)
Returns how the receiver's tick marks are aligned with it.

`setTickMarkPosition` (page 864)
Sets where tick marks appear relative to the receiver.

`numberOfTickMarks` (page 862)
Returns the number of tick marks associated with the receiver.

`setNumberOfTickMarks` (page 864)
Sets the number of tick marks displayed by the receiver (which include those assigned to the minimum and maximum values) to *`count`*.

`numberOfMajorTickMarks` (page 862)
Returns the number of major tick marks associated with the receiver.

`setNumberOfMajorTickMarks` (page 863)
Sets the number of major tick marks displayed by the receiver.

`tickMarkValueAtIndex` (page 865)
Returns the receiver's value represented by the tick mark at index (the minimum-value tick mark has an index of 0).

`rectOfTickMarkAtIndex` (page 862)
Returns the bounding rectangle of the tick mark identified by *`index`* (the minimum-value tick mark is at index 0).

Constructors

NSLevelIndicator

```
public NSLevelIndicator()
```

Discussion
Creates a new NSLevelIndicator object.

```
public NSLevelIndicator(NSRect frameRect)
```

Discussion
Creates a new NSLevelIndicator object in *frameRect*.

Instance Methods

criticalValue

Returns the receiver's critical value.

```
public double criticalValue()
```

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

See Also
setCriticalValue (page 862)

maxValue

Returns the receiver's maximum value.

```
public double maxValue()
```

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

See Also
setMaxValue (page 863)

minValue

Returns the receiver's minimum value.

```
public double minValue()
```

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

See Also
setMinValue (page 863)

numberOfMajorTickMarks

Returns the number of major tick marks associated with the receiver.

```
public int numberOfMajorTickMarks()
```

Availability

Available in Mac OS X v10.4 and later.

See Also

`setNumberOfMajorTickMarks` (page 863)

numberOfTickMarks

Returns the number of tick marks associated with the receiver.

```
public int numberOfTickMarks()
```

Discussion

The tick marks assigned to the minimum and maximum values are included.

Availability

Available in Mac OS X v10.4 and later.

See Also

`setNumberOfTickMarks` (page 864)

rectOfTickMarkAtIndex

Returns the bounding rectangle of the tick mark identified by *index* (the minimum-value tick mark is at index 0).

```
public NSRect rectOfTickMarkAtIndex(int index)
```

Discussion

If no tick mark is associated with *index*, the method throws a `RangeException`.

Availability

Available in Mac OS X v10.4 and later.

setCriticalValue

Sets the receiver's critical value to *criticalValue*.

```
public void setCriticalValue(double criticalValue)
```

Availability

Available in Mac OS X v10.4 and later.

See Also

`criticalValue` (page 861)

setMaxValue

Sets the maximum value the receiver can represent to *`maxValue`*.

```
public void setMaxValue(double maxValue)
```

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

See Also
`maxValue` (page 861)

setMinValue

Sets the minimum value the receiver can represent to *`minValue`*.

```
public void setMinValue(double minValue)
```

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

See Also
`minValue` (page 861)

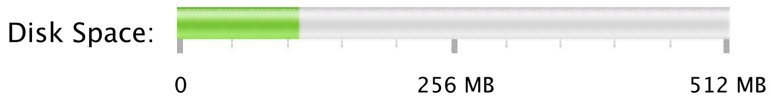
setNumberOfMajorTickMarks

Sets the number of major tick marks displayed by the receiver.

```
public void setNumberOfMajorTickMarks(int count)
```

Discussion
The *`count`* must be less than or equal to the number of tick marks returned by `numberOfTickMarks` (page 862). For example, if the number of tick marks is 11 and you specify 3 major tick marks, the resulting level indicator will display 3 major tickmarks alternating with 8 minor tick marks, as in the example shown in Figure 60-1.

Figure 60-1 Major and minor tick marks in a level indicator



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

See Also
`numberOfMajorTickMarks` (page 862)

setNumberOfTickMarks

Sets the number of tick marks displayed by the receiver (which include those assigned to the minimum and maximum values) to *count*.

```
public void setNumberOfTickMarks(int count)
```

Discussion
By default, this value is 0, and no tick marks appear. The number of tick marks assigned to a slider, along with the slider's minimum and maximum values, determines the values associated with the tick marks.

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

See Also
`numberOfTickMarks` (page 862)

setTickMarkPosition

Sets where tick marks appear relative to the receiver.

```
public void setTickMarkPosition(int position)
```

Discussion
This method has no effect if no tick marks have been assigned (that is, `numberOfTickMarks` (page 862) returns 0).

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

See Also
`tickMarkPosition` (page 864)

setWarningValue

Sets the receiver's warning value to *warningValue*.

```
public void setWarningValue(double warningValue)
```

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

See Also
`warningValue` (page 865)

tickMarkPosition

Returns how the receiver's tick marks are aligned with it.

```
public int tickMarkPosition()
```

Discussion

The default alignments are `TickMarkBelow` and `TickMarkLeft`.

Availability

Available in Mac OS X v10.4 and later.

See Also

`setTickMarkPosition` (page 864)

tickMarkValueAtIndex

Returns the receiver's value represented by the tick mark at index (the minimum-value tick mark has an index of 0).

```
public double tickMarkValueAtIndex(int index)
```

Availability

Available in Mac OS X v10.4 and later.

warningValue

Returns the receiver's warning value.

```
public double warningValue()
```

Availability

Available in Mac OS X v10.4 and later.

See Also

`setWarningValue` (page 864)

NSLevelIndicatorCell

Inherits from	NSActionCell : NSCell : NSObject
Package:	com.apple.cocoa.application
Availability	Available in Mac OS X v10.4 and later.

Overview

NSLevelIndicatorCell is a subclass of NSActionCell that provides several level indicator display styles including: capacity, ranking and relevancy. The capacity style provides both continuous and discrete modes.

Tasks

Constructors

`NSLevelIndicatorCell` (page 868)
Creates an NSLevelIndicatorCell object with the style specified by *`levelIndicatorStyle`*.

Specifying Value Range

`minValue` (page 870)
Returns the receiver's minimum value.

`setMinValue` (page 871)
Sets the minimum value the receiver can represent to *`minValue`*.

`maxValue` (page 869)
Returns the receiver's maximum value.

`setMaxValue` (page 871)
Sets the maximum value the receiver can represent to *`maxValue`*.

`levelIndicatorStyle` (page 869)
Returns the level indicator style of the receiver.

`setLevelIndicatorStyle` (page 871)
Sets the style of the receiver to *`levelIndicatorStyle`*.

`warningValue` (page 873)
Returns the receiver's warning value.

`setWarningValue` (page 872)
: Sets the receiver's warning value to *`warningValue`*.

`criticalValue` (page 869)
: Returns the receiver's critical value.

`setCriticalValue` (page 871)
: Sets the receiver's critical value to *`criticalValue`*.

Managing Tick Marks

`tickMarkPosition` (page 873)
: Returns how the receiver's tick marks are aligned with it.

`setTickMarkPosition` (page 872)
: Sets where tick marks appear relative to the receiver.

`numberOfTickMarks` (page 870)
: Returns the number of tick marks associated with the receiver.

`setNumberOfTickMarks` (page 872)
: Sets the number of tick marks displayed by the receiver (which include those assigned to the minimum and maximum values) to *`numberOfTickMarks`*.

`numberOfMajorTickMarks` (page 870)
: Returns the number of major tick marks associated with the receiver.

`setNumberOfMajorTickMarks` (page 872)
: Sets the number of major tick marks displayed by the receiver.

`tickMarkValueAtIndex` (page 873)
: Returns the receiver's value represented by the tick mark at index (the minimum-value tick mark has an index of 0).

`rectOfTickMarkAtIndex` (page 870)
: Returns the bounding rectangle of the tick mark identified by *`index`* (the minimum-value tick mark is at index 0).

Constructors

NSLevelIndicatorCell

```
public NSLevelIndicatorCell()
```

Discussion
Creates an NSLevelIndicatorCell object.

Creates an NSLevelIndicatorCell object with the style specified by *`levelIndicatorStyle`*.

```
public NSLevelIndicatorCell(int levelIndicatorStyle)
```

Discussion
The default value and minimum value are 0.0. The default maximum value is dependent on *levelIndicatorStyle*. For continuous styles, the default maximum value is 100.0. For discrete styles the default maximum value is 5.0.

Instance Methods

criticalValue

Returns the receiver's critical value.

```
public double criticalValue()
```

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

See Also
`setCriticalValue` (page 871)

levelIndicatorStyle

Returns the level indicator style of the receiver.

```
public int levelIndicatorStyle()
```

Discussion
Possible return values are described in "Constants" (page 873).

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

See Also
`setLevelIndicatorStyle` (page 871)

maxValue

Returns the receiver's maximum value.

```
public double maxValue()
```

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

See Also
`setLevelIndicatorStyle` (page 871)

minValue

Returns the receiver’s minimum value.

```
public double minValue()
```

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

See Also
`setMinValue` (page 871)

numberOfMajorTickMarks

Returns the number of major tick marks associated with the receiver.

```
public int numberOfMajorTickMarks()
```

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

See Also
`setNumberOfMajorTickMarks` (page 872)

numberOfTickMarks

Returns the number of tick marks associated with the receiver.

```
public int numberOfTickMarks()
```

Discussion
The tick marks assigned to the minimum and maximum values are included.

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

See Also
`setNumberOfTickMarks` (page 872)

rectOfTickMarkAtIndex

Returns the bounding rectangle of the tick mark identified by *`index`* (the minimum-value tick mark is at index 0).

```
public NSRect rectOfTickMarkAtIndex(int index)
```

Discussion
If no tick mark is associated with *`index`*, the method throws a `RangeException`.

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

setCriticalValue

Sets the receiver's critical value to *criticalValue*.

```
public void setCriticalValue(double criticalValue)
```

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

See Also
`criticalValue` (page 869)

setLevelIndicatorStyle

Sets the style of the receiver to *levelIndicatorStyle*.

```
public void setLevelIndicatorStyle(int levelIndicatorStyle)
```

Discussion
The available values of *levelIndicatorStyle* are described in "Constants" (page 873).

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

See Also
`levelIndicatorStyle` (page 869)

setMaxValue

Sets the maximum value the receiver can represent to *maxValue*.

```
public void setMaxValue(double maxValue)
```

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

See Also
`levelIndicatorStyle` (page 869)

setMinValue

Sets the minimum value the receiver can represent to *minValue*.

```
public void setMinValue(double minValue)
```

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

See Also
`minValue` (page 870)

setNumberOfMajorTickMarks

Sets the number of major tick marks displayed by the receiver.

```
public void setNumberOfMajorTickMarks(int count)
```

Discussion
The *count* must be less than or equal to the number of tick marks.

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

See Also
numberOfMajorTickMarks (page 870)

setNumberOfTickMarks

Sets the number of tick marks displayed by the receiver (which include those assigned to the minimum and maximum values) to *numberOfTickMarks*.

```
public void setNumberOfTickMarks(int count)
```

Discussion
By default, this value is 0, and no tick marks appear. The number of tick marks assigned to a slider, along with the slider's minimum and maximum values, determines the values associated with the tick marks.

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

See Also
numberOfTickMarks (page 870)

setTickMarkPosition

Sets where tick marks appear relative to the receiver.

```
public void setTickMarkPosition(int position)
```

Discussion
This method has no effect if no tick marks have been assigned (that is, numberOfTickMarks (page 870) returns 0).

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

See Also
tickMarkPosition (page 873)

setWarningValue

Sets the receiver's warning value to *warningValue*.

```
public void setWarningValue(double warningValue)
```

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

See Also
`warningValue` (page 873)

tickMarkPosition

Returns how the receiver's tick marks are aligned with it.

```
public int tickMarkPosition()
```

Discussion
The default alignments are `TickMarkBelow` and `TickMarkLeft`.

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

See Also
`setTickMarkPosition` (page 872)

tickMarkValueAtIndex

Returns the receiver's value represented by the tick mark at index (the minimum-value tick mark has an index of 0).

```
public double tickMarkValueAtIndex(int index)
```

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

warningValue

Returns the receiver's warning value.

```
public double warningValue()
```

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

See Also
`setWarningValue` (page 872)

Constants

The following constants specify the level indicator's style and are used by `levelIndicatorStyle` (page 869) and `setLevelIndicatorStyle` (page 871).

Constant	Description
`RatingLevelIndicatorStyle`	A style similar to the star ranking displays provided in iTunes and iPhoto. Available in Mac OS X v10.4 and later.
`DiscreteCapacityLevelIndicatorStyle`	A style similar to audio level indicators in iTunes. Available in Mac OS X v10.4 and later.
`ContinuousCapacityLevelIndicatorStyle`	A style that is often used to indicate conditions such as how much data is on a hard disk. Available in Mac OS X v10.4 and later.
`RelevancyLevelIndicatorStyle`	A style that is used to indicate the relevancy of a search result, for example in Mail. Available in Mac OS X v10.4 and later.

NSMatrix

Inherits from	NSControl : NSView : NSResponder : NSObject
Implements	NSCoding (NSResponder)
Package:	com.apple.cocoa.application
Companion guide	Matrix Programming Guide for Cocoa

Overview

NSMatrix is a class used for creating groups of NSCells that work together in various ways.

The cells 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). NSMatrix has the notion of a single selected cell, which is the cell that was most recently clicked or that was so designated by a `selectCellAtLocation` (page 896) or `selectCellWithTag` (page 896) message. The selected cell is the cell chosen for action messages except for `performClick` (page 318) (NSCell), which is assigned to the key cell. (The key cell is generally identical to the selected cell, but can be given click focus while leaving the selected cell unchanged.) If the user has selected multiple cells, the selected cell is the one lowest and furthest to the right in the matrix of cells.

Tasks

Constructors

`NSMatrix` (page 881)
Creates an empty NSMatrix.

Setting the Selection Mode

`mode` (page 892)

`setMode` (page 902)
Sets the selection mode of the receiver.

Configuring the NSMatrix

allowsEmptySelection (page 884)
Returns whether it's possible to have no cells selected in a radio-mode matrix.

isSelectionByRect (page 891)
Returns true if the user can select a rectangle of cells in the receiver by dragging the cursor, false otherwise.

setAllowsEmptySelection (page 899)

setSelectionByRect (page 903)
Sets whether the user can select a rectangle of cells in the receiver by dragging the cursor.

Setting the Cell Class

newCellClass (page 892)
Returns the subclass of NSCell that the receiver uses when creating new (empty) cells.

prototype (page 893)
Returns the prototype cell that's copied whenever a new cell needs to be created, or null if there is none.

setNewCellClass (page 902)
Configures the receiver to use instances of *aClass* when creating new cells.

setPrototype (page 903)
Sets the prototype cell that's copied whenever a new cell needs to be created.

Laying out the NSMatrix

addColumn (page 882)
Adds a new column of cells to the right of the last column, creating new cells as needed with makeCellAtLocation (page 891).

addColumnWithCells (page 883)
Adds a new column of cells to the right of the last column.

addRow (page 883)
Adds a new row of cells below the last row, creating new cells as needed with makeCellAtLocation (page 891).

addRowWithCells (page 884)
Adds a new row of cells below the last row.

cellFrameAtLocation (page 885)
Returns the frame rectangle of the cell that would be drawn at the location specified by *row* and *column* (whether or not the specified cell actually exists).

cellSize (page 886)
Returns the width and the height of each cell in the receiver (all cells in an NSMatrix are the same size).

`insertColumn` (page 889)
: Inserts a new column of cells before *`column`*, creating new cells if needed with `makeCellAtLocation` (page 891).

`insertColumnWithCells` (page 889)
: Inserts a new column of cells before *`column`*.

`insertRow` (page 890)
: Inserts a new row of cells before *`row`*, creating new cells if needed with `makeCellAtLocation` (page 891).

`insertRowWithCells` (page 890)
: Inserts a new row of cells before *`row`*.

`intercellSpacing` (page 890)
: Returns the vertical and horizontal spacing between cells in the receiver.

`makeCellAtLocation` (page 891)
: Creates a new cell at the location specified by *`row`* and *`column`* in the receiver.

`numberOfColumns` (page 893)
: Returns the number of columns in the receiver.

`numberOfRows` (page 893)
: Returns the number of rows in the receiver.

`putCellAtLocation` (page 893)
: Replaces the cell at the location specified by *`row`* and *`column`* with *`newCell`* and redraws the cell.

`removeColumn` (page 894)
: Removes the column at position *`column`* from the receiver and autoreleases the column's cells.

`removeRow` (page 894)
: Removes the row at position *`row`* from the receiver and autoreleases the row's cells.

`renewRowsAndColumns` (page 894)
: Changes the number of rows and columns in the receiver.

`setCellSize` (page 900)
: Sets the width and height of each of the cells in the receiver to those in *`aSize`*.

`setIntercellSpacing` (page 902)
: Sets the vertical and horizontal spacing between cells in the receiver to *`aSize`*.

`sortUsingMethod` (page 905)
: Sorts the receiver's cells in ascending order as defined by the comparison method *`comparator`*.

Modifying Individual Cells

`setStateAtLocation` (page 904)
: Sets the state of the cell at *`row`* and *`column`* to *`value`*.

`setToolTip` (page 905)
: Sets the tooltip for *`cell`* to be *`toolTipString`*.

`toolTip` (page 907)
: Returns the string used as the tooltip for *`cell`*.

Selecting Cells

`deselectAllCells` (page 887)
Deselects all cells in the receiver and, if necessary, redisplays the receiver.

`deselectSelectedCell` (page 887)
Deselects the selected cell or cells.

`keyCell` (page 891)
Returns the cell that will be clicked when the user presses the Space bar.

`selectAll` (page 896)
Selects and highlights all cells in the receiver, except for editable text cells and disabled cells.

`selectCellAtLocation` (page 896)
Selects the cell at the specified row and column within the receiver.

`selectCellWithTag` (page 896)
If the receiver has at least one cell whose tag is equal to *anInt*, the last cell (when viewing the matrix as a row-ordered array) is selected.

`selectedCell` (page 896)
Returns the most recently selected cell, or `null` if no cell is selected.

`selectedCells` (page 897)
Returns an array containing all of the receiver's highlighted cells plus its selected cell.

`selectedColumn` (page 897)
Returns the column number of the selected cell, or –1 if no cells are selected.

`selectedRow` (page 897)
Returns the row number of the selected cell, or –1 if no cells are selected.

`setKeyCell` (page 902)
Sets the cell that will be clicked when the user presses the Space bar to *aCell*.

`setSelectionWithAnchor` (page 903)
Programmatically selects a range of cells.

Finding Cells

`cellAtLocation` (page 885)
Returns the NSCell object at the location specified by *row* and *column*, or `null` if either *row* or *column* is outside the bounds of the receiver.

`cellWithTag` (page 886)
Searches the receiver and returns the last (when viewing the matrix as a row-ordered array) NSCell object that has a tag matching *anInt*, or `null` if no such cell exists.

`cells` (page 885)
Returns an NSArray that contains the receiver's cells.

`columnForPoint` (page 886)
Returns the column for the cell within which the specified point lies.

`columnOfCell` (page 886)
Searches the receiver for *aCell* and returns the column of the cell.

`rowForPoint` (page 895)
Returns the row for the cell within which the specified point lies.

rowOfCell (page 895)
Searches the receiver for *aCell* and returns the row of the cell.

Modifying Graphics Attributes

backgroundColor (page 885)
Returns the color used to draw the background of the receiver (the space between the cells).

cellBackgroundColor (page 885)
Returns the color used to fill the background of the receiver's cells.

drawsBackground (page 888)
Returns whether the receiver draws its background (the space between the cells).

drawsCellBackground (page 888)
Returns whether the receiver draws the background within each of its cells.

setBackgroundColor (page 900)
Sets the background color for the receiver to *aColor* and redraws the receiver.

setCellBackgroundColor (page 900)
Sets the background color for the cells in the receiver to *aColor*.

setDrawsBackground (page 901)
Sets whether the receiver draws its background (the space between the cells) to *flag*.

setDrawsCellBackground (page 901)
Sets whether the receiver draws the background within each of its cells to *flag*.

Editing Text in Cells

selectText (page 897)

selectTextAtLocation (page 898)

textDidBeginEditing (page 906)
Invoked when *notification* is posted indicating that there's a change in the text after the receiver gains first responder status.

textDidChange (page 906)
Invoked when *notification* is posted indicating a key-down event or paste operation that changes the receiver's contents.

textDidEndEditing (page 906)
Invoked when *notification* is posted indicating that text editing ends.

textShouldBeginEditing (page 907)
Invoked to let the NSTextField respond to impending changes to its text.

textShouldEndEditing (page 907)
Invoked to let the NSTextField respond to impending loss of first-responder status.

Setting Tab Key Behavior

`setTabKeyTraversesCells` (page 904)
Sets whether pressing the Tab key advances the key cell to the next selectable cell in the receiver.

`tabKeyTraversesCells` (page 905)
Returns whether pressing the Tab key advances the key cell to the next selectable cell in the receiver.

Assigning a Delegate

`delegate` (page 887)
Returns the delegate for messages from the field editor.

`setDelegate` (page 900)
Sets the delegate for messages from the field editor to *`anObject`*.

Resizing the Matrix and Its Cells

`autosizesCells` (page 884)

`setAutosizesCells` (page 899)
Sets whether the cell sizes change when the receiver is resized.

`setValidateSize` (page 905)
If *`flag`* is `true`, then the size information in the receiver is assumed to be correct.

`sizeToCells` (page 905)
Changes the width and the height of the receiver's frame so it exactly contains the cells.

Scrolling

`isAutoscroll` (page 891)
Returns whether the receiver will be automatically scrolled whenever the cursor is dragged outside the receiver after a mouse-down event within its bounds.

`scrollCellAtLocationToVisible` (page 895)
If the receiver is in a scrolling view, and *`row`* and *`column`* represent a valid cell within the receiver, this method scrolls the receiver so the specified cell is visible.

`setAutoscroll` (page 899)

`setScrollable` (page 903)

Displaying

`drawCellAtLocation` (page 888)
Displays the cell at the specified row and column, providing that *`row`* and *`column`* reference a cell within the receiver.

`highlightCellAtLocation` (page 889)
: Assuming that *row* and *column* indicate a valid cell within the receiver, this method highlights (if *flag* is `true`) or unhighlights (if *flag* is `false`) the specified cell.

Target and Action

`doubleAction` (page 888)
: Returns the action method sent by the receiver to its target when the user double-clicks an entry, or `NULL` if there's no double-click action.

`sendAction` (page 898)
: If the selected cell has both an action and a target, its action is sent to its target.

`sendActionToTargetForAllCells` (page 898)
: Iterates through all cells in the receiver (if *flag* is `true`) or just the selected cells in the receiver (if *flag* is `false`), sending *aSelector* to *anObject* for each.

`sendDoubleAction` (page 899)

`setDoubleAction` (page 901)
: Makes *aSelector* the action sent to the target of the receiver when the user double-clicks a cell.

Handling Event and Action Messages

`acceptsFirstMouse` (page 882)
: Returns `false` if the selection mode of the receiver is `ListMode`, `true` if the receiver is in any other selection mode.

`mouseDown` (page 892)
: Responds to *theEvent* mouse-down event.

`mouseDownFlags` (page 892)
: Returns the flags in effect at the mouse-down event that started the current tracking session.

`performKeyEquivalent` (page 893)

Managing the Cursor

`resetCursorRects` (page 895)
: Resets cursor rectangles so the cursor becomes an I-beam over text cells.

Constructors

NSMatrix

Creates an empty NSMatrix.

```
public NSMatrix()
```

Creates an NSMatrix, with default parameters in the frame specified by *frameRect*.

```
public NSMatrix(NSRect frameRect)
```

Discussion
The new NSMatrix contains no rows or columns. The default mode is `RadioMode`. The default cell class is NSActionCell.

Creates an NSMatrix, in the frame specified by *frameRect*

```
public NSMatrix(NSRect frameRect, int aMode, NSCell aCell, int numRows, int
    numColumns)
```

Discussion
The new NSMatrix contains *numRows* rows and *numColumns* columns. *aMode* is set as the tracking mode for the NSMatrix and can be one of the modes described in "Constants" (page 908).

The new Matrix creates cells by copying *aCell*, which should be an instance of a subclass of NSCell.

Creates an NSMatrix, in the frame specified by *frameRect*.

```
public NSMatrix(NSRect frameRect, int aMode, Class aClass, int numRows, int
    numColumns)
```

Discussion
The new NSMatrix contains *numRows* rows and *numColumns* columns. *aMode* is set as the tracking mode for the NSMatrix and can be one of the modes described in "Constants" (page 908).

The new NSMatrix creates and uses cells of class *aClass*.

Instance Methods

acceptsFirstMouse

Returns `false` if the selection mode of the receiver is `ListMode`, `true` if the receiver is in any other selection mode.

```
public boolean acceptsFirstMouse(NSEvent theEvent)
```

Discussion
The receiver does not accept first mouse in `ListMode` to prevent the loss of multiple selections. The NSEvent parameter, *theEvent*, is ignored.

See Also
`mode` (page 892)

addColumn

Adds a new column of cells to the right of the last column, creating new cells as needed with `makeCellAtLocation` (page 891).

```
public void addColumn()
```

Discussion

This method throws a `RangeException` if there are 0 rows or 0 columns. Use `renewRowsAndColumns` (page 894) to add new cells to an empty matrix.

If the number of rows or columns in the receiver has been changed with `renewRowsAndColumns` (page 894), new cells are created only if they are needed. This fact allows you to grow and shrink an NSMatrix without repeatedly creating and freeing the cells.

This method redraws the receiver. Your code may need to send `sizeToCells` (page 905) after sending this method to resize the receiver to fit the newly added cells.

See Also

`newCellClass` (page 892)
`insertColumn` (page 889)
`prototype` (page 893)
`addRow` (page 883)

addColumnWithCells

Adds a new column of cells to the right of the last column.

```
public void addColumnWithCells(NSArray newCells)
```

Discussion

The new column is filled with objects from *`newCells`*, starting with the object at index 0. Each object in *`newCells`* should be an instance of NSCell or one of its subclasses (usually NSActionCell). *`newCells`* should have a sufficient number of cells to fill the entire column. Extra cells are ignored, unless the matrix is empty. In that case, a matrix is created with one column and enough rows for all the elements of *`newCells`*.

This method redraws the receiver. Your code may need to send `sizeToCells` (page 905) after sending this method to resize the receiver to fit the newly added cells.

See Also

`insertColumnWithCells` (page 889)
`addRowWithCells` (page 884)

addRow

Adds a new row of cells below the last row, creating new cells as needed with `makeCellAtLocation` (page 891).

```
public void addRow()
```

Discussion

This method throws a `RangeException` if there are 0 rows or 0 columns. Use `renewRowsAndColumns` (page 894) to add new cells to an empty matrix.

If the number of rows or columns in the receiver has been changed with `renewRowsAndColumns` (page 894), then new cells are created only if they are needed. This fact allows you to grow and shrink an NSMatrix without repeatedly creating and freeing the cells.

This method redraws the receiver. Your code may need to send `sizeToCells` (page 905) after sending this method to resize the receiver to fit the newly added cells.

See Also

`newCellClass` (page 892)
`insertRow` (page 890)
`prototype` (page 893)
`addColumn` (page 882)

addRowWithCells

Adds a new row of cells below the last row.

```
public void addRowWithCells(NSArray newCells)
```

Discussion

The new row is filled with objects from *newCells*, starting with the object at index 0. Each object in *newCells* should be an instance of NSCell or one of its subclasses (usually NSActionCell). *newCells* should have a sufficient number of cells to fill the entire row. Extra cells are ignored, unless the matrix is empty. In that case, a matrix is created with one row and enough columns for all the elements of *newCells*.

This method redraws the receiver. Your code may need to send `sizeToCells` (page 905) after sending this method to resize the receiver to fit the newly added cells.

See Also

`insertRowWithCells` (page 890)
`addColumnWithCells` (page 883)

allowsEmptySelection

Returns whether it’s possible to have no cells selected in a radio-mode matrix.

```
public boolean allowsEmptySelection()
```

See Also

`mode` (page 892)
`setAllowsEmptySelection` (page 899)

autosizesCells

```
public boolean autosizesCells()
```

Discussion

Returns `true` if cells are resized proportionally to the receiver when its size changes (and intercell spacing is kept constant). Returns `false` if the cell size and intercell spacing remain constant.

See Also

`setAutosizesCells` (page 899)

backgroundColor

Returns the color used to draw the background of the receiver (the space between the cells).

```
public NSColor backgroundColor()
```

See Also

`cellBackgroundColor` (page 885)

`drawsBackground` (page 888)

`setBackgroundColor` (page 900)

cellAtLocation

Returns the NSCell object at the location specified by *`row`* and *`column`*, or `null` if either *`row`* or *`column`* is outside the bounds of the receiver.

```
public NSCell cellAtLocation(int row, int column)
```

See Also

`columnOfCell` (page 886)

`rowOfCell` (page 895)

cellBackgroundColor

Returns the color used to fill the background of the receiver's cells.

```
public NSColor cellBackgroundColor()
```

See Also

`backgroundColor` (page 885)

`drawsCellBackground` (page 888)

`setCellBackgroundColor` (page 900)

cellFrameAtLocation

Returns the frame rectangle of the cell that would be drawn at the location specified by *`row`* and *`column`* (whether or not the specified cell actually exists).

```
public NSRect cellFrameAtLocation(int row, int column)
```

See Also

`cellSize` (page 886)

cells

Returns an NSArray that contains the receiver's cells.

```
public NSArray cells()
```

Discussion

The cells in the array are row-ordered; that is, the first row of cells appears first in the array, followed by the second row, and so forth.

See Also

`cellAtLocation` (page 885)

cellSize

Returns the width and the height of each cell in the receiver (all cells in an NSMatrix are the same size).

```
public NSSize cellSize()
```

See Also

`cellFrameAtLocation` (page 885)

`intercellSpacing` (page 890)

`setCellSize` (page 900)

cellWithTag

Searches the receiver and returns the last (when viewing the matrix as a row-ordered array) NSCell object that has a tag matching *anInt*, or `null` if no such cell exists.

```
public NSCell cellWithTag(int anInt)
```

See Also

`selectCellWithTag` (page 896)

`setTag` (page 51) (NSActionCell)

columnForPoint

Returns the column for the cell within which the specified point lies.

```
public int columnForPoint(NSPoint aPoint)
```

Discussion

If *aPoint* falls outside the bounds of the receiver or lies within an intercell spacing, this method returns –1.

Make sure *aPoint* is in the coordinate system of the receiver.

See Also

`rowForPoint` (page 895)

`columnOfCell` (page 886)

columnOfCell

Searches the receiver for *aCell* and returns the column of the cell.

```
public int columnOfCell(NSCell aCell)
```

Discussion

If *aCell* is not found within the receiver, this method returns –1.

See Also

rowOfCell (page 895)

columnForPoint (page 886)

delegate

Returns the delegate for messages from the field editor.

```
public Object delegate()
```

See Also

textShouldBeginEditing (page 907)

textShouldEndEditing (page 907)

setDelegate (page 900)

deselectAllCells

Deselects all cells in the receiver and, if necessary, redisplays the receiver.

```
public void deselectAllCells()
```

Discussion

If the selection mode is RadioMode and empty selection is not allowed, this method does nothing.

See Also

allowsEmptySelection (page 884)

mode (page 892)

selectAll (page 896)

deselectSelectedCell

Deselects the selected cell or cells.

```
public void deselectSelectedCell()
```

Discussion

If the selection mode is RadioMode and empty selection is not allowed, or if nothing is currently selected, this method does nothing. This method doesn't redisplay the receiver.

See Also

allowsEmptySelection (page 884)

mode (page 892)

selectCellAtLocation (page 896)

doubleAction

Returns the action method sent by the receiver to its target when the user double-clicks an entry, or `NULL` if there's no double-click action.

```
public NSSelector doubleAction()
```

Discussion

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

`action` (page 448) (NSControl)
`target` (page 463) (NSControl)
`ignoresMultiClick` (page 451) (NSControl)
`sendDoubleAction` (page 899)
`setDoubleAction` (page 901)

drawCellAtLocation

Displays the cell at the specified row and column, providing that *row* and *column* reference a cell within the receiver.

```
public void drawCellAtLocation(int row, int column)
```

See Also

`drawCell` (page 450) (NSControl)
`drawCellInside` (page 450) (NSControl)

drawsBackground

Returns whether the receiver draws its background (the space between the cells).

```
public boolean drawsBackground()
```

See Also

`backgroundColor` (page 885)
`drawsCellBackground` (page 888)
`setDrawsBackground` (page 901)

drawsCellBackground

Returns whether the receiver draws the background within each of its cells.

```
public boolean drawsCellBackground()
```

See Also

`cellBackgroundColor` (page 885)
`drawsBackground` (page 888)

`setDrawsCellBackground` (page 901)

highlightCellAtLocation

Assuming that *row* and *column* indicate a valid cell within the receiver, this method highlights (if *flag* is `true`) or unhighlights (if *flag* is `false`) the specified cell.

```
public void highlightCellAtLocation(boolean flag, int row, int column)
```

insertColumn

Inserts a new column of cells before *column*, creating new cells if needed with `makeCellAtLocation` (page 891).

```
public void insertColumn(int column)
```

Discussion

If *column* is greater than the number of columns in the receiver, enough columns are created to expand the receiver to be *column* columns wide. This method redraws the receiver. Your code may need to send `sizeToCells` (page 905) after sending this method to resize the receiver to fit the newly added cells.

If the number of rows or columns in the receiver has been changed with `renewRowsAndColumns` (page 894), new cells are created only if they're needed. This fact allows you to grow and shrink an NSMatrix without repeatedly creating and freeing the cells.

See Also

`addColumn` (page 882)
`insertRow` (page 890)

insertColumnWithCells

Inserts a new column of cells before *column*.

```
public void insertColumnWithCells(int column, NSArray newCells)
```

Discussion

The new column is filled with objects from *newCells*, starting with the object at index 0. Each object in *newCells* should be an instance of NSCell or one of its subclasses (usually NSActionCell). If *column* is greater than the number of columns in the receiver, enough columns are created to expand the receiver to be *column* columns wide. *newCells* should either be empty or contain a sufficient number of cells to fill each new column. If *newCells* is `null` or an array with no elements, the call is equivalent to calling `insertColumn` (page 889). Extra cells are ignored, unless the matrix is empty. In that case, a matrix is created with one column and enough rows for all the elements of *newCells*.

This method redraws the receiver. Your code may need to send `sizeToCells` (page 905) after sending this method to resize the receiver to fit the newly added cells.

See Also

`addColumnWithCells` (page 883)
`insertRowWithCells` (page 890)

insertRow

Inserts a new row of cells before *row*, creating new cells if needed with `makeCellAtLocation` (page 891).

```
public void insertRow(int row)
```

Discussion

If *row* is greater than the number of rows in the receiver, enough rows are created to expand the receiver to be *row* rows high. This method redraws the receiver. Your code may need to send `sizeToCells` (page 905) after sending this method to resize the receiver to fit the newly added cells.

If the number of rows or columns in the receiver has been changed with `renewRowsAndColumns` (page 894), then new cells are created only if they're needed. This fact allows you to grow and shrink an NSMatrix without repeatedly creating and freeing the cells.

See Also

`addRow` (page 883)
`insertColumn` (page 889)

insertRowWithCells

Inserts a new row of cells before *row*.

```
public void insertRowWithCells(int row, NSArray newCells)
```

Discussion

The new row is filled with objects from *newCells*, starting with the object at index 0. Each object in *newCells* should be an instance of NSCell or one of its subclasses (usually NSActionCell). If *row* is greater than the number of rows in the receiver, enough rows are created to expand the receiver to be *row* rows high. *newCells* should either be empty or contain a sufficient number of cells to fill each new row. If *newCells* is `null` or an array with no elements, the call is equivalent to calling `insertRow` (page 890). Extra cells are ignored, unless the matrix is empty. In that case, a matrix is created with one row and enough columns for all the elements of *newCells*.

This method redraws the receiver. Your code may need to send `sizeToCells` (page 905) after sending this method to resize the receiver to fit the newly added cells.

See Also

`addRowWithCells` (page 884)
`insertColumnWithCells` (page 889)

intercellSpacing

Returns the vertical and horizontal spacing between cells in the receiver.

```
public NSSize intercellSpacing()
```

See Also

`cellSize` (page 886)
`setIntercellSpacing` (page 902)

isAutoscroll

Returns whether the receiver will be automatically scrolled whenever the cursor is dragged outside the receiver after a mouse-down event within its bounds.

```
public boolean isAutoscroll()
```

See Also

`scrollCellAtLocationToVisible` (page 895)

`setScrollable` (page 903)

isSelectionByRect

Returns `true` if the user can select a rectangle of cells in the receiver by dragging the cursor, `false` otherwise.

```
public boolean isSelectionByRect()
```

See Also

`setSelectionWithAnchor` (page 903)

keyCell

Returns the cell that will be clicked when the user presses the Space bar.

```
public NSCell keyCell()
```

See Also

`tabKeyTraversesCells` (page 905)

`setKeyCell` (page 902)

makeCellAtLocation

Creates a new cell at the location specified by *row* and *column* in the receiver.

```
public NSCell makeCellAtLocation(int row, int column)
```

Discussion

If the receiver has a prototype cell, it's copied to create the new cell. If not, and if the receiver has a cell class set, it creates an instance of that class. If the receiver hasn't had either a prototype cell or a cell class set, `makeCellAtLocation` creates an NSActionCell. Returns the newly created cell.

Your code should never invoke this method directly; it's used by `addRow` (page 883) and other methods when a cell must be created. It may be overridden to provide more specific initialization of cells.

See Also

`addColumn` (page 882)

`addRow` (page 883)

`insertColumn` (page 889)

`insertRow` (page 890)

`setNewCellClass` (page 902)

`setPrototype` (page 903)

mode

```
public int mode()
```

Discussion

Returns the selection mode of the receiver. Possible return values are listed in "Constants" (page 908).

See Also

`setMode` (page 902)

mouseDown

Responds to *theEvent* mouse-down event.

```
public void mouseDown(NSEvent theEvent)
```

Discussion

A mouse-down event in a text cell initiates editing mode. A double click in any cell type except a text cell sends the double-click action of the receiver (if there is one) in addition to the single-click action.

Your code should never invoke this method, but you may override it to implement different mouse tracking than NSMatrix does. The response of the receiver depends on its selection mode, as explained in the class description.

See Also

`sendAction` (page 898)

`sendDoubleAction` (page 899)

mouseDownFlags

Returns the flags in effect at the mouse-down event that started the current tracking session.

```
public int mouseDownFlags()
```

Discussion

NSMatrix's `mouseDown` (page 892) method obtains these flags by sending a `modifierFlags` (page 616) message to the event passed into `mouseDown` (page 892). Use this method if you want to access these flags. This method is valid only during tracking; it isn't useful if the target of the receiver initiates another tracking loop as part of its action method (as a cell that pops up a pop-up list does, for example).

See Also

`setEventMaskForSendingAction` (page 457) (NSCell)

newCellClass

Returns the subclass of NSCell that the receiver uses when creating new (empty) cells.

```
public Class newCellClass()
```

See Also

`prototype` (page 893)

`makeCellAtLocation` (page 891)

setNewCellClass (page 902)

numberOfColumns

Returns the number of columns in the receiver.

```
public int numberOfColumns()
```

numberOfRows

Returns the number of rows in the receiver.

```
public int numberOfRows()
```

performKeyEquivalent

```
public boolean performKeyEquivalent(NSEvent theEvent)
```

Discussion

If there's a cell in the receiver that has a key equivalent equal to the character in [*theEvent* charactersIgnoringModifiers (page 612)] (taking into account any key modifier flags) and that cell is enabled, 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 `true` if a cell in the receiver responds to the key equivalent in *theEvent*, `false` if no cell responds.

Your code should never send this message—it is sent when the receiver 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.

prototype

Returns the prototype cell that's copied whenever a new cell needs to be created, or `null` if there is none.

```
public NSCell prototype()
```

See Also

makeCellAtLocation (page 891)
setPrototype (page 903)

putCellAtLocation

Replaces the cell at the location specified by *row* and *column* with *newCell* and redraws the cell.

```
public void putCellAtLocation(NSCell newCell, int row, int column)
```

removeColumn

Removes the column at position *column* from the receiver and autoreleases the column's cells.

```
public void removeColumn(int column)
```

Discussion

Redraws the receiver. Your code should normally send sizeToCells (page 905) after invoking this method to resize the receiver so it fits the reduced cell count.

See Also

removeRow (page 894)

addColumn (page 882)

insertColumn (page 889)

removeRow

Removes the row at position *row* from the receiver and autoreleases the row's cells.

```
public void removeRow(int row)
```

Discussion

Redraws the receiver. Your code should normally send sizeToCells (page 905) after invoking this method to resize the receiver so it fits the reduced cell count.

See Also

removeColumn (page 894)

addRow (page 883)

insertRow (page 890)

renewRowsAndColumns

Changes the number of rows and columns in the receiver.

```
public void renewRowsAndColumns(int newRows, int newCols)
```

Discussion

This method uses the same cells as before, creating new cells only if the new size is larger; it never frees cells. Doesn't redisplay the receiver. Your code should normally send sizeToCells (page 905) after invoking this method to resize the receiver so it fits the changed cell arrangement. This method deselects all cells in the receiver.

See Also

addColumn (page 882)

addRow (page 883)

removeColumn (page 894)

removeRow (page 894)

resetCursorRects

Resets cursor rectangles so the cursor becomes an I-beam over text cells.

```
public void resetCursorRects()
```

Discussion

It does this by sending `resetCursorRect` (page 319) to each cell in the receiver. Any cell that has a cursor rectangle to set up should then send `addCursorRect` (page 1739) back to the receiver.

See Also

`resetCursorRect` (page 319) (NSCell)

`addCursorRect` (page 1739) (NSView)

rowForPoint

Returns the row for the cell within which the specified point lies.

```
public int rowForPoint(NSPoint aPoint)
```

Discussion

If *aPoint* falls outside the bounds of the receiver or lies within an intercell spacing, this method returns –1.

Make sure *aPoint* is in the coordinate system of the receiver.

See Also

`columnOfCell` (page 886)

`rowOfCell` (page 895)

rowOfCell

Searches the receiver for *aCell* and returns the row of the cell.

```
public int rowOfCell(NSCell aCell)
```

Discussion

If *aCell* is not found within the receiver, this method returns –1.

See Also

`columnOfCell` (page 886)

`rowForPoint` (page 895)

scrollCellAtLocationToVisible

If the receiver is in a scrolling view, and *row* and *column* represent a valid cell within the receiver, this method scrolls the receiver so the specified cell is visible.

```
public void scrollCellAtLocationToVisible(int row, int column)
```

See Also

`scrollRectToVisible` (page 1772) (NSView)

selectAll

Selects and highlights all cells in the receiver, except for editable text cells and disabled cells.

```
public void selectAll(Object sender)
```

Discussion

Redisplays the receiver. *sender* is ignored.

See Also

`selectCell` (page 453) (NSControl)

selectCellAtLocation

Selects the cell at the specified row and column within the receiver.

```
public void selectCellAtLocation(int row, int column)
```

Discussion

If the specified cell is an editable text cell, its text is selected. If either *row* or *column* is –1, then the current selection is cleared (unless the receiver is an `RadioMode` and doesn't allow empty selection). Redraws the affected cells.

See Also

`allowsEmptySelection` (page 884)

`mode` (page 892)

`selectCell` (page 453) (NSControl)

selectCellWithTag

If the receiver has at least one cell whose tag is equal to *anInt*, the last cell (when viewing the matrix as a row-ordered array) is selected.

```
public boolean selectCellWithTag(int anInt)
```

Discussion

If the specified cell is an editable text cell, its text is selected. Returns `true` if the receiver contains a cell whose tag matches *anInt*, or `false` if no such cell exists.

See Also

`cellWithTag` (page 886)

`selectCell` (page 453) (NSControl)

selectedCell

Returns the most recently selected cell, or `null` if no cell is selected.

```
public NSCell selectedCell()
```

Discussion

If more than one cell is selected, this method returns the cell that is lowest and farthest to the right in the receiver.

selectedCells

Returns an array containing all of the receiver's highlighted cells plus its selected cell.

```
public NSArray selectedCells()
```

Discussion

See the class description for a discussion of the selected cell.

As an alternative to using `setSelectionWithAnchor` (page 903) for programmatically making discontiguous selections of cells in a matrix, you could first set the single selected cell and then set subsequent cells to be highlighted; aftewards you can call `selectedCells` (page 897) to obtain the selection of cells.

See Also

`setHighlighted` (page 326) (NSCell)

`selectedCell` (page 896)

selectedColumn

Returns the column number of the selected cell, or –1 if no cells are selected.

```
public int selectedColumn()
```

Discussion

If cells in multiple columns are selected, this method returns the number of the last (rightmost) column containing a selected cell.

selectedRow

Returns the row number of the selected cell, or –1 if no cells are selected.

```
public int selectedRow()
```

Discussion

If cells in multiple rows are selected, this method returns the number of the last row containing a selected cell.

selectText

```
public void selectText(Object sender)
```

Discussion

If the currently selected cell is editable and enabled, its text is selected. Otherwise, the key cell is selected.

See Also

`keyCell` (page 891)

`selectText` (page 1569) (NSTextField)

selectTextAtLocation

```
public Object selectTextAtLocation(int row, int column)
```

Discussion

If *row* and *column* indicate a valid cell within the receiver, and that cell is both editable and selectable, this method selects and then returns the specified cell. If the cell specified by *row* and *column* is either not editable or not selectable, this method does nothing, and returns `null`. Finally, if *row* and *column* indicate a cell that is outside the receiver, this method does nothing and returns the receiver.

See Also

`selectText` (page 897)

sendAction

If the selected cell has both an action and a target, its action is sent to its target.

```
public boolean sendAction()
```

Discussion

If the cell has an action but no target, its action is sent to the target of the receiver. If the cell doesn't have an action, or if there is no selected cell, the receiver sends its own action to its target. Returns `true` if an action was successfully sent to a target.

If the selected cell is disabled, this method does nothing and returns `false`.

See Also

`sendDoubleAction` (page 899)

`action` (page 305) (NSCell)

`target` (page 336) (NSCell)

sendActionToTargetForAllCells

Iterates through all cells in the receiver (if *flag* is `true`) or just the selected cells in the receiver (if *flag* is `false`), sending *aSelector* to *anObject* for each.

```
public void sendActionToTargetForAllCells(NSSelector aSelector, Object anObject,
    boolean flag)
```

Discussion

Iteration begins with the cell in the upper-left corner of the receiver, proceeding through the appropriate entries in the first row, then on to the next.

The *aSelector* argument 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 `boolean`. If *aSelector* returns `false` for any cell, `sendActionToTargetForAllCells` terminates immediately, without sending the message for the remaining cells. If it returns `true`, `sendActionToTargetForAllCells` proceeds to the next cell.

This method is not invoked to send action messages to target objects in response to mouse-down events in the receiver. Instead, you can invoke it if you want to have multiple cells in an NSMatrix 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*.

sendDoubleAction

```
public void sendDoubleAction()
```

Discussion
If the receiver has a double-click action, `sendDoubleAction` sends that message to the target of the receiver. If not, then if the selected cell (as returned by `selectedCell` (page 896)) has an action, that message is sent to the selected cell's target. Finally, if the selected cell also has no action, then the single-click action of the receiver is sent to the target of the receiver.

If the selected cell is disabled, this method does nothing.

Your code shouldn't invoke this method; it's sent in response to a double-click event in the NSMatrix. Override it if you need to change the search order for an action to send.

See Also
`sendAction` (page 898)
`ignoresMultiClick` (page 451) (NSControl)

setAllowsEmptySelection

```
public void setAllowsEmptySelection(boolean flag)
```

Discussion
If *`flag`* is `true`, then the receiver will allow one or zero cells to be selected. If *`flag`* is `false`, then the receiver will allow one and only one cell (not zero cells) to be selected. This setting has effect only in the `RadioMode` selection mode.

See Also
`allowsEmptySelection` (page 884)

setAutoscroll

```
public void setAutoscroll(boolean flag)
```

Discussion
If *`flag`* is `true` and the receiver is in a scrolling view, it will be automatically scrolled whenever the cursor is dragged outside the receiver after a mouse-down event within the bounds of the receiver.

setAutosizesCells

Sets whether the cell sizes change when the receiver is resized.

```
public void setAutosizesCells(boolean flag)
```

Discussion
If *`flag`* is `true`, then whenever the receiver is resized, the sizes of the cells change in proportion, keeping the intercell space constant; further, this method verifies that the cell sizes and intercell spacing add up to the exact size of the receiver, adjusting the size of the cells and updating the receiver if they don't. If *`flag`* is `false`, then the intercell spacing and cell size remain constant.

See Also

`autosizesCells` (page 884)

setBackgroundColor

Sets the background color for the receiver to *aColor* and redraws the receiver.

```
public void setBackgroundColor(NSColor aColor)
```

Discussion

This color is used to fill the space between cells or the space behind any nonopaque cells. The default background color is NSColor's `controlColor` (page 361).

See Also

`drawsBackground` (page 888)

`setCellBackgroundColor` (page 900)

`backgroundColor` (page 885)

setCellBackgroundColor

Sets the background color for the cells in the receiver to *aColor*.

```
public void setCellBackgroundColor(NSColor aColor)
```

Discussion

This color is used to fill the space behind nonopaque cells. The default cell background color is NSColor's `controlColor` (page 361).

See Also

`drawsCellBackground` (page 888)

`setBackgroundColor` (page 900)

`cellBackgroundColor` (page 885)

setCellSize

Sets the width and height of each of the cells in the receiver to those in *aSize*.

```
public void setCellSize(NSSize aSize)
```

Discussion

This method may change the size of the receiver. Does not redraw the receiver.

See Also

`calcSize` (page 449) (NSControl)

`cellSize` (page 886)

setDelegate

Sets the delegate for messages from the field editor to *anObject*.

```
public void setDelegate(Object anObject)
```

See Also

`textShouldBeginEditing` (page 907)

`textShouldEndEditing` (page 907)

`delegate` (page 887)

setDoubleAction

Makes *aSelector* the action sent to the target of the receiver when the user double-clicks a cell.

```
public void setDoubleAction(NSSelector aSelector)
```

Discussion

A double-click action is always sent after the appropriate single-click action, which is the cell's single-click action, if it has one, or the receiver single-click action, otherwise. If *aSelector* is a non-`NULL` selector, this method also sets the *ignoresMultiClick* flag to `false`; otherwise, it leaves the flag unchanged.

If an NSMatrix has no double-click action set, then by default a double click is treated as a single click.

For the method to have any effect, the receiver's action and target must be set to the class in which the selector is declared. See *Action Messages* for additional information on action messages.

See Also

`sendDoubleAction` (page 899)

`setAction` (page 455) (NSControl)

`setTarget` (page 460) (NSControl)

`doubleAction` (page 888)

setDrawsBackground

Sets whether the receiver draws its background (the space between the cells) to *flag*.

```
public void setDrawsBackground(boolean flag)
```

See Also

`backgroundColor` (page 885)

`setDrawsCellBackground` (page 901)

`drawsBackground` (page 888)

setDrawsCellBackground

Sets whether the receiver draws the background within each of its cells to *flag*.

```
public void setDrawsCellBackground(boolean flag)
```

See Also

`cellBackgroundColor` (page 885)

`setDrawsBackground` (page 901)

`drawsCellBackground` (page 888)

setIntercellSpacing

Sets the vertical and horizontal spacing between cells in the receiver to *aSize*.

```
public void setIntercellSpacing(NSSize aSize)
```

Discussion
By default, both values are 1.0 in the receiver's coordinate system.

See Also
`cellSize` (page 886)
`intercellSpacing` (page 890)

setKeyCell

Sets the cell that will be clicked when the user presses the Space bar to *aCell*.

```
public void setKeyCell(NSCell aCell)
```

See Also
`setTabKeyTraversesCells` (page 904)
`keyCell` (page 891)

setMode

Sets the selection mode of the receiver.

```
public void setMode(int aMode)
```

Discussion
Possible values for *aMode* are listed in "Constants" (page 908).

See Also
`mode` (page 892)

setNewCellClass

Configures the receiver to use instances of *aClass* when creating new cells.

```
public void setNewCellClass(Class aClass)
```

Discussion
aClass should be the `id` of a subclass of NSCell, which can be obtained by sending the `class` message to either the NSCell subclass object or to an instance of that subclass. The default cell class is that set with the class method `setCellClass` (page 447), or NSActionCell if no other default cell class has been specified.

See Also
`addColumn` (page 882)
`addRow` (page 883)
`insertColumn` (page 889)
`insertRow` (page 890)

makeCellAtLocation (page 891)
setPrototype (page 903)
newCellClass (page 892)

setPrototype

Sets the prototype cell that's copied whenever a new cell needs to be created.

```
public void setPrototype(NSCell aCell)
```

See Also

makeCellAtLocation (page 891)
prototype (page 893)

setScrollable

```
public void setScrollable(boolean flag)
```

Discussion

If *flag* is true, makes all the cells scrollable, so the text they contain scrolls to remain in view if the user types past the edge of the cell. If *flag* is false, all cells are made nonscrolling. The prototype cell, if there is one, is also set accordingly.

See Also

prototype (page 893)
setScrollable (page 329) (NSCell)

setSelectionByRect

Sets whether the user can select a rectangle of cells in the receiver by dragging the cursor.

```
public void setSelectionByRect(boolean flag)
```

Discussion

If *flag* is false, selection is on a row-by-row basis. The default is true.

See Also

setSelectionWithAnchor (page 903)

setSelectionWithAnchor

Programmatically selects a range of cells.

```
public void setSelectionWithAnchor(int startPos, int endPos, int anchorPos, boolean
     lit)
```

Discussion

startPos, *endPos*, and *anchorPos* are cell positions, counting from 0 at the upper left cell of the receiver, in row order. For example, the third cell in the top row would be number 2. The *startPos* and *endPos* arguments are used to mark where the user would have pressed the mouse button and released it, respectively.

anchorPos specifies which cell to treat as the last cell the user would have selected. To simulate Shift-dragging (continuous selection) *anchorPos* should be the *endPos* used in the last method call. To simulate Command-dragging (discontinuous selection), *anchorPos* should be the same as this method call's *startPos*. To simulate dragging without a modifier key, deselecting anything that was selected before, call `deselectAllCells` before calling this method.

Finally, *lit* determines whether cells selected by this method should be highlighted.

See Also

`isSelectionByRect` (page 891)

`selectedCells` (page 897)

setStateAtLocation

Sets the state of the cell at *row* and *column* to *value*.

```
public void setStateAtLocation(int value, int row, int column)
```

Discussion

For radio-mode matrices, if *value* is nonzero the specified cell is selected before its state is set to *value*. If *value* is 0 and the receiver is a radio-mode matrix, the currently selected cell is deselected (providing that empty selection is allowed).

This method redraws the affected cell.

See Also

`allowsEmptySelection` (page 884)

`setState` (page 330) (NSCell)

`selectCellAtLocation` (page 896)

setTabKeyTraversesCells

Sets whether pressing the Tab key advances the key cell to the next selectable cell in the receiver.

```
public void setTabKeyTraversesCells(boolean flag)
```

Discussion

If *flag* is `false`, or if there aren't any selectable cells after the current one, when the user presses the Tab key the next view in the window becomes key. Pressing Shift-Tab causes the key cell to advance in the opposite direction (if *flag* is `false`, or if there aren't any selectable cells before the current one, the previous view in the window becomes key).

See Also

`selectKeyViewFollowingView` (page 1852) (NSWindow)

`selectNextKeyView` (page 1852) (NSWindow)

`setKeyCell` (page 902)

`tabKeyTraversesCells` (page 905)

setToolTip

Sets the tooltip for *cell* to be *toolTipString*.

```
public void setToolTip(String toolTipString, NSCell cell)
```

See Also
toolTip (page 907)

setValidateSize

If *flag* is true, then the size information in the receiver is assumed to be correct.

```
public void setValidateSize(boolean flag)
```

Discussion
If *flag* is false, NSControl's calcSize (page 449) method will be invoked before any further drawing is done.

sizeToCells

Changes the width and the height of the receiver's frame so it exactly contains the cells.

```
public void sizeToCells()
```

Discussion
Does not redraw the receiver.

See Also
setFrameSize (page 1777) (NSView)
sizeToFit (page 461) (NSControl)

sortUsingMethod

Sorts the receiver's cells in ascending order as defined by the comparison method *comparator*.

```
public void sortUsingMethod(NSSelector comparator)
```

Discussion
The comparator message is sent to each object in the matrix and has as its single argument another object in the array. The comparison method is used to compare two elements at a time and should return OrderedAscending if the receiver is smaller than the argument, OrderedDescending if the receiver is larger than the argument, and OrderedSame if they are equal.

See Also
sortUsingSelector (NSMutableArray)

tabKeyTraversesCells

Returns whether pressing the Tab key advances the key cell to the next selectable cell in the receiver.

```
public boolean tabKeyTraversesCells()
```

See Also

`keyCell` (page 891)

`setTabKeyTraversesCells` (page 904)

textDidBeginEditing

Invoked when *`notification`* is posted indicating that there's a change in the text after the receiver gains first responder status.

```
public void textDidBeginEditing(NSNotification notification)
```

Discussion

This method's default behavior is to post a `ControlTextDidBeginEditingNotification` (page 466) along with the receiving object to the default notification center. The posted notification's user info contains the contents of notification's user info dictionary, plus an additional key-value pair. The additional key is "`FieldEditor`"; the value for this key is the text object that began editing.

See Also

`textDidChange` (page 906)

`textDidEndEditing` (page 906)

`textShouldEndEditing` (page 907)

textDidChange

Invoked when *`notification`* is posted indicating a key-down event or paste operation that changes the receiver's contents.

```
public void textDidChange(NSNotification notification)
```

Discussion

This method's default behavior is to pass this message on to the selected cell (if the selected cell responds to `textDidChange`) and then to post a `ControlTextDidChangeNotification` (page 467) along with the receiving object to the default notification center. The posted notification's user info contains the contents of notification's user info dictionary, plus an additional key-value pair. The additional key is "`FieldEditor`"; the value for this key is the text object that changed.

See Also

`textDidBeginEditing` (page 906)

`textDidEndEditing` (page 906)

textDidEndEditing

Invoked when *`notification`* is posted indicating that text editing ends.

```
public void textDidEndEditing(NSNotification notification)
```

Discussion

This method's default behavior is to post a `ControlTextDidEndEditingNotification` (page 467) along with the receiving object to the default notification center. The posted notification's user info contains the contents of notification's user info dictionary, plus an additional key-value pair. The additional key is "`FieldEditor`"; the value for this key is the text object that began editing. After posting the notification, `textDidEndEditing` sends an `endEditing` (page 310) message to the selected cell, draws and makes the selected cell key, and then takes the appropriate action based on which key was used to end editing (Return, Tab, or Back-Tab).

See Also

`textDidBeginEditing` (page 906)

`textDidChange` (page 906)

`textShouldEndEditing` (page 907)

textShouldBeginEditing

Invoked to let the NSTextField respond to impending changes to its text.

```
public boolean textShouldBeginEditing(NSText textObject)
```

Discussion

This method's default behavior is to send `controlTextShouldBeginEditing` (page 465) to the receiver's delegate (passing the receiver and *`textObject`* as parameters). The `textShouldBeginEditing` method returns the value obtained from `controlTextShouldBeginEditing` (page 465), unless the delegate doesn't respond to that method, in which case it returns `true`, thereby allowing text editing to proceed.

See Also

`delegate` (page 887)

textShouldEndEditing

Invoked to let the NSTextField respond to impending loss of first-responder status.

```
public boolean textShouldEndEditing(NSText textObject)
```

Discussion

This method's default behavior checks the text field for validity; providing that the field contents are deemed valid, and providing that the delegate responds, `controlTextShouldEndEditing` (page 465) is sent to the receiver's delegate (passing the receiver and *`textObject`* as parameters). If the contents of the text field aren't valid, `textShouldEndEditing` sends the error action to the selected cell's target.

The `textShouldEndEditing` method returns `false` if the text field contains invalid contents; otherwise it returns the value passed back from `controlTextShouldEndEditing` (page 465).

See Also

`delegate` (page 887)

toolTip

Returns the string used as the tooltip for *`cell`*.

```
public String toolTip(NSCell cell)
```

See Also

`setToolTip` (page 905)

Constants

These constants determine how NSCells behave when the NSMatrix is tracking the mouse.

Constant	Description
`TrackMode`	The NSCells are asked to track the mouse with `trackMouse` (page 336) whenever the cursor is inside their bounds. No highlighting is performed.
`HighlightMode`	An NSCell is highlighted before it's asked to track the mouse, then unhighlighted when it's done tracking.
`RadioMode`	Selects no more than one NSCell at a time. Any time an NSCell is selected, the previously selected NSCell is unselected.
`ListMode`	NSCells are highlighted, but don't track the mouse.

NSMenu

Inherits from NSObject
Implements NSCoding

Package: com.apple.cocoa.application
Companion guide Application Menu and Pop-up List Programming Topics for Cocoa

Overview

This class defines an object that manages an application's menus.

Tasks

Constructors

`NSMenu` (page 913)
Creates a new empty menu.

Managing Delegates

`setDelegate` (page 921)
Sets the receiver's delegate.

`delegate` (page 915)
Returns the receiver's delegate.

Managing the Menu Bar

`menuBarVisible` (page 913)
Returns `true` if the menu bar is visible, `false` otherwise.

`setMenuBarVisible` (page 914)
Sets whether the menu bar is visible and selectable by the user.

`menuBarHeight` (page 919)
Returns the menu bar height for the current application's main menu.

Setting Up Menu Commands

`insertItemAtIndex` (page 917)
Inserts the menu item *newItem* in the receiver at location *index*.

`addItem` (page 914)
Adds the menu item *newItem* to the end of the receiver.

`removeItem` (page 920)
Removes *anItem* from the receiver.

`removeItemAtIndex` (page 920)
Removes the menu item at location *index*.

`itemChanged` (page 918)
Invoked when menu item *anObject* is modified visually (for example, its title changes).

Finding Menu Items

`itemWithTag` (page 919)
Returns the first menu item in the receiver with tag *aTag*.

`itemWithTitle` (page 919)
Returns the first menu item in the receiver with title *aString*.

`itemAtIndex` (page 918)
Returns the menu item at location *index* of the receiver.

`numberOfItems` (page 920)
Returns the number of menu items in the receiver, including separator items.

`itemArray` (page 918)
Returns the receiver's menu items.

Finding Indices of Menu Items

`indexOfItem` (page 916)
Returns the index identifying the location of menu item *anObject* in the receiver.

`indexOfItemWithTitle` (page 917)
Returns the index of the first menu item in the receiver that has the title *aTitle*.

`indexOfItemWithTag` (page 916)
Returns the index of the first menu item in the receiver identified by tag *aTag*.

`indexOfItemWithTargetAndAction` (page 917)
Returns the index of the first menu item in the receiver that has target *anObject* and action *actionSelector*.

`indexOfItemWithRepresentedObject` (page 916)
Returns the index of the first menu item in the receiver that has *anObject* as its represented object.

`indexOfItemWithSubmenu` (page 916)
Returns the index of the menu item in the receiver that has submenu *anObject*.

Managing Submenus

`setSubmenuForItem` (page 922)
: Makes *`aMenu`* a submenu controlled by *`anItem`*, automatically setting the action of *`anItem`* to `submenuAction` (page 923).

`submenuAction` (page 923)
: This method is the action method assigned to menu items that open submenus.

`attachedMenu` (page 915)
: Returns the menu currently attached to the receiver or `null` if there's no such object.

`isAttached` (page 918)
: Returns `true` if the receiver is currently attached to another menu, `false` otherwise.

`isTornOff` (page 918)
: Returns `false` if the receiver is offscreen or attached to another menu (or if it's the main menu), `true` otherwise.

`locationForSubmenu` (page 919)

`supermenu` (page 923)
: Returns the receiver's supermenu or `null` if it has none.

`setSupermenu` (page 922)
: Sets the receiver's supermenu to *`supermenu`*.

Enabling and Disabling Menu Items

`autoenablesItems` (page 915)
: Returns whether the receiver automatically enables and disables its menu items based on the "NSMenu.MenuValidation" (page 2011) interface.

`setAutoenablesItems` (page 921)
: Controls whether the receiver automatically enables and disables its menu items based on delegates implementing the NSMenu.MenuValidation interface.

`update` (page 923)
: Enables or disables the receiver's menu items based on the NSMenu.MenuValidation interface and sizes the menu to fit its current menu items if necessary.

Handling Keyboard Equivalents

`performKeyEquivalent` (page 920)

Simulating Mouse Clicks

`performActionForItemAtIndex` (page 920)
: Causes the application to send the action message of the menu item at *`index`* to its target.

Setting the Title

`setTitle` (page 922)
Sets the receiver's title to *aString*.

`title` (page 923)
Returns the receiver's title.

Setting the Representing Object

`setMenuRepresentation` (page 922)
Deprecated. Mac OS X does not use menu representations to draw menus.

`menuRepresentation` (page 920)
Deprecated. Returns `null`.

Updating Menu Layout

`menuChangedMessagesEnabled` (page 919)
Returns `true` if messages are being sent to the application's windows upon each change to the receiver, `false` otherwise.

`setMenuChangedMessagesEnabled` (page 921)
Controls whether the receiver sends messages to the application's windows upon each menu change.

`sizeToFit` (page 922)
Resizes the receiver to exactly fit its items.

Displaying Context-sensitive Help

`popUpContextMenu` (page 914)
Displays *menu* as a context menu over *view* for *event*.

`helpRequested` (page 915)
Overridden by subclasses to implement specialized context-sensitive help behavior by causing the Help manager to display the help associated with the receiver.

Deprecated Methods

`contextMenuRepresentation` (page 915)
Deprecated. Returns `null`.

`setContextMenuRepresentation` (page 921)
Deprecated. Mac OS X does not use menu representations to draw menus.

`tearOffMenuRepresentation` (page 923)
Deprecated. Returns `null`

`setTearOffMenuRepresentation` (page 922)
Deprecated. Mac OS X does not use menu representations to draw menus.

Populating a menu

`menuUpdateItemAtIndex` (page 925) *delegate method*
Called to let you update a menu item before it is displayed.

`menuNeedsUpdate` (page 925) *delegate method*
Called when a menu is about to be displayed at the start of a tracking session so the delegate can modify the menu.

`numberOfItemsInMenu` (page 925) *delegate method*
Called when a menu is about to be displayed at the start of a tracking session so the delegate can specify the number of items in the menu.

Handling key equivalents

`menuHasKeyEquivalent` (page 923) *delegate method*
Called to allow the delegate to return the key down event has a key equivalent.

`menuKeyEquivalentAction` (page 924) *delegate method*
Called to allow the delegate to return the action for a key down event.

`menuKeyEquivalentTarget` (page 924) *delegate method*
Called to allow the delegate to return the target for a key down event.

Constructors

NSMenu

Creates a new empty menu.

```
public NSMenu()
```

Creates a new menu using *aTitle* for its title with autoenabling of menu items turned on.

```
public NSMenu(String aTitle)
```

Static Methods

menuBarVisible

Returns `true` if the menu bar is visible, `false` otherwise.

```
public static boolean menuBarVisible()
```

Availability
Available in Mac OS X v10.2 and later.

See Also

"setMenuBarVisible" (page 914)

popUpContextMenu

Displays *menu* as a context menu over *view* for *event*.

```
public static void popUpContextMenu(NSMenu menu, NSEvent event, NSView view)
```

Displays *menu* as a context menu over *view* for *event* using *font*.

```
public static void popUpContextMenu(NSMenu menu, NSEvent event, NSView view, NSFont
     font)
```

Discussion

If you pass in `null` for the font, the method uses the default font for *menu*.

Availability

Available in Mac OS X v10.3 and later.

setMenuBarVisible

Sets whether the menu bar is visible and selectable by the user.

```
public static void setMenuBarVisible(boolean visible)
```

Availability

Available in Mac OS X v10.2 and later.

See Also

"menuBarVisible" (page 913)

Instance Methods

addItem

Adds the menu item *newItem* to the end of the receiver.

```
public void addItem(NSMenuItem newItem)
```

Discussion

The receiver does not accept the menu item if it already belongs to another menu. After adding the menu item, the receiver updates itself.

Adds a new item with title *aString*, action *aSelector*, and key equivalent *keyEquiv* to the end of the receiver.

```
public NSMenuItem addItem(String aString, NSSelector aSelector, String keyEquiv)
```

Discussion
Returns the new menu item. If you do not want the menu item to have a key equivalent, *keyEquiv* should be an empty string and not `null`.

attachedMenu

Returns the menu currently attached to the receiver or `null` if there's no such object.

```
public NSMenu attachedMenu()
```

autoenablesItems

Returns whether the receiver automatically enables and disables its menu items based on the "NSMenu.MenuValidation" (page 2011) interface.

```
public boolean autoenablesItems()
```

Discussion
By default NSMenus do autoenable their menu items. See the interface specification for more information.

See Also
`setAutoenablesItems` (page 921)

contextMenuRepresentation

Deprecated. Returns `null`.

```
public Object contextMenuRepresentation()
```

delegate

Returns the receiver's delegate.

```
public Object delegate()
```

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

See Also
`setDelegate` (page 921)

helpRequested

Overridden by subclasses to implement specialized context-sensitive help behavior by causing the Help manager to display the help associated with the receiver.

```
public void helpRequested(NSEvent event)
```

Discussion

Never invoke this method directly.

See Also

`showContextHelpForObject` (page 742) (NSHelpManager)

indexOfItem

Returns the index identifying the location of menu item *anObject* in the receiver.

```
public int indexOfItem(NSMenuItem anObject)
```

Discussion

If no such menu item is in the menu, returns –1.

See Also

`insertItemAtIndex` (page 917)

`itemAtIndex` (page 918)

indexOfItemWithRepresentedObject

Returns the index of the first menu item in the receiver that has *anObject* as its represented object.

```
public int indexOfItemWithRepresentedObject(Object anObject)
```

Discussion

If no such menu item is in the menu, returns –1.

See Also

`insertItemAtIndex` (page 917)

`itemAtIndex` (page 918)

indexOfItemWithSubmenu

Returns the index of the menu item in the receiver that has submenu *anObject*.

```
public int indexOfItemWithSubmenu(NSMenu anObject)
```

Discussion

If no such menu item is in the menu, returns –1.

See Also

`insertItemAtIndex` (page 917)

`itemAtIndex` (page 918)

indexOfItemWithTag

Returns the index of the first menu item in the receiver identified by tag *aTag*.

```
public int indexOfItemWithTag(int aTag)
```

Discussion

If no such menu item is in the menu, returns –1.

See Also

`insertItemAtIndex` (page 917)

`itemAtIndex` (page 918)

indexOfItemWithTargetAndAction

Returns the index of the first menu item in the receiver that has target *anObject* and action *actionSelector*.

```
public int indexOfItemWithTargetAndAction(Object anObject, NSSelector actionSelector)
```

Discussion

If *actionSelector* is `NULL`, the first menu item in the receiver that has target *anObject* is returned. If no menu item matching these criteria is in the menu, returns –1.

See Also

`insertItemAtIndex` (page 917)

`itemAtIndex` (page 918)

indexOfItemWithTitle

Returns the index of the first menu item in the receiver that has the title *aTitle*.

```
public int indexOfItemWithTitle(String aTitle)
```

Discussion

If no such menu item is in the menu, returns –1.

See Also

`insertItemAtIndex` (page 917)

`itemAtIndex` (page 918)

insertItemAtIndex

Inserts the menu item *newItem* in the receiver at location *index*.

```
public void insertItemAtIndex(NSMenuItem newItem, int index)
```

Discussion

If the menu item already exists in another menu, it is not inserted. This method posts an `MenuDidAddItemNotification` (page 926), allowing interested observers to update as appropriate. It also causes the menu to update itself. This method is a primitive method. All item addition methods end up calling this method, so this is where you should implement custom behavior on adding new items to a menu in a custom subclass.

Adds a new item at location *index* in the receiver with title *aString*, action *aSelector*, and key equivalent *keyEquiv*.

```
public NSMenuItem insertItemAtIndex(String aString, NSSelector aSelector, String
    keyEquiv, int index)
```

Discussion

Returns the new menu item. If you do not want the menu item to have a key equivalent, *keyEquiv* should be an empty string and not `null`.

See Also

`addItem` (page 914)

`itemAtIndex` (page 918)

`removeItem` (page 920)

isAttached

Returns `true` if the receiver is currently attached to another menu, `false` otherwise.

```
public boolean isAttached()
```

isTornOff

Returns `false` if the receiver is offscreen or attached to another menu (or if it's the main menu), `true` otherwise.

```
public boolean isTornOff()
```

itemArray

Returns the receiver's menu items.

```
public NSArray itemArray()
```

See Also

`numberOfItems` (page 920)

itemAtIndex

Returns the menu item at location *index* of the receiver.

```
public NSMenuItem itemAtIndex(int index)
```

Discussion

It throws an exception if *index* is out of bounds.

itemChanged

Invoked when menu item *anObject* is modified visually (for example, its title changes).

```
public void itemChanged(NSMenuItem anObject)
```

Discussion

It is not called for action, target, represented object, or tag changes. Posts a `MenuDidChangeItemNotification` (page 926).

itemWithTag

Returns the first menu item in the receiver with tag *aTag*.

```
public NSMenuItem itemWithTag(int aTag)
```

itemWithTitle

Returns the first menu item in the receiver with title *aString*.

```
public NSMenuItem itemWithTitle(String aString)
```

locationForSubmenu

```
public NSPoint locationForSubmenu(NSMenu aSubmenu)
```

Discussion

Returns the screen coordinates where *aSubmenu* will be displayed when opened as a submenu of the receiver.

menuBarHeight

Returns the menu bar height for the current application's main menu.

```
public float menuBarHeight()
```

Discussion

Returns 0.0 if the receiver is some other menu. This method supersedes the `"menuBarHeight"` (page 962) class method of the NSMenuView class.

Availability

Available in Mac OS X v10.4 or later.

menuChangedMessagesEnabled

Returns `true` if messages are being sent to the application's windows upon each change to the receiver, `false` otherwise.

```
public boolean menuChangedMessagesEnabled()
```

See Also

`setMenuChangedMessagesEnabled` (page 921)

menuRepresentation

Deprecated. Returns `null`.

```
public Object menuRepresentation()
```

See Also
`setMenuRepresentation` (page 922)

numberOfItems

Returns the number of menu items in the receiver, including separator items.

```
public int numberOfItems()
```

See Also
`itemArray` (page 918)

performActionForItemAtIndex

Causes the application to send the action message of the menu item at *`index`* to its target.

```
public void performActionForItemAtIndex(int index)
```

Discussion
If a target is not specified, the message is sent to the first responder. As a side effect, this method posts `MenuWillSendActionNotification` (page 927) and `MenuDidSendActionNotification` (page 927).

performKeyEquivalent

```
public boolean performKeyEquivalent(NSEvent theEvent)
```

Discussion
Performs the action for the menu item that corresponds to the key equivalent in *`theEvent`*.

removeItem

Removes *`anItem`* from the receiver.

```
public void removeItem(NSMenuItem anItem)
```

removeItemAtIndex

Removes the menu item at location *`index`*.

```
public void removeItemAtIndex(int index)
```

Discussion
Posts a `MenuDidRemoveItemNotification` (page 926).

setAutoenablesItems

Controls whether the receiver automatically enables and disables its menu items based on delegates implementing the NSMenu.MenuValidation interface.

```
public void setAutoenablesItems(boolean flag)
```

Discussion
If *flag* is `true`, menu items are automatically enabled and disabled. If *flag* is `false`, menu items are not automatically enabled or disabled. See the "NSMenu.MenuValidation" (page 2011) interface specification for more information.

See Also
`autoenablesItems` (page 915)

setContextMenuRepresentation

Deprecated. Mac OS X does not use menu representations to draw menus.

```
public void setContextMenuRepresentation(Object menuRep)
```

setDelegate

Sets the receiver's delegate.

```
public void setDelegate(Object anObject)
```

Discussion
You can use the delegate to populate a menu just before it is going to be drawn and to check for key equivalents without creating a menu item.

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

See Also
`delegate` (page 915)

setMenuChangedMessagesEnabled

Controls whether the receiver sends messages to the application's windows upon each menu change.

```
public void setMenuChangedMessagesEnabled(boolean flag)
```

Discussion
To avoid the "flickering" effect of many successive menu changes, invoke this method with *flag* set to `false`, make changes to the menu, and invoke the method again with *flag* set to `true`. This approach has the effect of batching changes and applying them all at once.

See Also
`menuChangedMessagesEnabled` (page 919)

setMenuRepresentation

Deprecated. Mac OS X does not use menu representations to draw menus.

```
public void setMenuRepresentation(Object menuRep)
```

See Also

`menuRepresentation` (page 920)

setSubmenuForItem

Makes *aMenu* a submenu controlled by *anItem*, automatically setting the action of *anItem* to `submenuAction` (page 923).

```
public void setSubmenuForItem(NSMenu aMenu, NSMenuItem anItem)
```

setSupermenu

Sets the receiver's supermenu to *supermenu*.

```
public void setSupermenu(NSMenu supermenu)
```

Discussion

You should never invoke this method directly; it is public so subclassers can add behavior to the default implementation. Subclassers should call the superclass's method as part of their implementation.

See Also

`supermenu` (page 923)

setTearOffMenuRepresentation

Deprecated. Mac OS X does not use menu representations to draw menus.

```
public void setTearOffMenuRepresentation(Object menuRep)
```

setTitle

Sets the receiver's title to *aString*.

```
public void setTitle(String aString)
```

See Also

`title` (page 923)

sizeToFit

Resizes the receiver to exactly fit its items.

```
public void sizeToFit()
```

submenuAction

This method is the action method assigned to menu items that open submenus.

```
public void submenuAction(Object sender)
```

Discussion
Never invoke this method directly.

supermenu

Returns the receiver's supermenu or `null` if it has none.

```
public NSMenu supermenu()
```

tearOffMenuRepresentation

Deprecated. Returns `null`

```
public Object tearOffMenuRepresentation()
```

Discussion
.

title

Returns the receiver's title.

```
public String title()
```

See Also
`setTitle` (page 922)

update

Enables or disables the receiver's menu items based on the NSMenu.MenuValidation interface and sizes the menu to fit its current menu items if necessary.

```
public void update()
```

Discussion
See the "NSMenu.MenuValidation" (page 2011) interface specification for more information.

Delegate Methods

menuHasKeyEquivalent

Called to allow the delegate to return the key down event has a key equivalent.

```
public abstract boolean menuHasKeyEquivalent(NSMenu menu, NSEvent event)
```

Discussion
If there is a valid and enabled menu item that corresponds to this key down even, return `true`. Return `false` if there are no items with that key equivalent or if the item is disabled. If the delegate does not define this method, the menu is populated to find out if any items have a matching key equivalent.

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

See Also
`menuKeyEquivalentTarget` (page 924)
`menuKeyEquivalentAction` (page 924)

menuKeyEquivalentAction

Called to allow the delegate to return the action for a key down event.

```
public abstract NSSelector menuKeyEquivalentAction(NSMenu menu, NSEvent event)
```

Discussion
If there is a valid and enabled menu item that corresponds to this key down even, return the action. Return `null` if there are no items with that key equivalent or if the item is disabled.

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

See Also
`menuHasKeyEquivalent` (page 923)
`menuKeyEquivalentTarget` (page 924)

menuKeyEquivalentTarget

Called to allow the delegate to return the target for a key down event.

```
public abstract Object menuKeyEquivalentTarget(NSMenu menu, NSEvent event)
```

Discussion
If there is a valid and enabled menu item that corresponds to this key down even, return the target. Return `null` if there are no items with that key equivalent or if the item is disabled.

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

See Also
`menuHasKeyEquivalent` (page 923)
`menuKeyEquivalentAction` (page 924)

menuNeedsUpdate

Called when a menu is about to be displayed at the start of a tracking session so the delegate can modify the menu.

```
public abstract void menuNeedsUpdate(NSMenu menu)
```

Discussion

You can change the menu by adding, removing or modifying menu items. Be sure to set the proper enable state for any new menu items. If populating the menu will take a long time, implement `numberOfItemsInMenu` (page 925) and `menuUpdateItemAtIndex` (page 925) instead.

Availability

Available in Mac OS X v10.3 and later.

See Also

`setDelegate` (page 921)

menuUpdateItemAtIndex

Called to let you update a menu item before it is displayed.

```
public abstract boolean menuUpdateItemAtIndex(NSMenu menu, NSMenuItem item, int
    index, boolean shouldCancel)
```

Discussion

If your `numberOfItemsInMenu` (page 925) delegate method returns a positive value, then your `menuUpdateItemAtIndex` method is called for each item in the menu. You can then update the menu title, image, and so forth for the menu item. Return `true` to continue the process. If you return `false`, your `menuUpdateItemAtIndex` is not called again. In that case, it is your responsibility to trim any extra items from the menu.

The *shouldCancel* parameter is set to `true` when your delegate is called if, due to some user action, the menu no longer needs to be displayed before all the menu items have been updated. You can ignore this flag, return `true`, and continue; or you can save your work (to save time the next time your delegate is called) and return `false` to stop the updating.

Availability

Available in Mac OS X v10.3 and later.

numberOfItemsInMenu

Called when a menu is about to be displayed at the start of a tracking session so the delegate can specify the number of items in the menu.

```
public abstract int numberOfItemsInMenu(NSMenu menu)
```

Discussion

If you return a positive value, the menu is resized by either removing or adding items. Newly created items are blank. After the menu is resized, your `menuUpdateItemAtIndex` (page 925) method is called for each item. If you return a negative value, the number of items is left unchanged and `menuUpdateItemAtIndex` is not called. If you can populate the menu quickly, you can implement `menuNeedsUpdate` (page 925) instead of `numberOfItemsInMenu` and `menuUpdateItemAtIndex`.

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

See Also
`setDelegate` (page 921)

Notifications

MenuDidAddItemNotification

Posted after a menu item is added to the menu. The notification object is the NSMenu that just added the new menu item. The *userInfo* dictionary contains the following information:

Key	Value
"NSMenuItemIndex"	The integer index of the menu item that was added.

MenuDidChangeItemNotification

Posted after a menu item in the menu changes appearance. Changes include enabling/disabling, changes in state, and changes to title. The notification object is the NSMenu with the menu item that changed. The *userInfo* dictionary contains the following information:

Key	Value
"NSMenuItemIndex"	The integer index of the menu item that changed.

MenuDidEndTrackingNotification

Posted when menu tracking ends, even if no action is sent. The notification object is the main menu bar (`NSApplication.sharedApplication().mainMenu.()`) or the root menu of a popup button. This notification does not contain a *userInfo* dictionary.

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

MenuDidRemoveItemNotification

Posted after a menu item is removed from the menu. The notification object is the NSMenu that just removed the menu item. The *userInfo* dictionary contains the following information:

Key	Value
"NSMenuItemIndex"	The integer index of the menu item that was removed. Note that this index may no longer be valid and in any event no longer points to the menu item that was removed.

MenuDidSendActionNotification

Posted just after the application dispatches a menu item's action method to the menu item's target. The notification object is the NSMenu containing the chosen menu item. The *userInfo* dictionary contains the following information:

Key	Value
"MenuItem"	The menu item that was chosen.

MenuWillSendActionNotification

Posted just before the application dispatches a menu item's action method to the menu item's target. The notification object is the NSMenu containing the chosen menu item. The *userInfo* dictionary contains the following information:

Key	Value
"MenuItem"	The menu item that was chosen.

NSMenuItem

Inherits from	NSObject
Implements	NSValidatedUserInterfaceItem _NSObsoleteMenuItemProtocol NSCoding
Package:	com.apple.cocoa.application
Companion guide	Application Menu and Pop-up List Programming Topics for Cocoa

Overview

The NSMenuItem class defines objects that are used as command items in menus. Additionally, the NSMenuItem class also includes some private functionality needed to maintain binary compatibility with other components of Cocoa. Because of this fact, you cannot replace the NSMenuItem class with a different class. You may, however, subclass NSMenuItem if necessary.

Interfaces Implemented

NSValidatedUserInterfaceItem

`action` (page 2033)

`tag` (page 2033)

Tasks

Constructors

`NSMenuItem` (page 933)

Creates a new menu item with no action and key equivalent and the title "`NSMenuItem`".

Enabling a Menu Item

`setEnabled` (page 940)

Sets whether the receiver is enabled based on *`flag`*.

`isEnabled` (page 936)
Returns `true` if the receiver is enabled, `false` if not.

Setting the Target and Action

`setTarget` (page 944)
Sets the receiver's target to *`anObject`*.

`target` (page 946)
Returns the receiver's target.

`setAction` (page 939)
Sets the receiver's action method to *`aSelector`*.

`action` (page 935)
Returns the receiver's action method.

Setting the Title

`setTitle` (page 945)
Sets the receiver's title to *`aString`*.

`title` (page 946)
Returns the receiver's title.

`setAttributedTitle` (page 939)
Specifies a custom string for a menu item.

`attributedTitle` (page 935)
Returns the custom title string for a menu item.

Setting the Tag

`setTag` (page 944)
Sets the receiver's tag to *`anInt`*.

`tag` (page 946)
Returns the receiver's tag.

Setting the State

`setState` (page 944)
Sets the state of the receiver to *`itemState`*, which should be one of `NSCell.OffState`, `NSCell.OnState`, or `NSCell.MixedState`.

`state` (page 945)
Returns the state of the receiver, which is `NSCell.OffState` (the default), `NSCell.OnState`, or `NSCell.MixedState`.

Setting the Image

`setImage` (page 940)
Sets the receiver's image to *menuImage*.

`image` (page 935)
Returns the image displayed by the receiver, or `null` if it displays no image.

`setOnStateImage` (page 943)
Sets the image of the receiver that indicates an "on" state.

`onStateImage` (page 938)
Returns the image used to depict the receiver's "on" state, or `null` if the image has not been set.

`setOffStateImage` (page 943)
Sets the image of the receiver that indicates an "off" state.

`offStateImage` (page 938)
Returns the image used to depict the receiver's "off" state, or `null` if the image has not been set.

`setMixedStateImage` (page 942)
Sets the image of the receiver that indicates a "mixed" state, that is, a state neither "on" nor "off."

`mixedStateImage` (page 937)
Returns the image used to depict a "mixed state."

Managing Submenus

`setSubmenu` (page 944)
Sets the submenu of the receiver to *aSubmenu*.

`submenu` (page 946)
Returns the submenu associated with the receiving menu item, or `null` if no submenu is associated with it.

`hasSubmenu` (page 935)
Returns `true` if the receiver has a submenu, `false` if it doesn't.

Getting a Separator Item

`protocolSeparatorItem` (page 934)
Returns a menu item that is used to separate logical groups of menu commands.

`separatorItem` (page 939)
Returns a menu item that is used to separate logical groups of menu commands.

`isSeparatorItem` (page 936)
Returns whether the receiver is a separator item (that is, a menu item used to visually segregate related menu items).

Setting the Owning Menu

`setMenu` (page 942)
Sets the receiver's menu to *aMenu*.

`menu` (page 937)
Returns the menu to which the receiver belongs, or `null` if no menu has been set.

Managing Key Equivalents

`setKeyEquivalent` (page 941)
Sets the receiver's unmodified key equivalent to *`aString`*.

`keyEquivalent` (page 936)
Returns the receiver's unmodified keyboard equivalent, or the empty string if one hasn't been defined.

`setKeyEquivalentModifierMask` (page 941)
Sets the receiver's keyboard equivalent modifiers (indicating modifiers such as the Shift or Option keys) to those in *`mask`*.

`keyEquivalentModifierMask` (page 937)
Returns the receiver's keyboard equivalent modifier mask.

Managing Mnemonics

`setMnemonicLocation` (page 942)
Deprecated. Sets the character of the menu item title at location that is to be underlined.

`mnemonicLocation` (page 937)
Deprecated. Returns the position of the underlined character in the menu item title used as a mnemonic.

`setTitleWithMnemonic` (page 945)
Deprecated. Sets the title of a menu item with a character denoting an access key.

`mnemonic` (page 937)
Deprecated. Returns the character in the menu item title that appears underlined for use as a mnemonic.

Managing User Key Equivalents

`setUsesUserKeyEquivalents` (page 934)
If *`flag`* is `true`, menu items conform to user preferences for key equivalents; otherwise, the key equivalents originally assigned to the menu items are used.

`usesUserKeyEquivalents` (page 934)
Returns `true` if menu items conform to user preferences for key equivalents; otherwise, returns `false`.

`userKeyEquivalent` (page 947)
Returns the user-assigned key equivalent for the receiver.

`userKeyEquivalentModifierMask` (page 947)
Returns the modifier mask for the receiver's user-assigned key equivalent.

Managing Alternates

`setAlternate` (page 939)
: Marks the receiver as an alternate to the previous menu item.

`isAlternate` (page 936)
: Returns whether the receiver is an alternate to the previous menu item.

Managing Indentation Levels

`setIndentationLevel` (page 940)
: Sets the menu item indentation level for the receiver.

`indentationLevel` (page 935)
: Returns the menu item indentation level for the receiver.

Managing Tool Tips

`setToolTip` (page 945)
: Sets a help tag for a menu item.

`toolTip` (page 946)
: Returns the help tag for a menu item.

Representing an Object

`setRepresentedObject` (page 943)
: Sets the object represented by the receiver to *anObject*.

`representedObject` (page 938)
: Returns the object that the receiving menu item represents.

Constructors

NSMenuItem

Creates a new menu item with no action and key equivalent and the title "NSMenuItem".

```
public NSMenuItem()
```

Creates a new NSMenuItem.

```
public NSMenuItem(String itemName, NSSelector action, String charCode)
```

Discussion
The arguments *itemName* and *charCode* must not be `null` (if there is no title or key equivalent, specify an empty string). The *action* argument must be a valid selector or `null`. For instances of the `NSMenuItem` class, the default initial state is `NSCell.OffState`, the default on-state image is a checkmark, and the default mixed-state image is a dash.

Static Methods

protocolSeparatorItem

Returns a menu item that is used to separate logical groups of menu commands.

```
public static _NSObsoleteMenuItemProtocol protocolSeparatorItem()
```

Discussion
This menu item is disabled. The default separator item is blank space.

See Also
`isSeparatorItem` (page 936)
`setEnabled` (page 940)

setUsesUserKeyEquivalents

If *flag* is `true`, menu items conform to user preferences for key equivalents; otherwise, the key equivalents originally assigned to the menu items are used.

```
public static void setUsesUserKeyEquivalents(boolean flag)
```

See Also
`usesUserKeyEquivalents` (page 934)
`userKeyEquivalent` (page 947)

usesUserKeyEquivalents

Returns `true` if menu items conform to user preferences for key equivalents; otherwise, returns `false`.

```
public static boolean usesUserKeyEquivalents()
```

See Also
`setUsesUserKeyEquivalents` (page 934)
`userKeyEquivalent` (page 947)

Instance Methods

action

Returns the receiver's action method.

```
public NSSelector action()
```

See Also
`target` (page 946)
`setAction` (page 939)

attributedTitle

Returns the custom title string for a menu item.

```
public NSAttributedString attributedTitle()
```

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

See Also
`setAttributedTitle` (page 939)
`title` (page 946)

hasSubmenu

Returns `true` if the receiver has a submenu, `false` if it doesn't.

```
public boolean hasSubmenu()
```

See Also
`setSubmenuForItem` (page 922) (NSMenu)

image

Returns the image displayed by the receiver, or `null` if it displays no image.

```
public NSImage image()
```

See Also
`setImage` (page 940)

indentationLevel

Returns the menu item indentation level for the receiver.

```
public int indentationLevel()
```

Discussion

The return value is from 0 to 15. The default indentation level is 0.

Availability

Available in Mac OS X v10.3 and later.

See Also

`setIndentationLevel` (page 940)

isAlternate

Returns whether the receiver is an alternate to the previous menu item.

```
public boolean isAlternate()
```

Availability

Available in Mac OS X v10.3 and later.

See Also

`setAlternate` (page 939)

isEnabled

Returns `true` if the receiver is enabled, `false` if not.

```
public boolean isEnabled()
```

See Also

`setEnabled` (page 940)

isSeparatorItem

Returns whether the receiver is a separator item (that is, a menu item used to visually segregate related menu items).

```
public boolean isSeparatorItem()
```

See Also

`separatorItem` (page 939)

keyEquivalent

Returns the receiver's unmodified keyboard equivalent, or the empty string if one hasn't been defined.

```
public String keyEquivalent()
```

Discussion

Use `keyEquivalentModifierMask` (page 937) to determine the modifier mask for the key equivalent.

See Also

`userKeyEquivalent` (page 947)

mnemonic (page 937)
setKeyEquivalent (page 941)

keyEquivalentModifierMask

Returns the receiver's keyboard equivalent modifier mask.

```
public int keyEquivalentModifierMask()
```

See Also
setKeyEquivalentModifierMask (page 941)

menu

Returns the menu to which the receiver belongs, or `null` if no menu has been set.

```
public NSMenu menu()
```

See Also
setMenu (page 942)

mixedStateImage

Returns the image used to depict a "mixed state."

```
public NSImage mixedStateImage()
```

Discussion
A mixed state is useful for indicating "off" and "on" attribute values in a group of selected objects, such as a selection of text containing boldface and plain (nonboldface) words. By default this is a horizontal line.

See Also
setMixedStateImage (page 942)

mnemonic

Deprecated. Returns the character in the menu item title that appears underlined for use as a mnemonic.

```
public String mnemonic()
```

Discussion
If there is no mnemonic character, returns an empty string.

See Also
setTitleWithMnemonic (page 945)

mnemonicLocation

Deprecated. Returns the position of the underlined character in the menu item title used as a mnemonic.

```
public int mnemonicLocation()
```

Discussion

The position is the 0 based index of that character in the title string. If the receiver has no mnemonic character, returns `NSArray.NotFound`.

See Also

`setMnemonicLocation` (page 942)

offStateImage

Returns the image used to depict the receiver's "off" state, or `null` if the image has not been set.

```
public NSImage offStateImage()
```

Discussion

By default there is no image.

See Also

`setOffStateImage` (page 943)

onStateImage

Returns the image used to depict the receiver's "on" state, or `null` if the image has not been set.

```
public NSImage onStateImage()
```

Discussion

By default this image is a checkmark.

See Also

`setOnStateImage` (page 943)

representedObject

Returns the object that the receiving menu item represents.

```
public Object representedObject()
```

Discussion

For example, you might have a menu list the names of views that are swapped into the same panel. The represented objects would be the appropriate `NSView` objects. The user would then be able to switch back and forth between the different views that are displayed by selecting the various menu items.

See Also

`tag` (page 946)

`setRepresentedObject` (page 943)

separatorItem

Returns a menu item that is used to separate logical groups of menu commands.

```
public NSMenuItem separatorItem()
```

Discussion

This menu item is disabled. The default separator item is blank space.

See Also

isSeparatorItem (page 936)

setEnabled (page 940)

setAction

Sets the receiver's action method to *aSelector*.

```
public void setAction(NSSelector aSelector)
```

Discussion

See *Action Messages* for additional information on action messages.

See Also

setTarget (page 944)

action (page 935)

setAlternate

Marks the receiver as an alternate to the previous menu item.

```
public void setAlternate(boolean isAlternate)
```

Discussion

If the receiver has the same key equivalent as the previous item, but has different key equivalent modifiers, the items are folded into a single visible item and the appropriate item shows while tracking the menu. The menu items may also have no key equivalent as long as the key equivalent modifiers are different.

If there are two or more items with no key equivalent but different modifiers, then the only way to get access to the alternate items is with the mouse. If you mark items as alternates but their key equivalents don't match, they might be displayed as separate items. Marking the first item as an alternate has no effect.

The *isAlternate* value is archived.

Availability

Available in Mac OS X v10.3 and later.

See Also

isAlternate (page 936)

setAttributedTitle

Specifies a custom string for a menu item.

```
public void setAttributedTitle(NSAttributedString string)
```

Discussion
You can use this method to add styled text and embedded images to menu item strings. If you do not set a text color for the attributed string, it is black when not selected, white when selected, and gray when disabled. Colored text remains unchanged when selected.

When you call this method to set the menu title to an attributed string, the `setTitle` (page 945) method is also called to set the menu title with a plain string. If you clear the attributed title, the plain title remains unchanged. To clear the attributed title, set the attributed string to either `null` or an empty attributed string.

The attributed string is not archived in the old nib format.

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

See Also
`attributedTitle` (page 935)
`setTitle` (page 945)

setEnabled

Sets whether the receiver is enabled based on *flag*.

```
public void setEnabled(boolean flag)
```

Discussion
If a menu item is disabled, its keyboard equivalent is also disabled. See the "NSMenu.MenuValidation" (page 2011) interface specification for cautions regarding this method.

See Also
`isEnabled` (page 936)

setImage

Sets the receiver's image to *menuImage*.

```
public void setImage(NSImage menuImage)
```

Discussion
If *menuImage* is `null`, the current image (if any) is removed. This image is not affected by changes in menu-item state.

See Also
`image` (page 935)

setIndentationLevel

Sets the menu item indentation level for the receiver.

```
public void setIndentationLevel(int indentationLevel)
```

Discussion

The value for *indentationLevel* may be from 0 to 15. If *indentationLevel* is greater than 15, the value is pinned to the maximum. If *indentationLevel* is less than 0, an exception is thrown. The default indentation level is 0.

The *indentationLevel* value is archived.

Availability

Available in Mac OS X v10.3 and later.

See Also

indentationLevel (page 935)

setKeyEquivalent

Sets the receiver's unmodified key equivalent to *aString*.

```
public void setKeyEquivalent(String aString)
```

Discussion

If you want to remove the key equivalent from a menu item, pass an empty string (" ") for *aString* (never pass null). Use setKeyEquivalentModifierMask (page 941) to set the appropriate mask for the modifier keys for the key equivalent.

If you want to specify the Backspace key as the key equivalent for a menu item, use a single character string with NSBackspaceCharacter (defined in NSText.h as 0x08) and for the Forward Delete key, use NSDeleteCharacter (defined in NSText.h as 0x7F). Note that these are not the same characters you get from an NSEvent key-down event when pressing those keys.

See Also

setMnemonicLocation (page 942)

keyEquivalent (page 936)

setKeyEquivalentModifierMask

Sets the receiver's keyboard equivalent modifiers (indicating modifiers such as the Shift or Option keys) to those in *mask*.

```
public void setKeyEquivalentModifierMask(int mask)
```

Discussion

mask is an integer bit field containing any of these modifier key masks, combined using the C bitwise OR operator:

```
NSEvent.ShiftKeyMask
NSEvent.AlternateKeyMask
NSEvent.CommandKeyMask
NSEvent.ControlKeyMask
```

You should always set NSEvent.CommandKeyMask in *mask*.

`NSEvent.ShiftKeyMask` is relevant only for function keys—that is, for key events whose modifier flags include `NSEvent.FunctionKeyMask`. For all other key events `NSEvent.ShiftKeyMask` is ignored, and characters typed while the Shift key is pressed are interpreted as the shifted versions of those characters; for example, Command-Shift-c is interpreted as Command-C.

See the NSEvent (page 603) class specification for more information about modifier mask values.

See Also
`keyEquivalentModifierMask` (page 937)

setMenu

Sets the receiver's menu to *aMenu*.

```
public void setMenu(NSMenu aMenu)
```

Discussion
This method is invoked by the owning NSMenu when the receiver is added or removed. You shouldn't have to invoke this method in your own code, although it can be overridden to provide specialized behavior.

See Also
`menu` (page 937)

setMixedStateImage

Sets the image of the receiver that indicates a "mixed" state, that is, a state neither "on" nor "off."

```
public void setMixedStateImage(NSImage itemImage)
```

Discussion
If *itemImage* is `null`, any current mixed-state image is removed. Changing state images is currently unsupported in Mac OS X.

See Also
`mixedStateImage` (page 937)
`setOffStateImage` (page 943)
`setOnStateImage` (page 943)
`setState` (page 944)

setMnemonicLocation

Deprecated. Sets the character of the menu item title at location that is to be underlined.

```
public void setMnemonicLocation(int location)
```

Discussion
location must be from 0 to 254. This character identifies the access key by which users can access the menu item.

See Also
`mnemonicLocation` (page 937)

setOffStateImage

Sets the image of the receiver that indicates an "off" state.

```
public void setOffStateImage(NSImage itemImage)
```

Discussion

If `itemImage` is `null`, any current off-state image is removed. Changing state images is currently unsupported in Mac OS X.

See Also

`offStateImage` (page 938)

`setMixedStateImage` (page 942)

`setOffStateImage` (page 943)

`setState` (page 944)

setOnStateImage

Sets the image of the receiver that indicates an "on" state.

```
public void setOnStateImage(NSImage itemImage)
```

Discussion

If `itemImage` is `null`, any current on-state image is removed. Changing state images is currently unsupported in Mac OS X.

See Also

`onStateImage` (page 938)

`setMixedStateImage` (page 942)

`setOffStateImage` (page 943)

`setState` (page 944)

setRepresentedObject

Sets the object represented by the receiver to `anObject`.

```
public void setRepresentedObject(Object anObject)
```

Discussion

By setting a represented object for a menu item, you make an association between the menu item and that object. The represented object functions as a more specific form of tag that allows you to associate any object, not just an int, with the items in a menu.

For example, an NSView object might be associated with a menu item—when the user chooses the menu item, the represented object is fetched and displayed in a panel. Several menu items might control the display of multiple views in the same panel.

See Also

`setTag` (page 944)

`representedObject` (page 938)

setState

Sets the state of the receiver to `itemState`, which should be one of `NSCell.OffState`, `NSCell.OnState`, or `NSCell.MixedState`.

```
public void setState(int itemState)
```

Discussion

The image associated with the new state is displayed to the left of the menu item.

See Also

`state` (page 945)

`setMixedStateImage` (page 942)

`setOffStateImage` (page 943)

`setOnStateImage` (page 943)

setSubmenu

Sets the submenu of the receiver to `aSubmenu`.

```
public void setSubmenu(NSMenu aSubmenu)
```

Discussion

The default implementation of the `NSMenuItem` class throws an exception if `aSubmenu` already has a supermenu.

See Also

`submenu` (page 946)

`hasSubmenu` (page 935)

setTag

Sets the receiver's tag to `anInt`.

```
public void setTag(int anInt)
```

See Also

`setRepresentedObject` (page 943)

`tag` (page 946)

setTarget

Sets the receiver's target to `anObject`.

```
public void setTarget(Object anObject)
```

See Also

`setAction` (page 939)

`target` (page 946)

setTitle

Sets the receiver's title to *aString*.

```
public void setTitle(String aString)
```

See Also

title (page 946)

setTitleWithMnemonic

Deprecated. Sets the title of a menu item with a character denoting an access key.

```
public void setTitleWithMnemonic(String aString)
```

Discussion

Use an ampersand character to mark the character (the one following the ampersand) to be designated.

See Also

mnemonic (page 937)

setMnemonicLocation (page 942)

setToolTip

Sets a help tag for a menu item.

```
public void setToolTip(String toolTip)
```

Discussion

You can call this method for any menu item, including items in the main menu bar.

This string is not archived in the old nib format.

Availability

Available in Mac OS X v10.3 and later.

See Also

toolTip (page 946)

state

Returns the state of the receiver, which is NSCell.OffState (the default), NSCell.OnState, or NSCell.MixedState.

```
public int state()
```

See Also

setState (page 944)

submenu

Returns the submenu associated with the receiving menu item, or `null` if no submenu is associated with it.

```
public NSMenu submenu()
```

Discussion

If the receiver responds `true` to `hasSubmenu` (page 935), the submenu is returned.

See Also

`hasSubmenu` (page 935)

`setSubmenu` (page 944)

tag

Returns the receiver's tag.

```
public int tag()
```

See Also

`representedObject` (page 938)

`setTag` (page 944)

target

Returns the receiver's target.

```
public Object target()
```

See Also

`action` (page 935)

`setTarget` (page 944)

title

Returns the receiver's title.

```
public String title()
```

See Also

`setTitle` (page 945)

toolTip

Returns the help tag for a menu item.

```
public String toolTip()
```

Availability

Available in Mac OS X v10.3 and later.

See Also

`setToolTip` (page 945)

userKeyEquivalent

Returns the user-assigned key equivalent for the receiver.

```
public String userKeyEquivalent()
```

See Also

`keyEquivalent` (page 936)

userKeyEquivalentModifierMask

Returns the modifier mask for the receiver’s user-assigned key equivalent.

```
public int userKeyEquivalentModifierMask()
```

NSMenuItemCell

Inherits from	NSButtonCell : NSActionCell : NSCell : NSObject
Implements	NSCoding (NSCell)
Package:	com.apple.cocoa.application
Companion guide	Application Menu and Pop-up List Programming Topics for Cocoa

Overview

Note: NSMenuItemCell and NSMenuView are deprecated and are no longer used to draw menus. Using them will not affect the appearance of your menus.

NSMenuItemCell is a class that handles the measurement and display of a single menu item in its encompassing frame. Instances of NSMenuItemCell work in conjunction with an NSMenuView object to control the overall appearance of the menu.

Tasks

Constructors

`NSMenuItemCell` (page 951)
    This class is deprecated and is no longer used to draw menus.

Getting and Setting Menu Item Attributes

`isHighlighted` (page 954)
    Returns `true` if the receiver currently draws its menu item with a highlighted appearance

`setHighlighted` (page 956)
    Sets the highlight state for the receiver to *`flag`*.

`menuItem` (page 955)
    Returns the NSMenuItem associated with the receiver.

`setMenuItem` (page 956)
    Sets the NSMenuItem for the receiver to *`item`*.

`menuView` (page 955)
Returns the menu view associated with the receiver.

`setMenuView` (page 956)
Sets the menu view for the receiver *`menuView`*.

Calculating Menu Item Sizes

`calcSize` (page 951)
Calculates the minimum required width and height of the receiver's menu item.

`needsSizing` (page 955)
Returns `true` if the size of the menu item needs to be calculated; otherwise returns `false`.

`setNeedsSizing` (page 957)
Sets a flag that indicates whether or not the menu item must be resized.

`imageWidth` (page 954)
Returns the width of the image associated with a menu item.

`titleWidth` (page 958)
Returns the width of the menu item text.

`keyEquivalentWidth` (page 955)
Returns the width of the key equivalent associated with the menu item.

`stateImageWidth` (page 957)
Returns the width of the image used to indicate the state of the menu item.

Getting the Menu Item's Drawing Rectangle

`imageRectForBounds` (page 953)
Returns the rectangle into which the menu item's image should be drawn.

`keyEquivalentRectForBounds` (page 954)
Returns the rectangle into which the menu item's key equivalent should be drawn.

`stateImageRectForBounds` (page 957)
Returns the rectangle into which the menu item's state image should be drawn.

`titleRectForBounds` (page 958)
Returns the rectangle into which the menu item's title should be drawn.

Drawing the Menu Item

`drawBorderAndBackgroundWithFrameInView` (page 952)
Draws the borders and background associated with the receiver's menu item (if any).

`drawImageWithFrameInView` (page 952)
Draws the image associated with the menu item.

`drawKeyEquivalentWithFrameInView` (page 952)
Draws the key equivalent associated with the menu item.

`drawSeparatorItemWithFrameInView` (page 953)
Draws a menu item separator.

`drawStateImageWithFrameInView` (page 953)
    Draws the state image associated with the menu item.

`drawTitleWithFrameInView` (page 953)
    Draws the title associated with the menu item.

`needsDisplay` (page 955)
    Returns `true` if the menu item needs to be displayed; otherwise returns `false`.

`setNeedsDisplay` (page 956)
    Sets whether the menu item needs to be drawn to *`flag`*.

Assigning a Tag

`tag` (page 958)
    Returns the tag of the selected menu item, or 0 if no item is selected.

Constructors

NSMenuItemCell

This class is deprecated and is no longer used to draw menus.

```
public NSMenuItemCell()
```

Discussion
Do not use this constructor.

This class is deprecated and is no longer used to draw menus.

```
public NSMenuItemCell(String aString)
```

Discussion
Do not use this constructor.

This class is deprecated and is no longer used to draw menus.

```
public NSMenuItemCell(NSImage anImage)
```

Discussion
Do not use this constructor.

Instance Methods

calcSize

Calculates the minimum required width and height of the receiver's menu item.

```
public void calcSize()
```

Discussion
The calculated values are cached for future use. This method also calculates the sizes of individual components of the cell's menu item and caches those values.

This method is invoked automatically when necessary. You should not need to invoke it directly.

See Also
`needsSizing` (page 955)

drawBorderAndBackgroundWithFrameInView

Draws the borders and background associated with the receiver's menu item (if any).

```
public void drawBorderAndBackgroundWithFrameInView(NSRect cellFrame, NSView
    controlView)
```

Discussion
This method invokes `imageRectForBounds` (page 953), passing it *`cellFrame`*, to calculate the rectangle in which to draw the image. The *`controlView`* parameter specifies the view that contains this cell. The cell invokes this method before invoking the methods to draw the other menu item components.

See Also
`drawWithFrameInView` (page 310) (NSCell)

drawImageWithFrameInView

Draws the image associated with the menu item.

```
public void drawImageWithFrameInView(NSRect cellFrame, NSView controlView)
```

Discussion
This method invokes `imageRectForBounds` (page 953), passing it *`cellFrame`*, to calculate the rectangle in which to draw the image. The *`controlView`* parameter specifies the view that contains this cell.

This method is invoked by the cell's `drawWithFrame` method. You should not need to invoke it directly. Subclasses may override this method to control the drawing of the image.

drawKeyEquivalentWithFrameInView

Draws the key equivalent associated with the menu item.

```
public void drawKeyEquivalentWithFrameInView(NSRect cellFrame, NSView controlView)
```

Discussion
This method invokes `keyEquivalentRectForBounds` (page 954), passing it *`cellFrame`*, to calculate the rectangle in which to draw the key equivalent. The *`controlView`* parameter specifies the view that contains this cell.

This method is invoked by the cell's `drawWithFrame` method. You should not need to invoke it directly. Subclasses may override this method to control the drawing of the key equivalent.

drawSeparatorItemWithFrameInView

Draws a menu item separator.

```
public void drawSeparatorItemWithFrameInView(NSRect cellFrame, NSView controlView)
```

Discussion

This method uses the `cellFrame` parameter to calculate the rectangle in which to draw the menu item separator. This method uses the `controlView` to determine whether the separator item should be drawn normally or flipped.

You should not need to invoke this method directly. Subclasses may override this method to control the drawing of the separator.

See Also

`drawKeyEquivalentWithFrameInView` (page 952)

`drawTitleWithFrameInView` (page 953)

`isFlipped` (page 1756) (NSView)

drawStateImageWithFrameInView

Draws the state image associated with the menu item.

```
public void drawStateImageWithFrameInView(NSRect cellFrame, NSView controlView)
```

Discussion

This method invokes `stateImageRectForBounds` (page 957), passing it `cellFrame`, to calculate the rectangle in which to draw the state image. The `controlView` parameter specifies the view that contains this cell.

This method is invoked by the cell's `drawWithFrame` method. You should not need to invoke it directly. Subclasses may override this method to control the drawing of the state image.

drawTitleWithFrameInView

Draws the title associated with the menu item.

```
public void drawTitleWithFrameInView(NSRect cellFrame, NSView controlView)
```

Discussion

This method invokes `titleRectForBounds` (page 958), passing it `cellFrame`, to calculate the rectangle in which to draw the title. The `controlView` parameter specifies the view that contains this cell.

This method is invoked by the cell's `drawWithFrame` method. You should not need to invoke it directly. Subclasses may override this method to control the drawing of the title.

imageRectForBounds

Returns the rectangle into which the menu item's image should be drawn.

```
public NSRect imageRectForBounds(NSRect cellFrame)
```

Discussion

The returned rectangle is based on *cellFrame* but encompasses only the area to be occupied by the image.

See Also

`stateImageRectForBounds` (page 957)

`imageRectForBounds` (page 953)

`titleRectForBounds` (page 958)

`keyEquivalentRectForBounds` (page 954)

imageWidth

Returns the width of the image associated with a menu item.

```
public float imageWidth()
```

Discussion

You can associate an image with a menu item using NSMenuItem's `setImage` (page 1928) method.

See Also

`stateImageWidth` (page 957)

`calcSize` (page 951)

`needsSizing` (page 955)

isHighlighted

Returns `true` if the receiver currently draws its menu item with a highlighted appearance

```
public boolean isHighlighted()
```

Discussion

.

See Also

`setHighlighted` (page 956)

keyEquivalentRectForBounds

Returns the rectangle into which the menu item's key equivalent should be drawn.

```
public NSRect keyEquivalentRectForBounds(NSRect cellFrame)
```

Discussion

The returned rectangle is based on *cellFrame* but encompasses only the area to be occupied by the key equivalent.

See Also

`keyEquivalent` (page 1924) (NSMenuItem)

`stateImageRectForBounds` (page 957)

`imageRectForBounds` (page 953)

`titleRectForBounds` (page 958)

keyEquivalentRectForBounds (page 954)

keyEquivalentWidth

Returns the width of the key equivalent associated with the menu item.

```
public float keyEquivalentWidth()
```

Discussion
You can associate a key equivalent with a menu item using NSMenuItem's setKeyEquivalent (page 1928) method.

See Also
calcSize (page 951)
needsSizing (page 955)

menuItem

Returns the NSMenuItem associated with the receiver.

```
public NSMenuItem menuItem()
```

See Also
setMenuItem (page 956)

menuView

Returns the menu view associated with the receiver.

```
public NSMenuView menuView()
```

See Also
setMenuView (page 956)

needsDisplay

Returns `true` if the menu item needs to be displayed; otherwise returns `false`.

```
public boolean needsDisplay()
```

See Also
setNeedsDisplay (page 956)

needsSizing

Returns `true` if the size of the menu item needs to be calculated; otherwise returns `false`.

```
public boolean needsSizing()
```

See Also

`setNeedsSizing` (page 957)

`calcSize` (page 951)

setHighlighted

Sets the highlight state for the receiver to *`flag`*.

```
public void setHighlighted(boolean flag)
```

Discussion

You should not need to call this method directly. It is invoked by the NSMenuView's `setHighlightedItemIndex` (page 969) method to provide user feedback during menu tracking or when the user selects a menu item (either by clicking or using a key equivalent).

See Also

`isHighlighted` (page 954)

`setHighlightedItemIndex` (page 969) (NSMenuView)

setMenuItem

Sets the NSMenuItem for the receiver to *`item`*.

```
public void setMenuItem(NSMenuItem item)
```

See Also

`menuItem` (page 955)

setMenuView

Sets the menu view for the receiver *`menuView`*.

```
public void setMenuView(NSMenuView menuView)
```

See Also

`menuView` (page 955)

setNeedsDisplay

Sets whether the menu item needs to be drawn to *`flag`*.

```
public void setNeedsDisplay(boolean flag)
```

See Also

`needsDisplay` (page 955)

setNeedsSizing

Sets a flag that indicates whether or not the menu item must be resized.

```
public void setNeedsSizing(boolean flag)
```

Discussion

If *flag* is true, the next attempt to obtain any size-related information from this menu item cell invokes the calcSize (page 951) method to recalculate the information. If *flag* is false, the next attempt to obtain size-related information returns the currently cached values.

Subclasses that drastically change the way a menu item is drawn may need to invoke this method to recalculate the menu item information. Other parts of your application should not need to invoke this method directly. The cell invokes this method as necessary when the content of its menu item changes.

See Also

needsSizing (page 955)

stateImageRectForBounds

Returns the rectangle into which the menu item's state image should be drawn.

```
public NSRect stateImageRectForBounds(NSRect cellFrame)
```

Discussion

The returned rectangle is based on *cellFrame* but encompasses only the area to be occupied by the menu item's state image.

See Also

stateImageRectForBounds (page 957)

imageRectForBounds (page 953)

titleRectForBounds (page 958)

keyEquivalentRectForBounds (page 954)

stateImageWidth

Returns the width of the image used to indicate the state of the menu item.

```
public float stateImageWidth()
```

Discussion

If the menu item has multiple images associated with it (to indicate any of the available states: on, off, or mixed), this method returns the width of the largest image. You can set the state images for a menu item using NSMenuItem's setOnStateImage (page 1930), setOffStateImage (page 1930), and setMixedStateImage (page 1929) methods.

To change the state of the cell's menu item, use NSMenuItem's setState (page 1931) method.

See Also

calcSize (page 951)

needsSizing (page 955)

setState (page 1931) (NSMenuItem)

tag

Returns the tag of the selected menu item, or 0 if no item is selected.

```
public int tag()
```

Discussion

Setting the tag value of an NSMenuItemCell with `setTag` (page 51) does nothing.

See Also

`setTag` (page 51) (NSActionCell)

titleRectForBounds

Returns the rectangle into which the menu item's title should be drawn.

```
public NSRect titleRectForBounds(NSRect cellFrame)
```

Discussion

The returned rectangle is based on *`cellFrame`* but encompasses only the area to be occupied by the text of the title.

See Also

`stateImageRectForBounds` (page 957)

`imageRectForBounds` (page 953)

`titleRectForBounds` (page 958)

`keyEquivalentRectForBounds` (page 954)

titleWidth

Returns the width of the menu item text.

```
public float titleWidth()
```

Discussion

To set the menu item text, use NSMenuItem's `setTitle` (page 1932) method.

See Also

`calcSize` (page 951)

`needsSizing` (page 955)

NSMenuView

Inherits from	NSView : NSResponder : NSObject
Implements	NSCoding (NSResponder)
Package:	com.apple.cocoa.application
Companion guide	Application Menu and Pop-up List Programming Topics for Cocoa

Overview

Note: NSMenuItemCell and NSMenuView are deprecated and are no longer used to draw menus. Calling their methods will not affect the appearance of your menus.

The NSMenuView class handles the display of menus on the user's screen. A menu view displays its menu either horizontally or vertically and allows the user to interact with the items of that menu, either to navigate through hierarchical menus or to select a particular item.

Tasks

Constructors

`NSMenuView` (page 962)
    This class is deprecated and is no longer used to draw menus.

Getting and Setting Menu View Attributes

`menuBarHeight` (page 962)
    Returns the height of the menu bar.

`setMenu` (page 970)
    Sets the menu to be displayed in this view to *`menu`*.

`menu` (page 968)
    Returns the NSMenu associated with this menu view.

`setHorizontal` (page 970)
    Sets the orientation of the menu.

`isHorizontal` (page 966)
: Returns `true` if the menu is displayed horizontally; such as for a menu bar, otherwise returns `false`.

`setFont` (page 969)
: Sets the default font to use when drawing the menu text.

`font` (page 964)
: Returns the default font used to draw the menu text.

`setHighlightedItemIndex` (page 969)
: Highlights the menu item at *`index`*.

`highlightedItemIndex` (page 964)
: Returns the index of the currently highlighted menu item, or –1 if no menu item in the menu is highlighted.

`setMenuItemCellForItemAtIndex` (page 970)
: Replaces the menu item cell at *`index`* with *`cell`*.

`menuItemCellForItemAtIndex` (page 968)
: Returns the menu item cell at the specified *`index`*.

`attachedMenuView` (page 963)
: Returns the NSMenuView of this object's attached menu view.

`attachedMenu` (page 962)
: Returns the NSMenu object associated with this object's attached menu view.

`isAttached` (page 965)
: Returns `true` if this menu is currently attached to its parent menu.

`isTornOff` (page 966)
: Deprecated. Tear-off menus are not supported in Mac OS X.

`horizontalEdgePadding` (page 964)
: Returns the amount of horizontal space used for padding menu item components.

`setHorizontalEdgePadding` (page 970)
: Sets the horizontal padding for menu item components to *`pad`*.

Notification Methods

`itemChanged` (page 966)
: Marks the menu view as needing to be resized so changes in size resulting from a change in the menu will be tracked.

`itemAdded` (page 966)
: Creates a new menu item cell for the newly created item and marks the menu view as needing to be resized.

`itemRemoved` (page 967)
: Removes the removed item's menu item cell and marks the menu view as needing to be resized.

Working with Submenus

`detachSubmenu` (page 963)
: Detaches the window associated with the currently visible submenu and removes any menu item highlights.

`attachSubmenuForItemAtIndex` (page 963)
: Attaches the submenu associated with the menu item at *index*.

Calculating Menu Geometry

`update` (page 973)
: Asks the associated NSMenu to update itself.

`setNeedsSizing` (page 971)
: Sets a flag that indicates whether the layout is invalid and needs resizing. If *flag* is `true`, the menu contents have changed or the menu appearance has changed.

`needsSizing` (page 968)
: Returns `true` if the menu view needs to be resized due to changes in the NSMenu.

`sizeToFit` (page 972)
: Used internally by the menu view to cache information about the menu item geometry.

`stateImageOffset` (page 972)
: Returns the offset to the space reserved for state images of this menu.

`stateImageWidth` (page 972)
: Returns the maximum width of the state images used by this menu.

`imageAndTitleOffset` (page 964)
: Returns the offset to the starting point of a menu item's image and title section.

`imageAndTitleWidth` (page 965)
: Returns the maximum width of a menu item's image and title section.

`keyEquivalentOffset` (page 967)
: Returns the beginning position of the menu's key equivalent text.

`keyEquivalentWidth` (page 967)
: Returns the width of the menu's key equivalent text.

`innerRect` (page 965)
: Returns the drawing rectangle for the menu contents.

`rectOfItemAtIndex` (page 969)
: Returns the drawing rectangle of the menu item at *index*.

`indexOfItemAtPoint` (page 965)
: Returns the index of the menu item underneath *point* or –1 if no menu item is underneath *point*.

`setNeedsDisplayForItemAtIndex` (page 971)
: Adds the region occupied by the menu item at *index* to the menu view's invalid region.

`locationForSubmenu` (page 967)
: Returns the origin of the submenu view's window.

`setWindowFrameForAttachingToRect` (page 971)
: Causes the menu view to resize its window so its frame is the appropriate size to attach to *screenRect* within *screen*.

Event Handling

`performActionWithHighlightingForItemAtIndex` (page 968)
: Uses the associated NSMenu object to perform the action associated with the item at *`index`* when a key equivalent is pressed.

`trackWithEvent` (page 973)
: Handles events sent to this menu view.

Constructors

NSMenuView

This class is deprecated and is no longer used to draw menus.

```
public NSMenuView()
```

Discussion
Do not use this constructor.

This class is deprecated and is no longer used to draw menus.

```
public NSMenuView(boolean flag)
```

Discussion
Do not use this constructor.

Static Methods

menuBarHeight

Returns the height of the menu bar.

```
public static float menuBarHeight()
```

Discussion
This method is superseded in Mac OS X v10.4 by the NSMenu `menuBarHeight` (page 919) instance method.

Instance Methods

attachedMenu

Returns the NSMenu object associated with this object's attached menu view.

```
public NSMenu attachedMenu()
```

Discussion
The attached menu view is the one associated with the currently visible submenu, if any.

See Also
`attachedMenuView` (page 963)
`isAttached` (page 965)

attachedMenuView

Returns the NSMenuView of this object's attached menu view.

```
public NSMenuView attachedMenuView()
```

Discussion
The attached menu view is the one associated with the currently visible submenu, if any.

See Also
`attachedMenu` (page 962)
`detachSubmenu` (page 963)
`isAttached` (page 965)

attachSubmenuForItemAtIndex

Attaches the submenu associated with the menu item at *index*.

```
public void attachSubmenuForItemAtIndex(int index)
```

Discussion
This method prepares the submenu for display by positioning its window and ordering it to the front.

See Also
`setWindowFrameForAttachingToRect` (page 971)
`orderFront` (page 1844) (NSWindow)

detachSubmenu

Detaches the window associated with the currently visible submenu and removes any menu item highlights.

```
public void detachSubmenu()
```

Discussion
If the submenu itself displays further submenus, this method detaches the windows associated with those submenus as well.

See Also
`attachSubmenuForItemAtIndex` (page 963)
`setHighlightedItemIndex` (page 969)
`orderOut` (page 1845) (NSWindow)

font

Returns the default font used to draw the menu text.

```
public NSFont font()
```

Discussion

New items use this font by default, although the item's menu item cell can use a different font.

See Also

`setFont` (page 969)

highlightedItemIndex

Returns the index of the currently highlighted menu item, or –1 if no menu item in the menu is highlighted.

```
public int highlightedItemIndex()
```

See Also

`setHighlightedItemIndex` (page 969)

horizontalEdgePadding

Returns the amount of horizontal space used for padding menu item components.

```
public float horizontalEdgePadding()
```

Discussion

The edge padding is added to the sides of each menu item component. This space is used to provide a visual separation between components of the menu item.

See Also

`setHorizontalEdgePadding` (page 970)

imageAndTitleOffset

Returns the offset to the starting point of a menu item's image and title section.

```
public float imageAndTitleOffset()
```

Discussion

The image and title section of a menu item displays an image, a title, or possibly both as a way to identify the purpose of the menu item. The value returned by this method is used for all menu items of the menu.

If any changes have been made to the menu's contents, this method invokes `sizeToFit` (page 972) to update the menu view information.

See Also

`imageAndTitleWidth` (page 965)

`stateImageOffset` (page 972)

`keyEquivalentOffset` (page 967)

imageAndTitleWidth

Returns the maximum width of a menu item's image and title section.

```
public float imageAndTitleWidth()
```

Discussion

The image and title section of a menu item displays an image, a title, or possibly both as a way to identify the purpose of the menu item. The value returned by this method is used for all menu items of the menu.

If any changes have been made to the menu's contents, this method invokes `sizeToFit` (page 972) to update the menu view information.

See Also

`imageAndTitleOffset` (page 964)
`stateImageWidth` (page 972)
`keyEquivalentWidth` (page 967)

indexOfItemAtPoint

Returns the index of the menu item underneath *`point`* or –1 if no menu item is underneath *`point`*.

```
public int indexOfItemAtPoint(NSPoint point)
```

Discussion

This method considers the menu borders as part of the item when calculating whether *`point`* is in the menu item rectangle. This method invokes the `rectOfItemAtIndex` (page 969) method to obtain the basic rectangle for each menu item but may adjust that rectangle before testing.

See Also

`rectOfItemAtIndex` (page 969)

innerRect

Returns the drawing rectangle for the menu contents.

```
public NSRect innerRect()
```

Discussion

This rectangle is different (typically smaller) from the view bounds in that it does not include the space used to draw the menu borders.

See Also

`bounds` (page 1743) (NSView)

isAttached

Returns `true` if this menu is currently attached to its parent menu.

```
public boolean isAttached()
```

See Also

attachedMenu (page 962)

attachedMenuView (page 963)

isHorizontal

Returns true if the menu is displayed horizontally; such as for a menu bar, otherwise returns false.

```
public boolean isHorizontal()
```

See Also

setHorizontal (page 970)

isTornOff

Deprecated. Tear-off menus are not supported in Mac OS X.

```
public boolean isTornOff()
```

Discussion

Returns true if this menu view's window is disassociated from its parent menu.

itemAdded

Creates a new menu item cell for the newly created item and marks the menu view as needing to be resized.

```
public void itemAdded(NSNotification notification)
```

Discussion

This method is registered with the menu view's associated NSMenu object for notifications of the type MenuDidAddItemNotification (page 926). The *notification* parameter contains the notification data.

See Also

setNeedsSizing (page 971)

itemChanged

Marks the menu view as needing to be resized so changes in size resulting from a change in the menu will be tracked.

```
public void itemChanged(NSNotification notification)
```

Discussion

This method is registered with the menu view's associated NSMenu object for notifications of the type MenuDidChangeItemNotification (page 926). The *notification* parameter contains the notification data.

See Also

setNeedsSizing (page 971)

itemRemoved

Removes the removed item's menu item cell and marks the menu view as needing to be resized.

```
public void itemRemoved(NSNotification notification)
```

Discussion

This method is registered with the menu view's associated NSMenu object for notifications of the type `MenuDidRemoveItemNotification` (page 926). The *notification* parameter contains the notification data.

See Also

`setNeedsSizing` (page 971)

keyEquivalentOffset

Returns the beginning position of the menu's key equivalent text.

```
public float keyEquivalentOffset()
```

Discussion

If any changes have been made to the menu's contents, this method invokes `sizeToFit` (page 972) to update the menu view information.

See Also

`keyEquivalentWidth` (page 967)

`stateImageOffset` (page 972)

`imageAndTitleOffset` (page 964)

keyEquivalentWidth

Returns the width of the menu's key equivalent text.

```
public float keyEquivalentWidth()
```

Discussion

If any changes have been made to the menu's contents, this method invokes `sizeToFit` (page 972) to update the menu view information.

See Also

`keyEquivalentOffset` (page 967)

`stateImageWidth` (page 972)

`imageAndTitleWidth` (page 965)

locationForSubmenu

Returns the origin of the submenu view's window.

```
public NSPoint locationForSubmenu(NSMenu aSubMenu)
```

Discussion

The *aSubmenu* parameter specifies the submenu being positioned and must belong to a menu item of this menu view. This method positions the submenu adjacent to its menu item as well as possible given the type of menu and the space constraints of the user's screen.

If any changes have been made to the menu's contents, this method invokes `sizeToFit` (page 972) to update the menu view information.

See Also

`setWindowFrameForAttachingToRect` (page 971)

`sizeToFit` (page 972)

menu

Returns the NSMenu associated with this menu view.

```
public NSMenu menu()
```

See Also

`setMenu` (page 970)

menuItemCellForItemAtIndex

Returns the menu item cell at the specified *index*.

```
public NSMenuItemCell menuItemCellForItemAtIndex(int index)
```

See Also

`setMenuItemCellForItemAtIndex` (page 970)

`sizeToFit` (page 972)

needsSizing

Returns `true` if the menu view needs to be resized due to changes in the NSMenu.

```
public boolean needsSizing()
```

See Also

`setNeedsSizing` (page 971)

performActionWithHighlightingForItemAtIndex

Uses the associated NSMenu object to perform the action associated with the item at *index* when a key equivalent is pressed.

```
public void performActionWithHighlightingForItemAtIndex(int index)
```

Discussion

Because the menu item at index might not currently be visible, this method provides visual feedback by highlighting the nearest visible parent menu item before performing the action. After the action has been sent, this method removes the highlighting for the menu item.

See Also

`performActionForItemAtIndex` (page 920) (NSMenu)

rectOfItemAtIndex

Returns the drawing rectangle of the menu item at *`index`*.

```
public NSRect rectOfItemAtIndex(int index)
```

Discussion

The drawing rectangle may not be the same width or height as the actual menu and in fact is typically smaller to account for borders drawn by the menu view.

If any changes have been made to the menu's contents, this method invokes `sizeToFit` (page 972) to update the menu view information.

See Also

`innerRect` (page 965)
`needsSizing` (page 968)
`sizeToFit` (page 972)

setFont

Sets the default font to use when drawing the menu text.

```
public void setFont(NSFont font)
```

See Also

`font` (page 964)

setHighlightedItemIndex

Highlights the menu item at *`index`*.

```
public void setHighlightedItemIndex(int index)
```

Discussion

Specify –1 for *`index`* to remove all highlighting from the menu.

The rectangle of the menu item is marked as invalid and is redrawn the next time the event loop comes around. If another menu item was previously highlighted, that menu item is redrawn without highlights when the event loop comes around again.

See Also

`setNeedsDisplayForItemAtIndex` (page 971)
`highlightedItemIndex` (page 964)

setHorizontal

Sets the orientation of the menu.

```
public void setHorizontal(boolean flag)
```

Discussion

If *flag* is `true`, the menu's items are displayed horizontally; otherwise the menu's items are displayed vertically.

See Also

`isHorizontal` (page 966)

setHorizontalEdgePadding

Sets the horizontal padding for menu item components to *pad*.

```
public void setHorizontalEdgePadding(float pad)
```

See Also

`horizontalEdgePadding` (page 964)

setMenu

Sets the menu to be displayed in this view to *menu*.

```
public void setMenu(NSMenu menu)
```

Discussion

This method invokes the `setNeedsSizing` (page 971) method to force the menu view's layout to be recalculated before drawing.

This method adds the menu view to the new NSMenu object's list of observers. The notifications this method establishes notify this menu view when menu items in the NSMenu object are added, removed, or changed. This method removes the menu view from its previous NSMenu object's list of observers.

See Also

`setNeedsSizing` (page 971)

`itemAdded` (page 966)

`itemRemoved` (page 967)

`itemChanged` (page 966)

setMenuItemCellForItemAtIndex

Replaces the menu item cell at *index* with *cell*.

```
public void setMenuItemCellForItemAtIndex(NSMenuItemCell cell, int index)
```

Discussion

This method does not change the contents of the menu itself; it changes only the cell used to display the menu item at *index*. The old cell is released, and both the new cell and the menu view are marked as needing resizing.

See Also

`menuItemCellForItemAtIndex` (page 968)

`setNeedsSizing` (page 971)

setNeedsDisplayForItemAtIndex

Adds the region occupied by the menu item at *index* to the menu view's invalid region.

```
public void setNeedsDisplayForItemAtIndex(int index)
```

Discussion

The region to be redrawn includes the space occupied by the menu borders. This invalid region is redrawn the next time the event loop comes around.

See Also

`rectOfItemAtIndex` (page 969)

`setNeedsDisplay` (page 1779) (NSView)

setNeedsSizing

Sets a flag that indicates whether the layout is invalid and needs resizing. If *flag* is `true`, the menu contents have changed or the menu appearance has changed.

```
public void setNeedsSizing(boolean flag)
```

Discussion

This method is used internally; you should not need to invoke it directly unless you are implementing a subclass that can cause the layout to become invalid.

See Also

`sizeToFit` (page 972)

setWindowFrameForAttachingToRect

Causes the menu view to resize its window so its frame is the appropriate size to attach to *screenRect* within *screen*.

```
public void setWindowFrameForAttachingToRect(NSRect screenRect, NSScreen screen,
    int edge, int selectedItemIndex)
```

Discussion

If *selectedItemIndex* contains a value other than –1, this method attempts to position the menu such that the item at *selectedItemIndex* appears on top of *screenRect*.

The *selectedItemIndex* parameter specifies the amount by which the selected item's rectangle overlaps *screenRect*.

If the preferred edge, *edge*, cannot be honored, because there is not enough room, the opposite edge is used. If the rectangle does not completely fit either edge, this method uses the edge where there is more room.

If any changes have been made to the menu's contents, this method invokes `sizeToFit` (page 972) to update the menu view information.

See Also

`sizeToFit` (page 972)

sizeToFit

Used internally by the menu view to cache information about the menu item geometry.

```
public void sizeToFit()
```

Discussion

This cache is updated as necessary when menu items are added, removed, or changed.

The geometry of each menu item is determined by asking its corresponding menu item cell. The menu item cell is obtained from the `menuItemCellForItemAtIndex` (page 968) method.

See Also

`setNeedsSizing` (page 971)
`menuItemCellForItemAtIndex` (page 968)

stateImageOffset

Returns the offset to the space reserved for state images of this menu.

```
public float stateImageOffset()
```

Discussion

The offset is used for all menu items of the menu.

If any changes have been made to the menu's contents, this method invokes `sizeToFit` (page 972) to update the menu view information.

See Also

`horizontalEdgePadding` (page 964)
`setHorizontalEdgePadding` (page 970)
`sizeToFit` (page 972)

stateImageWidth

Returns the maximum width of the state images used by this menu.

```
public float stateImageWidth()
```

Discussion

The width is used for all menu items of the menu.

If any changes have been made to the menu's contents, this method invokes `sizeToFit` (page 972) to update the menu view information.

See Also

`sizeToFit` (page 972)

trackWithEvent

Handles events sent to this menu view.

```
public boolean trackWithEvent(NSEvent event)
```

Discussion

If *event* is a mouse event, this method tracks the cursor position in the menu and displays the menus as appropriate. This method also handles mouse clicks that result in the selection of a menu item, in which case the menu item's action is performed.

You should not need to use this method directly.

update

Asks the associated NSMenu to update itself.

```
public void update()
```

Discussion

If any changes have been made to the menu's contents, this method invokes `sizeToFit` (page 972) to update the menu view's layout.

See Also

`sizeToFit` (page 972)

`setNeedsSizing` (page 971)

`update` (page 923) (NSMenu)

NSModalSession

Inherits from	Object
Package:	com.apple.cocoa.application
Companion guide	Window Programming Guide for Cocoa

Overview

An application uses an NSModalSession object when it begins and runs a modal session. An NSModalSession object encapsulates certain information about a session, such as the application, window, and graphics context involved. Because NSModalSession is a final class, you cannot subclass it. The NSApplication method `beginModalSessionForWindow` (page 108) creates a modal session, and the NSApplication methods `runModalSession` (page 120) and `endModalSession` (page 111) take it as an argument.

Tasks

Constructors

`NSModalSession` (page 975)

Constructors

NSModalSession

```
public NSModalSession(int sessionID)
```

Discussion
Returns an NSModalSession identified by the session number *sessionID*, which must be unique for the session.

NSMovie

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.application
Companion guide	Video

Overview

An NSMovie is a wrapper for a QuickTime `Movie`, providing a simple interface for loading a movie into memory. The movie data can come from a URL or a pasteboard, including the drag-and-drop and cut-and-paste pasteboards. The data can be of any type recognized by QuickTime, including nonvideo data such as pure audio or even still images. Once loaded, you can obtain a pointer to the movie data and use the extensive QuickTime APIs to manipulate the data.

Tasks

Constructors

`NSMovie` (page 978)
Creates an empty NSMovie instance.

Checking Data Types

`canInitWithPasteboard` (page 978)
Tests whether the class can initialize an instance of itself from the data on *`pasteboard`*.

`movieUnfilteredFileTypes` (page 978)
Returns an array of strings representing those file types that contain supported movie data.

`movieUnfilteredPasteboardTypes` (page 979)
Returns an array of pasteboard types from which an NSMovie instance can be created.

Accessing Movie Information

`URL` (page 979)
Returns the URL of the file used to initialize the receiver.

Constructors

NSMovie

Creates an empty NSMovie instance.

```
public NSMovie()
```

Creates a new NSMovie instance initialized with data from *aPasteboard*.

```
public NSMovie(NSPasteboard aPasteboard)
```

Discussion

aPasteboard should contain data either of a type returned by `movieUnfilteredPasteboardTypes` (page 979) or of type `NSPasteboard.FilenamesPboardType`. In the latter case, the filename on *aPasteboard* should be for a file of a type returned by `movieUnfilteredFileTypes` (page 978). If multiple filenames are on *aPasteboard*, only the first name is used.

When archiving an NSMovie object, the movie data is encoded if the data was obtained directly from *aPasteboard*. If instead a filename was on the pasteboard, only the file's URL is encoded.

Creates a new NSMovie instance initialized with data located at *aURL*.

```
public NSMovie(java.net.URL aURL, boolean byRef)
```

Discussion

aURL can use any appropriate URL protocol, including `file:`, `http:`, or `rtsp:`, and reference any type of data recognized by QuickTime, including video, pure audio, or still images. If *byRef* is `true`, only the URL is encoded when the NSMovie is archived. If *byRef* is `false`, the movie's QuickTime header information is encoded.

Static Methods

canInitWithPasteboard

Tests whether the class can initialize an instance of itself from the data on *pasteboard*.

```
public static boolean canInitWithPasteboard(NSPasteboard pasteboard)
```

Discussion

Returns `true` if the receiver's list of supported pasteboard types includes a data type available from *pasteboard*.

See Also

`movieUnfilteredPasteboardTypes` (page 979)

movieUnfilteredFileTypes

Returns an array of strings representing those file types that contain supported movie data.

```
public static NSArray movieUnfilteredFileTypes()
```

Discussion

The default list contains the filename extensions “mov” and “MOV” and the HFS file type “‘MooV’”. The array returned by this method may be passed directly to NSOpenPanel’s `runModalForTypes` (page 1024) method.

See Also

`movieUnfilteredPasteboardTypes` (page 979)

movieUnfilteredPasteboardTypes

Returns an array of pasteboard types from which an NSMovie instance can be created.

```
public static NSArray movieUnfilteredPasteboardTypes()
```

Discussion

Compare the elements of this array to the available data types on a pasteboard to detect the presence of movie data.

See Also

`canInitWithPasteboard` (page 978)

Instance Methods

URL

Returns the URL of the file used to initialize the receiver.

```
public java.net.URL URL()
```

Discussion

If the receiver was not initialized from a file, it returns `null`.

NSMovieView

Inherits from	NSView : NSResponder : NSObject
Implements	NSCoding (NSResponder)
Package:	com.apple.cocoa.application
Companion guide	Video

Overview

An NSMovieView displays an NSMovie (a wrapper for a QuickTime movie) in a frame and provides methods for playing and editing the movie. The view can optionally display a standard QuickTime movie controller, or you can provide your own interface linked to NSMovieView's action methods.

Tasks

Constructors

`NSMovieView` (page 983)
Creates an NSMovieView instance with a zero-size frame.

Setting Movie

`movie` (page 987)
Returns the NSMovie object displayed in the view.

`setMovie` (page 989)
Sets the NSMovie displayed in the view to *movie*.

Playing a Movie

`gotoBeginning` (page 985)
This action method repositions the play position to the beginning of the movie.

`gotoEnd` (page 985)
This action method repositions the play position to the end of the movie.

`gotoPosterFrame` (page 985)
: This action method repositions the play position to the movie's poster frame.

`isPlaying` (page 986)
: Returns `true` if the movie is currently playing.

`rate` (page 988)
: Returns the relative frame rate at which the movie is to be played.

`setRate` (page 990)
: Sets the frame rate, relative to the movie's internal frame rate, at which to play the movie.

`start` (page 991)
: This action method starts the movie playing at its current location.

`stepBack` (page 991)
: This action method repositions the movie's play position to one frame before the current frame.

`stepForward` (page 992)
: This action method repositions the movie's play position to one frame after the current frame.

`stop` (page 992)
: This action method stops the movie.

Sound

`isMuted` (page 986)
: Returns `true` if the movie's sound is currently muted.

`setMuted` (page 989)
: Sets whether the movie's sound is muted.

`setVolume` (page 990)
: Sets the relative sound volume of the movie.

`volume` (page 992)
: Returns the relative volume at which the movie is to be played. Default is 1.0.

Play Modes

`loopMode` (page 987)
: Returns the playback behavior for when the end of the movie is reached.

`setLoopMode` (page 989)
: Sets the playback behavior for when the end of the movie is reached.

`playsSelectionOnly` (page 988)
: Returns `true` if the movie is configured to play only the selected portion.

`setPlaysSelectionOnly` (page 990)
: Sets whether only the selected portion of the movie is played to *`flag`*.

`playsEveryFrame` (page 987)
: Returns `true` if the movie is configured to display every frame when playing.

`setPlaysEveryFrame` (page 990)
: Sets whether the movie plays every frame of the movie.

Setting Controller

isControllerVisible (page 986)
: Returns true if the movie controller is visible.

showController (page 991)
: Sets whether a standard QuickTime movie controller is displayed beneath the movie to *show*.

Sizing

movieRect (page 987)
: Returns the rectangle into which the movie is to be placed.

resizeWithMagnification (page 988)
: Resizes the view's frame to the size required to display the movie with a magnification of *magnification* and with a movie controller below it.

sizeForMagnification (page 991)
: Returns the required size of the movie view if the movie were magnified to *magnification*.

Editing

clear (page 984)
: This action method deletes the current movie selection from the movie.

copy (page 984)
: This action method copies the current movie selection onto the clipboard.

cut (page 984)
: This action method deletes the current movie selection from the movie, placing it on the clipboard.

delete (page 985)
: This action method deletes the current movie selection from the movie, placing it on the clipboard.

paste (page 987)
: This action method inserts the contents of the clipboard (if it contains a movie clip) into the movie at the current play position.

selectAll (page 988)
: This action method selects the entire movie.

isEditable (page 986)
: Returns true if the movie is editable.

setEditable (page 988)
: Sets whether the movie can be edited.

Constructors

NSMovieView

Creates an NSMovieView instance with a zero-size frame.

```
public NSMovieView()
```

Creates an NSMovieView instance with a frame of *aRect*.

```
public NSMovieView(NSRect aRect)
```

Instance Methods

clear

This action method deletes the current movie selection from the movie.

```
public void clear(Object sender)
```

Discussion
If there is no selection, the current frame is deleted. This action is undoable. If the movie is not editable, this method does nothing.

This method has been deprecated use `delete` (page 985) instead.

Availability
Deprecated in Mac OS X v10.3.

copy

This action method copies the current movie selection onto the clipboard.

```
public void clear(Object sender)
```

Discussion
If there is no selection, the current frame is copied. The movie does not need to be editable.

See Also
`paste` (page 987)

cut

This action method deletes the current movie selection from the movie, placing it on the clipboard.

```
public void cut(Object sender)
```

Discussion
If there is no selection, the current frame is deleted. This action is undoable. If the movie is not editable, this method does nothing.

See Also
`paste` (page 987)

delete

This action method deletes the current movie selection from the movie, placing it on the clipboard.

```
public void delete(Object sender)
```

Discussion

If there is no selection, the current frame is deleted. This action is undoable. If the movie is not editable, this method does nothing.

Availability

Available in Mac OS X v10.3 and later.

gotoBeginning

This action method repositions the play position to the beginning of the movie.

```
public void gotoBeginning(Object sender)
```

Discussion

If the movie is playing, the movie continues playing from the new position.

See Also

`gotoEnd` (page 985)

`gotoPosterFrame` (page 985)

gotoEnd

This action method repositions the play position to the end of the movie.

```
public void gotoEnd(Object sender)
```

Discussion

If the movie is playing in one of the looping modes, the movie continues playing accordingly; otherwise, play stops.

See Also

`gotoBeginning` (page 985)

`gotoPosterFrame` (page 985)

gotoPosterFrame

This action method repositions the play position to the movie's poster frame.

```
public void gotoPosterFrame(Object sender)
```

Discussion

If no poster frame is defined, the movie jumps to the beginning. If the movie is playing, the movie continues playing from the new position.

See Also

`gotoBeginning` (page 985)

`gotoEnd` (page 985)

isControllerVisible

Returns `true` if the movie controller is visible.

```
public boolean isControllerVisible()
```

Discussion
The default is `true`.

See Also
`showController` (page 991)

isEditable

Returns `true` if the movie is editable.

```
public boolean isEditable()
```

Discussion
When editable, a movie can be modified using the `clear` (page 984), `cut` (page 984), and `paste` (page 987) methods and associated key commands. You can also drag movie files into the view, replacing the movie. The default is `true`.

See Also
`setEditable` (page 988)

isMuted

Returns `true` if the movie's sound is currently muted.

```
public boolean isMuted()
```

See Also
`setMuted` (page 989)

isPlaying

Returns `true` if the movie is currently playing.

```
public boolean isPlaying()
```

See Also
`start` (page 991)
`stop` (page 992)

loopMode

Returns the playback behavior for when the end of the movie is reached.

```
public int loopMode()
```

Discussion

Return value is one of the constants defined in "Constants" (page 992). Default is `NormalPlayback`.

See Also

`setLoopMode` (page 989)

movie

Returns the NSMovie object displayed in the view.

```
public NSMovie movie()
```

See Also

`setMovie` (page 989)

movieRect

Returns the rectangle into which the movie is to be placed.

```
public NSRect movieRect()
```

Discussion

By default, this method returns the view's bounding rectangle. Override this method if you want the movie to be positioned or sized differently within the view.

paste

This action method inserts the contents of the clipboard (if it contains a movie clip) into the movie at the current play position.

```
public void paste(Object sender)
```

Discussion

This action is undoable. If the movie is not editable, this method does nothing.

See Also

`copy` (page 984)

`cut` (page 984)

playsEveryFrame

Returns `true` if the movie is configured to display every frame when playing.

```
public boolean playsEveryFrame()
```

Discussion
Default is `false`.

See Also
`setPlaysEveryFrame` (page 990)

playsSelectionOnly

Returns `true` if the movie is configured to play only the selected portion.

```
public boolean playsSelectionOnly()
```

Discussion
Default is `false`.

See Also
`setPlaysSelectionOnly` (page 990)

rate

Returns the relative frame rate at which the movie is to be played.

```
public float rate()
```

Discussion
The default value of 1.0 indicates the normal frame rate defined by the movie.

See Also
`setRate` (page 990)

resizeWithMagnification

Resizes the view's frame to the size required to display the movie with a magnification of *`magnification`* and with a movie controller below it.

```
public void resizeWithMagnification(float magnification)
```

See Also
`sizeForMagnification` (page 991)

selectAll

This action method selects the entire movie.

```
public void selectAll(Object sender)
```

setEditable

Sets whether the movie can be edited.

```
public void setEditable(boolean flag)
```

Discussion

If *flag* is `true`, you can use the `clear` (page 984), `cut` (page 984), and `paste` (page 987) methods and associated key commands to modify the movie. You can also drag a new movie file into the view, replacing the current movie. If *flag* is `false`, these features are disabled.

See Also

`isEditable` (page 986)

setLoopMode

Sets the playback behavior for when the end of the movie is reached.

```
public void setLoopMode(int flag)
```

Discussion

flag is one of the constants defined in "Constants" (page 992). If *flag* is `NormalPlayback`, the movie stops playing when it reaches the end. If *flag* is `LoopingPlayback`, the movie will continue playing at the beginning. If *flag* is `LoopingBackAndForthPlayback`, the movie will play in reverse, then forward again, as it reaches each end of the movie. If `playsSelectionOnly` (page 988) is `true`, these behaviors apply to the endpoints of the selection, not the movie.

See Also

`loopMode` (page 987)

setMovie

Sets the NSMovie displayed in the view to *movie*.

```
public void setMovie(NSMovie movie)
```

See Also

`movie` (page 987)

setMuted

Sets whether the movie's sound is muted.

```
public void setMuted(boolean flag)
```

Discussion

When *flag* is `true`, muting is turned on. When muting is turned off again by sending `false` for *flag*, the previous sound volume is restored.

See Also

`isMuted` (page 986)

`setVolume` (page 990)

setPlaysEveryFrame

Sets whether the movie plays every frame of the movie.

```
public void setPlaysEveryFrame(boolean flag)
```

Discussion

If *flag* is `true`, every frame of the movie is displayed, even if this requires playing the movie slower than its preferred rate. If *flag* is `false`, the movie may skip some frames if needed to maintain its time sequence.

If *flag* is `true`, audio is not played.

See Also

`playsEveryFrame` (page 987)

setPlaysSelectionOnly

Sets whether only the selected portion of the movie is played to *flag*.

```
public void setPlaysSelectionOnly(boolean flag)
```

Discussion

If there is no selection, the entire movie is played.

See Also

`playsSelectionOnly` (page 988)

setRate

Sets the frame rate, relative to the movie's internal frame rate, at which to play the movie.

```
public void setRate(float rate)
```

Discussion

The default *rate* of 1.0 indicates the movie is played at its normal rate. Larger values indicate faster rates, and fractional values indicate slower rates. Negative values are allowed, causing the movie to play in reverse. Invoking this method does not automatically start the movie playing.

This value is ignored if the movie is started using the movie controller, which always plays the movie at the normal rate.

See Also

`rate` (page 988)

setVolume

Sets the relative sound volume of the movie.

```
public void setVolume(float volume)
```

Discussion

The default *volume* of 1.0 indicates the current system volume.

See Also

`setMuted` (page 989)

`volume` (page 992)

showController

Sets whether a standard QuickTime movie controller is displayed beneath the movie to *show*.

```
public void showController(boolean show, boolean adjustSize)
```

Discussion

If *adjustSize* is `true`, the view's height is modified so that the size and position of the movie are unchanged. If *adjustSize* is `false`, the view's size is unchanged, and the movie is resized to fit into the frame. The adjustment is made only if the visibility of the controller is indeed changed.

See Also

`isControllerVisible` (page 986)

sizeForMagnification

Returns the required size of the movie view if the movie were magnified to *magnification*.

```
public NSSize sizeForMagnification(float magnification)
```

Discussion

An extra 16 pixels are added to the vertical dimension to allow room for the movie controller, even if it is currently hidden.

See Also

`resizeWithMagnification` (page 988)

start

This action method starts the movie playing at its current location.

```
public void start(Object sender)
```

Discussion

This method does nothing if the movie is already playing.

See Also

`isPlaying` (page 986)

`stop` (page 992)

stepBack

This action method repositions the movie's play position to one frame before the current frame.

```
public void stepBack(Object sender)
```

Discussion
If the movie is playing, the movie will stop at the new frame.

See Also
`stepForward` (page 992)

stepForward

This action method repositions the movie's play position to one frame after the current frame.

```
public void stepForward(Object sender)
```

Discussion
If the movie is playing, the movie will stop at the new frame.

See Also
`stepBack` (page 991)

stop

This action method stops the movie.

```
public void stop(Object sender)
```

See Also
`isPlaying` (page 986)
`start` (page 991)

volume

Returns the relative volume at which the movie is to be played. Default is 1.0.

```
public float volume()
```

See Also
`setVolume` (page 990)

Constants

The following constants are defined as a convenience by NSMovieView:

Constant	Description
`LoopingBackAndForthPlayback`	Playback runs forward and backward between both endpoints.
`LoopingPlayback`	Restarts playback at beginning when end is reached.
`NormalPlayback`	Playback stops when end is reached.

NSMutableParagraphStyle

Inherits from	NSParagraphStyle : NSObject
Implements	NSCoding (NSParagraphStyle)
Package:	com.apple.cocoa.application
Companion guide	Rulers and Paragraph Styles

Overview

NSMutableParagraphStyle adds methods to its superclass, NSParagraphStyle, for changing the values of the subattributes in a paragraph style attribute. See the NSParagraphStyle (page 1057) and NSAttributedString specifications for more information.

Tasks

Constructors

`NSMutableParagraphStyle` (page 995)
Creates an empty NSMutableParagraphStyle.

Setting Tab Stops

`setTabStops` (page 1000)
Replaces the tab stops in the receiver with *`tabStops`*.

`addTabStop` (page 995)
Adds *`tabStop`* to the receiver.

`removeTabStop` (page 995)
Removes the first text tab whose location and type are equal to those of *`tabStop`*.

Setting Other Style Information

`setParagraphStyle` (page 1000)
Replaces the subattributes of the receiver with those in *`aStyle`*.

`setAlignment` (page 995)
: Sets the alignment of the receiver to *alignment*.

`setFirstLineHeadIndent` (page 997)
: Sets the distance in points from the leading margin of a text container to the beginning of the paragraph's first line to *aFloat*.

`setHeadIndent` (page 997)
: Sets the distance in points from the leading margin of a text container to the beginning of lines other than the first to *aFloat*.

`setTailIndent` (page 1000)
: Sets the distance in points from the margin of a text container to the end of lines to *aFloat*.

`setLineBreakMode` (page 998)
: Sets the mode used to break lines in a layout container to *mode*.

`setMaximumLineHeight` (page 999)
: Sets the maximum height that any line in the paragraph style will occupy, regardless of the font size or size of any attached graphic, to *aFloat*.

`setMinimumLineHeight` (page 999)
: Sets the minimum height that any line in the paragraph style will occupy, regardless of the font size or size of any attached graphic, to *aFloat*.

`setLineSpacing` (page 998)
: Sets the space in points added between lines within the paragraph to *aFloat*.

`setParagraphSpacing` (page 999)
: Sets the space added at the end of the paragraph to separate it from the following paragraph to *aFloat*.

`setBaseWritingDirection` (page 996)
: Sets the base writing direction for the receiver.

`setLineHeightMultiple` (page 998)
: Sets the line height multiple for the receiver.

`setParagraphSpacingBefore` (page 1000)
: Sets the distance between the paragraph's top and the beginning of its text content

`setDefaultTabInterval` (page 996)
: Sets the default tab interval for the receiver.

Setting Text Blocks and Lists

`setTextBlocks` (page 1001)
: Sets the text blocks containing the paragraph, nested from outermost to innermost to *array*.

`setTextLists` (page 1001)

Controlling Hyphenation and Truncation

`setHyphenationFactor` (page 997)
: Specifies the threshold for hyphenation.

`setTighteningFactorForTruncation` (page 1001)
: Specifies the threshold for using tightening as an alternative to truncation.

Setting HTML Header Level

`setHeaderLevel` (page 997)
: Specifies whether the paragraph is to be treated as a header for purposes of HTML generation.

Constructors

NSMutableParagraphStyle

Creates an empty NSMutableParagraphStyle.

```
public NSMutableParagraphStyle()
```

Instance Methods

addTabStop

Adds *tabStop* to the receiver.

```
public void addTabStop(NSTextTab tabStop)
```

See Also

`removeTabStop` (page 995)
`setTabStops` (page 1000)
`tabStops` (page 1064) (NSParagraphStyle)

removeTabStop

Removes the first text tab whose location and type are equal to those of *tabStop*.

```
public void removeTabStop(NSTextTab tabStop)
```

See Also

`addTabStop` (page 995)
`setTabStops` (page 1000)
`tabStops` (page 1064) (NSParagraphStyle)

setAlignment

Sets the alignment of the receiver to *alignment*.

```
public void setAlignment(int alignment)
```

Discussion

`alignment` may be one of:

```
NSText.LeftTextAlignment
NSText.RightTextAlignment
NSText.CenterTextAlignment
NSText.JustifiedTextAlignment
NSText.NaturalTextAlignment
```

See Also

`alignment` (page 1060) (NSParagraphStyle)

setBaseWritingDirection

Sets the base writing direction for the receiver.

```
public void setBaseWritingDirection(int writingDirection)
```

Discussion

It can be `WritingDirectionNaturalDirection`, `WritingDirectionLeftToRight`, or `WritingDirectionRightToLeft`. If you specify `WritingDirectionNaturalDirection`, the receiver resolves the writing direction to either `WritingDirectionLeftToRight` or `WritingDirectionRightToLeft`, depending on the direction for the user's language preference setting.

Availability

Available in Mac OS X v10.2 and later.

See Also

`defaultWritingDirectionForLanguage` (page 1060) (NSParagraphStyle)

`baseWritingDirection` (page 1060) (NSParagraphStyle)

setDefaultTabInterval

Sets the default tab interval for the receiver.

```
public void setDefaultTabInterval(float aFloat)
```

Discussion

Tabs after the last specified in `tabStops` (page 1064) are placed at integral multiples of this distance. This value must be nonnegative.

Availability

Available in Mac OS X v10.3 and later.

See Also

`defaultTabInterval` (page 1061) (NSParagraphStyle)

setFirstLineHeadIndent

Sets the distance in points from the leading margin of a text container to the beginning of the paragraph's first line to *aFloat*.

```
public void setFirstLineHeadIndent(float aFloat)
```

Discussion

This value must be nonnegative.

See Also

`setHeadIndent` (page 997)

`setTailIndent` (page 1000)

`firstLineHeadIndent` (page 1061) (NSParagraphStyle)

setHeaderLevel

Specifies whether the paragraph is to be treated as a header for purposes of HTML generation.

```
public void setHeaderLevel(int level)
```

Discussion

Should be set to 0 (the default value) if the paragraph is not a header, or from 1 through 6 if the paragraph is to be treated as a header.

Availability

Available in Mac OS X v10.4 and later.

See Also

`headerLevel` (page 1061) (NSParagraphStyle)

setHeadIndent

Sets the distance in points from the leading margin of a text container to the beginning of lines other than the first to *aFloat*.

```
public void setHeadIndent(float aFloat)
```

Discussion

This value must be nonnegative.

See Also

`setFirstLineHeadIndent` (page 997)

`setTailIndent` (page 1000)

`headIndent` (page 1062) (NSParagraphStyle)

setHyphenationFactor

Specifies the threshold for hyphenation.

```
public void setHyphenationFactor(float aFactor)
```

Discussion

Valid values lie between 0.0 and 1.0 inclusive. The default value is 0.0. Hyphenation is attempted when the ratio of the text width (as broken without hyphenation) to the width of the line fragment is less than the hyphenation factor. When the paragraph's hyphenation factor is 0.0, the layout manager's hyphenation factor is used instead. When both are 0.0, hyphenation is disabled.

Availability

Available in Mac OS X v10.4 and later.

See Also

`hyphenationFactor` (page 1062) (NSParagraphStyle)

setLineBreakMode

Sets the mode used to break lines in a layout container to *mode*.

```
public void setLineBreakMode(int mode)
```

Discussion

mode may be one of:

```
NSParagraphStyle.LineBreakByWordWrapping
NSParagraphStyle.LineBreakByCharWrapping
NSParagraphStyle.LineBreakByClipping
NSParagraphStyle.LineBreakByTruncatingHead
NSParagraphStyle.LineBreakByTruncatingTail
NSParagraphStyle.LineBreakByTruncatingMiddle
```

See the description of `lineBreakMode` (page 1062) in the NSParagraphStyle class specification for descriptions of these values.

setLineHeightMultiple

Sets the line height multiple for the receiver.

```
public void setLineHeightMultiple(float aFloat)
```

Discussion

The natural line height of the receiver is multiplied by this factor before being constrained by minimum and maximum line height. This value must be nonnegative.

Availability

Available in Mac OS X v10.3 and later.

See Also

`lineHeightMultiple` (page 1062) (NSParagraphStyle)

setLineSpacing

Sets the space in points added between lines within the paragraph to *aFloat*.

```
public void setLineSpacing(float aFloat)
```

Discussion
This value must be nonnegative.

See Also
`setMaximumLineHeight` (page 999)
`setMinimumLineHeight` (page 999)
`setParagraphSpacing` (page 999)
`lineSpacing` (page 1063) (NSParagraphStyle)

setMaximumLineHeight

Sets the maximum height that any line in the paragraph style will occupy, regardless of the font size or size of any attached graphic, to *aFloat*.

```
public void setMaximumLineHeight(float aFloat)
```

Discussion
Glyphs and graphics exceeding this height will overlap neighboring lines; however, a maximum height of 0 implies no line height limit. This value must be nonnegative.

Although this limit applies to the line itself, line spacing adds extra space between adjacent lines.

See Also
`setMinimumLineHeight` (page 999)
`setLineSpacing` (page 998)
`maximumLineHeight` (page 1063) (NSParagraphStyle)

setMinimumLineHeight

Sets the minimum height that any line in the paragraph style will occupy, regardless of the font size or size of any attached graphic, to *aFloat*.

```
public void setMinimumLineHeight(float aFloat)
```

Discussion
This value must be nonnegative.

See Also
`setMaximumLineHeight` (page 999)
`setLineSpacing` (page 998)
`minimumLineHeight` (page 1063) (NSParagraphStyle)

setParagraphSpacing

Sets the space added at the end of the paragraph to separate it from the following paragraph to *aFloat*.

```
public void setParagraphSpacing(float aFloat)
```

Discussion
This value must be nonnegative.

See Also
`setLineSpacing` (page 998)
`paragraphSpacing` (page 1064) (NSParagraphStyle)

setParagraphSpacingBefore

Sets the distance between the paragraph's top and the beginning of its text content

```
public void setParagraphSpacingBefore(float aFloat)
```

Discussion
. This value must be nonnegative.

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

See Also
`setParagraphSpacing` (page 999)
`paragraphSpacingBefore` (page 1064) (NSParagraphStyle)

setParagraphStyle

Replaces the subattributes of the receiver with those in *aStyle*.

```
public void setParagraphStyle(NSParagraphStyle aStyle)
```

setTabStops

Replaces the tab stops in the receiver with *tabStops*.

```
public void setTabStops(NSArray tabStops)
```

See Also
`addTabStop` (page 995)
`removeTabStop` (page 995)
`tabStops` (page 1064) (NSParagraphStyle)

setTailIndent

Sets the distance in points from the margin of a text container to the end of lines to *aFloat*.

```
public void setTailIndent(float aFloat)
```

Discussion
If positive, this is the distance from the leading margin (for example, the left margin in left-to-right text). That is, it's the absolute line width. If 0 or negative, it's the distance from the trailing margin—the value is added to the line width.

For example, to create a paragraph style that fits exactly in a 2-inch wide container, set its head indent to 0.0 and its tail indent to 0.0. To create a paragraph style with quarter-inch margins, set its head indent to 0.25 and its tail indent to –0.25.

See Also
`setHeadIndent` (page 997)
`setFirstLineHeadIndent` (page 997)
`tailIndent` (page 1065) (NSParagraphStyle)

setTextBlocks

Sets the text blocks containing the paragraph, nested from outermost to innermost to *array*.

```
public void setTextBlocks(NSArray array)
```

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

See Also
`textBlocks` (page 1065) (NSParagraphStyle)

setTextLists

```
public void setTextLists(NSArray array)
```

Discussion
Sets the text lists containing the paragraph, nested from outermost to innermost, to *array*.

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

See Also
`textLists` (page 1065) (NSParagraphStyle)

setTighteningFactorForTruncation

Specifies the threshold for using tightening as an alternative to truncation.

```
public void setTighteningFactorForTruncation(float aFactor)
```

Discussion
When the line break mode specifies truncation, the text system attempts to tighten intercharacter spacing as an alternative to truncation, provided that the ratio of the text width to the line fragment width does not exceed 1.0 + the value returned by `tighteningFactorForTruncation` (page 1065). Otherwise the text is truncated at a location determined by the line break mode. This method accepts positive and negative values, excluding 0. The default value is 0.05.

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

See Also

`tighteningFactorForTruncation` (page 1065) (NSParagraphStyle)

NSNib

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.application
Availability	Available in Mac OS X v10.3 and later.
Companion guide	Resource Programming Guide

Overview

Instances of the NSNib class serve as object wrappers, or containers, for Interface Builder nib files. An NSNib object keeps the contents of a nib file resident in memory, ready for unarchiving and instantiation.

When you create an NSNib instance from a nib file, the object loads the contents of the referenced nib bundle—the object graph as well as any images and sounds—into memory; but it does not yet unarchive it. To unarchive all of the nib data and thus truly instantiate the nib you must call an NSNib `instantiate...` method.

During the instantiation process, each object in the archive is unarchived and then initialized with the method befitting its type. View classes are initialized using their `initWithFrame:` method. Custom objects are initialized using their `init` method. In the case of Cocoa views (and custom views that have options on an associated Interface Builder palette) the initialization process also reads in any values set by the user in Interface Builder.

Once all objects have been instantiated and initialized from the archive, the nib loading code attempts to reestablish the connections between each object's outlets and the corresponding target objects. If your custom objects have outlets, NSNib attempts to reestablish any connections you created in Interface Builder. It starts by trying to establish the connections using your object's own methods first. For each outlet that needs a connection, NSNib looks for a method of the form `set`*OutletName*`:` in your object. If that method exists, NSNib calls it, passing the target object as a parameter. If you did not define a setter method with that exact name, NSNib searches the object for an instance variable (of type `IBOutlet id`) with the corresponding outlet name and tries to set its value directly. If an instance variable with the correct name cannot be found, initialization of that connection does not occur.

Subclassing Notes

You can subclass NSNib if you want to extend or specialize nib-loading behavior. For example, you could create a custom NSNib subclass that performs some post-processing on the top-level objects returned from the `instantiateNib...` methods. If you want to modify how nib instantiations are performed, it is recommended that you override the primitive method `instantiateNibWithExternalNameTable` (page 1005). Note that the instance variables of NSNib are private and thus are not available to subclasses.

Tasks

Constructors

`NSNib` (page 1004)
Creates and returns an empty NSNib.

Instantiating a Nib

`instantiateNibWithOwner` (page 1005)
Unarchives and instantiates the in-memory contents of the nib file represented by the receiver, creating a distinct object tree and top level objects.

`instantiateNibWithExternalNameTable` (page 1005)
Unarchives and instantiates the in-memory contents of the nib file represented by the receiver, creating a distinct object tree and top level objects.

Constructors

NSNib

Creates and returns an empty NSNib.

```
public NSNib()
```

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

Creates and returns an NSNib representing the nib file located at *nibFileURL*.

```
public NSNib(java.net.URL nibFileURL)
```

Discussion
Because this constructor does not associate the nib file with a bundle, the owner's bundle, if specified, is used for the resource map during nib instantiation. If no owner is specified, the application's main bundle is used instead. Returns `null` if it cannot locate the nib file or if there are any other errors during creation.

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

Creates and returns an NSNib representing a nib file named *nibName* in the specified bundle, *bundle*.

```
public NSNib(String nibName, NSBundle bundle)
```

Discussion
If *bundle* is `null`, this constructor looks for the nib file in the main bundle. NSNib uses this bundle for its resource map during instantiation. Returns `null` if it cannot locate the nib file or if there are any other errors during creation.

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

Instance Methods

instantiateNibWithExternalNameTable

Unarchives and instantiates the in-memory contents of the nib file represented by the receiver, creating a distinct object tree and top level objects.

```
public boolean instantiateNibWithExternalNameTable(NSDictionary externalNameTable)
```

Discussion
The sole argument, *externalNameTable*, may contain the nib file's owner and, optionally, an NSMutableArray object that will be populated with the top-level objects of the object tree upon return. With this method you may instantiate a nib file multiple times. Each instantiation of the nib must have a distinct owner object that is responsible for the resulting object tree. This is the primitive method for performing instantiations of a nib file.

Returns `true` if the nib was successfully instantiated.

The possible keys and values for *externalNameTable* are described in "Constants" (page 1006).

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

instantiateNibWithOwner

Unarchives and instantiates the in-memory contents of the nib file represented by the receiver, creating a distinct object tree and top level objects.

```
public boolean instantiateNibWithOwner(Object owner, NSMutableArray topLevelObjects)
```

Discussion
The object owning the nib file must be specified in *owner*. If *topLevelObjects* is non-`null`, it points to, upon return, an array of the top-level objects of the instantiated object tree. With this method you may instantiate a nib file multiple times. This is a convenience method that composes the name-table dictionary and invokes the `instantiateNibWithExternalNameTable` (page 1005), passing it the name table.

Returns `true` if the nib was successfully instantiated.

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

Constants

NSNib defines the following constants which are used as keys in the dictionary passed to `instantiateNibWithExternalNameTable` (page 1005):

Constant	Description
`NibOwner`	The external object that is responsible for the instantiated nib (File's Owner).
`TopLevelObjects`	An NSMutableArray object that, if present, is populated with the top-level objects of the newly instantiated nib.

NSObjectController

Inherits from	NSController : NSObject
Implements	NSCoding (NSController)
Package:	com.apple.cocoa.application
Availability	Available in Mac OS X v10.3 and later.
Companion guides	Cocoa Bindings Programming Topics Predicate Programming Guide Core Data Programming Guide

Overview

NSObjectController is a Cocoa bindings compatible controller class. Properties of the content object of an instance of this class can be bound to user interface elements to access and modify their values.

By default the content object of an NSObjectController instance is NSMutableDictionary. This allows a single NSObjectController instance to be used to manage many different properties referenced by key value paths. The default content object class can be changed by calling `setObjectClass` (page 1017), which subclassers must override.

Tasks

Constructors

`NSObjectController` (page 1009)
Creates and returns an empty NSObjectController.

Managing Content

`setContent` (page 1016)
Set's the receiver's content object to *`content`*.

`content` (page 1011)
Returns the receiver's content object.

`setAutomaticallyPreparesContent` (page 1015)
Sets whether the receiver creates and inserts new content objects automatically when loading from a nib file.

`automaticallyPreparesContent` (page 1010)
Returns `true` if the receiver invokes `prepareContent` (page 1014) automatically when the receiver is loaded from a nib.

`prepareContent` (page 1014)
Typically overridden by subclasses that require additional control over the creation of new objects.

Setting the Content Class

`setObjectClass` (page 1017)
Sets the object class used when creating new objects.

`objectClass` (page 1013)
Returns the class used when creating new objects.

Managing Objects

`managedObjectContext` (page 1013)
Returns the receiver's managed object context.

`newObject` (page 1013)
Creates and returns a new object of the class specified by `objectClass` (page 1013).

`addObject` (page 1010)
Sets *`object`* as the receiver's content object.

`removeObject` (page 1014)
If *`object`* is the receiver's content object, the receiver's content is set to `null`.

`add` (page 1010)
Creates a new object of the class specified by `objectClass` (page 1013) and sets it as the receiver's content object using `addObject` (page 1010).

`canAdd` (page 1011)
Returns `true` if an object can be added to the receiver using `add` (page 1010).

`remove` (page 1014)
Removes the receiver's content object using `removeObject` (page 1014).

`canRemove` (page 1011)
Returns `true` if an object can be removed from the receiver using `remove` (page 1014).

Managing Entity Names

`entityName` (page 1012)
Returns the entity name used by the receiver to create new objects.

`setEntityName` (page 1016)
Sets the receiver's entity name to *`entityName`*.

Managing Editing

`setEditable` (page 1016)
Sets whether the receiver allows adding and removing objects.

`isEditable` (page 1012)
Returns `true` if the receiver allows adding and removing objects.

Managing Fetch Predicates

`setFetchPredicate` (page 1017)
Sets the receiver's fetch predicate to *`predicate`*.

Core Data Object Contexts

`setManagedObjectContext` (page 1017)
Sets the receiver's managed object context to *`managedObjectContext`*.

`fetch` (page 1012)
Causes the receiver to fetch the data objects specified by the entity name and fetch predicate.

Obtaining Selections

`selectedObjects` (page 1015)
Returns an array of all objects to be affected by editing.

`selection` (page 1015)
Returns a proxy object representing the receiver's selection.

Validating Menu Items

`validateMenuItem` (page 1017)
Validates menu item *`anItem`*, returning `true` if it should be enabled, `false` otherwise.

Constructors

NSObjectController

Creates and returns an empty NSObjectController.

```
public NSObjectController()
```

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

Creates and returns an NSObjectController with the specified *`content`*.

```
public NSObjectController(Object content)
```

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

Instance Methods

add

Creates a new object of the class specified by `objectClass` (page 1013) and sets it as the receiver's content object using `addObject` (page 1010).

```
public void add(Object sender)
```

Discussion
The *sender* is typically the object that invoked this method.

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

See Also
`canAdd` (page 1011)
`remove` (page 1014)

addObject

Sets *object* as the receiver's content object.

```
public void addObject(Object object)
```

Discussion
If the receiver's content is bound to another object or controller through a relationship key, the relationship of the "master" object is changed.

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

See Also
`removeObject` (page 1014)

automaticallyPreparesContent

Returns `true` if the receiver invokes `prepareContent` (page 1014) automatically when the receiver is loaded from a nib.

```
public boolean automaticallyPreparesContent()
```

Discussion
If the receiver has a mananged object context set, it automatically fetches data from the managed object context using the current fetch predicate. The default is `true`..

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

See Also
`setAutomaticallyPreparesContent` (page 1015)
`prepareContent` (page 1014)

canAdd

Returns `true` if an object can be added to the receiver using `add` (page 1010).

```
public boolean canAdd()
```

Discussion
Bindings can use this method to control the enabling of user interface objects.

This property is observable using key-value observing.

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

See Also
`canRemove` (page 1011)
`add` (page 1010)

canRemove

Returns `true` if an object can be removed from the receiver using `remove` (page 1014).

```
public boolean canRemove()
```

Discussion
Bindings can use this method to control the enabling of user interface objects.

This property is observable using key-value observing.

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

See Also
`canAdd` (page 1011)
`remove` (page 1014)

content

Returns the receiver's content object.

```
public Object content()
```

Discussion
This property is observable using key-value observing.

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

See Also
`setContent` (page 1016)

entityName

Returns the entity name used by the receiver to create new objects.

```
public NSString entityName()
```

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

See Also
`setEntityName` (page 1016)

fetch

Causes the receiver to fetch the data objects specified by the entity name and fetch predicate.

```
public void fetch(Object sender)
```

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

See Also
`setFetchPredicate` (page 1017)

isEditable

Returns `true` if the receiver allows adding and removing objects.

```
public boolean isEditable()
```

Discussion
This property is observable using key-value observing.

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

See Also
`setEditable` (page 1016)

managedObjectContext

Returns the receiver's managed object context.

```
public NSManagedObjectContext managedObjectContext()
```

Availability

Available in Mac OS X v10.4 and later.

See Also

`setManagedObjectContext` (page 1017)

newObject

Creates and returns a new object of the class specified by `objectClass` (page 1013).

```
public Object newObject()
```

Discussion

This method is called when adding and inserting objects if `setAutomaticallyPreparesContent` (page 1015) is `true`.

The default implementation assumes the class returned by `objectClass` (page 1013) has a standard initialization method.

Availability

Available in Mac OS X v10.3 and later.

See Also

`setObjectClass` (page 1017)
`objectClass` (page 1013)

objectClass

Returns the class used when creating new objects.

```
public Class objectClass()
```

Discussion

The default class is NSMutableDictionary.

This property is observable using key-value observing.

Availability

Available in Mac OS X v10.3 and later.

See Also

`setObjectClass` (page 1017)
`managedObjectContext` (page 1013)

prepareContent

Typically overridden by subclasses that require additional control over the creation of new objects.

```
public void prepareContent()
```

Discussion
Subclasses that implement this method are responsible for creating the new content object and setting it as the receiver's content object. This method is only called if `setAutomaticallyPreparesContent` (page 1015) has been set to `true`.

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

See Also
`automaticallyPreparesContent` (page 1010)
`setAutomaticallyPreparesContent` (page 1015)

remove

Removes the receiver's content object using `removeObject` (page 1014).

```
public void remove(Object sender)
```

Discussion
The *`sender`* is typically the object that invoked this method.

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

See Also
`canRemove` (page 1011)
`add` (page 1010)

removeObject

If *`object`* is the receiver's content object, the receiver's content is set to `null`.

```
public void removeObject(Object object)
```

Discussion
If the receiver's content is bound to another object or controller through a relationship key, the relationship of the 'master' object is cleared.

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

See Also
`addObject` (page 1010)

selectedObjects

Returns an array of all objects to be affected by editing.

```
public NSArray selectedObjects()
```

Discussion

If the receiver supports a selection mechanism the array contains key value coding compliant proxies of the selected objects; otherwise proxies for all content objects are returned. If the receiver is a concrete instance of NSObjectController, returns an array containing the receiver's content object.

You should avoid registering for key-value observing changes for key paths that pass *through* this method, that is `selectedObjects.firstName`. Using the proxy returned by the `selection` (page 1015) method is better for performance.

This property is observable using key-value observing.

Availability

Available in Mac OS X v10.3 and later.

See Also

`selection` (page 1015)

selection

Returns a proxy object representing the receiver's selection.

```
public Object selection()
```

Discussion

If a value requested from the selection proxy using key-value coding returns multiple objects, the controller has no selection, or the proxy is not key-value coding compliant for the requested key, the appropriate marker (`MultipleValuesMarker`, `NoSelectionMarker` or `NotApplicableMarker`) is returned. Otherwise, the value of the key is returned. This proxy is fully key-value coding compliant.

This property is observable using key-value observing.

Availability

Available in Mac OS X v10.3 and later.

See Also

`selectedObjects` (page 1015)

setAutomaticallyPreparesContent

Sets whether the receiver creates and inserts new content objects automatically when loading from a nib file.

```
public void setAutomaticallyPreparesContent(boolean flag)
```

Discussion

If *`flag`* is `true` and the receiver is not using a managed object context, `prepareContent` (page 1014) is used to create the content object. If *`flag`* is `true` and a managed object context is set, the initial content is fetched from the managed object context using the current fetch predicate.

Setting *flag* to `true` is the same as checking the "Automatically Prepares Content" option in the Interface Builder controller inspector.

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

See Also
`automaticallyPreparesContent` (page 1010)
`prepareContent` (page 1014)

setContent

Set's the receiver's content object to *content*.

```
public void setContent(Object content)
```

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

See Also
`content` (page 1011)

setEditable

Sets whether the receiver allows adding and removing objects.

```
public void setEditable(boolean flag)
```

Discussion
The default is `true`.

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

See Also
`isEditable` (page 1012)

setEntityName

Sets the receiver's entity name to *entityName*.

```
public void setEntityName(NSString entityName)
```

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

See Also
`entityName` (page 1012)

setFetchPredicate

Sets the receiver’s fetch predicate to *predicate*.

```
public void setFetchPredicate(NSPredicate predicate)
```

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

See Also
`fetch` (page 1012)

setManagedObjectContext

Sets the receiver’s managed object context to *managedObjectContext*.

```
public void setManagedObjectContext(NSManagedObjectContext managedObjectContext)
```

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

See Also
`managedObjectContext` (page 1013)

setObjectClass

Sets the object class used when creating new objects.

```
public void setObjectClass(Class objectClass)
```

Discussion
NSObjectController’s default implementation assumes that instances of objectClass are initialized using a method that takes no arguments.

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

See Also
`objectClass` (page 1013)
`managedObjectContext` (page 1013)

validateMenuItem

Validates menu item *anItem*, returning `true` if it should be enabled, `false` otherwise.

```
public boolean validateMenuItem(_NSObsoleteMenuItemProtocol anItem)
```

Discussion
For example, if `canAdd` (page 1011) returns `false`, menu items with the `add` action and the receiver as the target object are disabled.

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

NSOpenPanel

Inherits from	NSSavePanel : NSPanel : NSWindow : NSResponder : NSObject
Implements	NSCoding (NSResponder)
Package:	com.apple.cocoa.application
Companion guides	Application File Management Sheet Programming Topics for Cocoa

Overview

NSOpenPanel provides the Open panel for the Cocoa 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.

Tasks

Constructors

`NSOpenPanel` (page 1021)
Creates an empty NSOpenPanel.

Obtaining

`openPanel` (page 1022)
Creates and returns a new NSOpenPanel object.

Running the Panel

`beginForDirectory` (page 1023)
Presents a modeless Open panel.

`beginSheetForDirectory` (page 1023)
Presents a sheet Open panel on a given window, *`docWindow`*.

`runModalInDirectory` (page 1025)
Displays the receiver and begins a modal event loop that is terminated when the user clicks either OK or Cancel, resulting in the return of `NSPanel.OKButton` or `NSPanel.CancelButton`, respectively.

`runModalForTypes` (page 1024)
Invokes the `runModalInDirectory` (page 1025) method, using `null` for both the *`filename`* and *`directory`* arguments.

Getting the User Selection

`filenames` (page 1024)
Returns an array containing the absolute paths (as String objects) of the selected files and directories.

`URLs` (page 1026)
Returns an array containing the absolute paths of the selected files and directories as URLs.

Specifying the File Types

`allowedFileTypes` (page 1022)
Returns an array of the allowed file types.

`setAllowedFileTypes` (page 1025)
Specifies the allowed file types for the receiver.

Allowing Browser Selections

`setCanChooseFiles` (page 1026)
Sets whether the user can select files in the receiver's browser.

`canChooseFiles` (page 1024)
Returns whether the receiver allows the user to choose files to open.

`setCanChooseDirectories` (page 1026)
Sets whether the user can select directories in the receiver's browser.

`canChooseDirectories` (page 1024)
Returns whether the receiver allows the user to choose directories to open.

`setResolvesAliases` (page 1026)
Sets whether the receiver resolves aliases to *`resolvesAliases`*.

`resolvesAliases` (page 1024)
Returns whether the receiver resolves aliases.

Allowing Multiple Selections

`setAllowsMultipleSelection` (page 1025)
Sets whether the user can select multiple files (and directories) at one time for opening to *`flag`*.

`allowsMultipleSelection` (page 1022)
Returns whether the receiver's browser allows the user to open multiple files (and directories) at a time.

Constructors

NSOpenPanel

Creates an empty NSOpenPanel.

```
public NSOpenPanel()
```

Creates a new NSOpenPanel.

```
public NSOpenPanel(NSRect contentRect, int styleMask, int backingType, boolean
    flag)
```

Discussion

The *contentRect* argument specifies the location and size of the panel's content area in screen coordinates. Note that the Window Server limits window position coordinates to ±16,000 and sizes to 10,000.

The *styleMask* argument specifies the panel's style. Either it can be `NSWindow.BorderlessWindowMask`, or it can contain any of the options described in NSWindow's "Constants" (page 1875), combined using the C bitwise OR operator.

Borderless windows display none of the usual peripheral elements and are generally useful only for display or caching purposes; you should normally not need to create them. Also, note that an NSWindow's style mask should include `NSWindow.TitledWindowMask` if it includes any of the others.

The *backingType* argument specifies how the drawing done in the panel is buffered by the object's window device, and possible values are described in NSWindow's "Constants" (page 1875).

The *flag* argument determines whether the window server creates a window device for the new panel immediately. If *flag* is `true`, it defers creating the window until the panel is moved onscreen. All display messages sent are postponed until the panel is created, just before it's moved onscreen. Deferring the creation of the window improves launch time and minimizes the virtual memory load on the window server.

The new panel creates an instance of NSView to be its default content view. You can replace it with your own object by using the `setContentView` (page 1858) method.

Creates a new NSOpenPanel.

```
public NSOpenPanel(NSRect contentRect, int styleMask, int backingType, boolean
    flag, NSScreen aScreen)
```

Discussion

This constructor is equivalent to `NSOpenPanel` (page 1021) except *contentRect* is specified relative to the lower-left corner of *aScreen*.

If *aScreen* is `null`, *contentRect* 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.

Static Methods

openPanel

Creates and returns a new NSOpenPanel object.

```
public static NSOpenPanel openPanel()
```

Discussion
The open panel has been initialized with default values.

Instance Methods

allowedFileTypes

Returns an array of the allowed file types.

```
public NSArray allowedFileTypes()
```

Discussion
File type strings encoding HFS file types are valid values for this attribute. A `null` return value, which is the default, indicates that all file types are allowed.

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

See Also
`setAllowedFileTypes` (page 1025)

allowsMultipleSelection

Returns whether the receiver's browser allows the user to open multiple files (and directories) at a time.

```
public boolean allowsMultipleSelection()
```

Discussion
If multiple files or directories are allowed, then the `filename` (page 1235) method—inherited from NSSavePanel—returns a non-`null` value only if one and only one file is selected. By contrast, NSOpenPanel's `filenames` (page 1024) method always returns the selected files, even if only one file is selected.

See Also
`filename` (page 1235) (NSSavePanel)
`filenames` (page 1024)
`setAllowsMultipleSelection` (page 1025)

beginForDirectory

Presents a modeless Open panel.

```
public void beginForDirectory(String path, String filename, NSArray fileTypes,
    Object modelessDelegate, NSSelector didEndSelector, Object contextInfo)
```

Discussion

Similar to `beginSheetForDirectory` (page 1023), but allows for modeless operation of the panel.

The receiver displays the files in *directory* (an absolute directory path), and allows selections of ones that match the types in *fileTypes* (an NSArray of file extensions and/or HFS file types). If *directory* is `null` the directory is the same directory used in the previous invocation of the panel. Passing `null` for *directory* is probably the best choice for most situations. If all files in a directory should be selectable in the browser, *fileTypes* should be `null`. The *filename* argument specifies a particular file in *directory* that is selected when the Open panel is presented to the user; otherwise, *filename* should be `null`. When the panel operation is ended, *didEndSelector* is invoked on the *modelessDelegate*, passing *contextInfo* as an argument. *modelessDelegate* is not the same as a delegate assigned to the panel. This delegate is temporary and the relationship only lasts until the panel is dismissed.

didEndSelector should have the following signature:

```
void openPanelDidEnd(NSOpenPanel panel, int returnCode, void contextInfo)
```

The value passed as *returnCode* will be either `NSPanel.CancelButton` or `NSPanel.OKButton`.

Availability

Available in Mac OS X v10.3 and later.

beginSheetForDirectory

Presents a sheet Open panel on a given window, *docWindow*.

```
public void beginSheetForDirectory(String directory, String filename, NSArray
    fileTypes, NSWindow docWindow, Object modalDelegate, NSSelector didEndSelector,
     Object contextInfo)
```

Discussion

The receiver displays the files in *directory* (an absolute directory path), and allows selection of ones that match the types in *fileTypes* (an NSArray of file extensions and/or HFS file types). If *directory* is `null` the directory is the same directory used in the previous invocation of the panel. Passing `null` for *directory* is probably the best choice for most situations. If all files in a directory should be selectable in the browser, *fileTypes* should be `null`. The *filename* argument specifies a particular file in *directory* that is selected when the Open panel is presented to the user; otherwise, *filename* should be `null`. When the modal session is ended, *didEndSelector* is invoked on the *modalDelegate*, passing *contextInfo* as an argument. *modalDelegate* is not the same as a delegate assigned to the panel. Modal delegates in sheets are temporary and the relationship only lasts until the sheet is dismissed.

didEndSelector should have the following signature:

```
public void openPanelDidEnd (NSOpenPanel sheet, int returnCode,  Object
contextInfo)
```

The value passed as *returnCode* will be either `NSPanel.CancelButton` or `NSPanel.OKButton`.

canChooseDirectories

Returns whether the receiver allows the user to choose directories to open.

```
public boolean canChooseDirectories()
```

See Also

setCanChooseDirectories (page 1026)

canChooseFiles

Returns whether the receiver allows the user to choose files to open.

```
public boolean canChooseFiles()
```

See Also

setCanChooseFiles (page 1026)

filenames

Returns an array containing the absolute paths (as String objects) of the selected files and directories.

```
public NSArray filenames()
```

Discussion

If multiple selections aren't allowed, the array contains a single name. The `filenames` method is preferable over NSSavePanel's filename (page 1235) to get the name or names of files and directories that the user has selected.

See Also

URLs (page 1026)

resolvesAliases

Returns whether the receiver resolves aliases.

```
public boolean resolvesAliases()
```

Discussion

If `true`, the effect is that dropping an alias on the receiver or asking for filenames or URLs returns the resolved aliases. The default is `true`.

See Also

setResolvesAliases (page 1026)

runModalForTypes

Invokes the runModalInDirectory (page 1025) method, using `null` for both the *filename* and *directory* arguments.

```
public int runModalForTypes(NSArray fileTypes)
```

Discussion
See the description of `runModalInDirectory` (page 1025) for details. The *fileTypes* argument is an NSArray containing the extensions of files to be selectable in the browser. Returns `NSPanel.OKButton` (if the user clicks the OK button) or `NSPanel.CancelButton` (if the user clicks the Cancel button).

runModalInDirectory

Displays the receiver and begins a modal event loop that is terminated when the user clicks either OK or Cancel, resulting in the return of `NSPanel.OKButton` or `NSPanel.CancelButton`, respectively.

```
public int runModalInDirectory(String directory, String filename, NSArray fileTypes)
```

Discussion
The receiver displays the files in *directory* (an absolute directory path), and allows selection of ones that match the types in *fileTypes* (an NSArray of file extensions and/or HFS file types). If *directory* is `null` the directory is the same directory used in the previous invocation of the panel. Passing `null` for *directory* is probably the best choice for most situations. If all files in a directory should be selectable in the browser, *fileTypes* should be `null`. You can control whether directories and files appear in the browser with the `setCanChooseDirectories` (page 1026) and `setCanChooseFiles` (page 1026) methods. The *filename* argument specifies a particular file in *directory* that is selected when the Open panel is presented to the user; otherwise, *filename* should be `null`.

If *window* is not `null`, the Open panel slides down as a sheet running as a document modal window. If *window* is `null`, the behavior defaults to a standalone modal panel.

```
public int runModalInDirectory(String directory, String filename, NSArray fileTypes,
     NSWindow window)
```

See Also
`runModalForTypes` (page 1024)

setAllowedFileTypes

Specifies the allowed file types for the receiver.

```
public void setAllowedFileTypes(NSArray types)
```

Discussion
types may not be empty. The items in *types* should not include the period that begins the extension. File type strings encoding HFS file types are valid values. Pass `null`, to allow any file type, which is the default.

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

See Also
`allowedFileTypes` (page 1022)

setAllowsMultipleSelection

Sets whether the user can select multiple files (and directories) at one time for opening to *flag*.

```
public void setAllowsMultipleSelection(boolean flag)
```

See Also
`allowsMultipleSelection` (page 1022)

setCanChooseDirectories

Sets whether the user can select directories in the receiver's browser.

```
public void setCanChooseDirectories(boolean flag)
```

Discussion
When a directory is selected, the OK button is enabled only if *flag* is `true`.

See Also
`canChooseDirectories` (page 1024)

setCanChooseFiles

Sets whether the user can select files in the receiver's browser.

```
public void setCanChooseFiles(boolean flag)
```

See Also
`canChooseFiles` (page 1024)

setResolvesAliases

Sets whether the receiver resolves aliases to *resolvesAliases*.

```
public void setResolvesAliases(boolean resolvesAliases)
```

Discussion
If `true`, the effect is that dropping an alias on the receiver or asking for filenames or URLs returns the resolved aliases. Set this value to `false` to allow selection of aliases without resolving.

See Also
`resolvesAliases` (page 1024)

URLs

Returns an array containing the absolute paths of the selected files and directories as URLs.

```
public NSArray URLs()
```

Discussion
If multiple selections aren't allowed, the array contains a single name.

See Also
`filenames` (page 1024)

NSOutlineView

Inherits from	NSTableView : NSControl : NSView : NSResponder : NSObject
Implements	NSCoding (NSResponder)
Package:	com.apple.cocoa.application
Companion guides	Outline View Programming Topics for Cocoa Drag and Drop Programming Topics for Cocoa

Class at a Glance

An NSOutlineView object uses a row-and-column format to display hierarchical data that can be expanded and collapsed, such as directories and files in a file system. A user can expand and collapse rows, edit values, and resize and rearrange columns.

Principal Attributes

- Expands and collapses rows.
- Works with NSTableView.
- Requires that each item in the outline view be unique.
- Gets data from object you provide.
- Retrieves only data that needs to be displayed.
- Uses a delegate.

Commonly Used Methods

`numberOfRows` (page 1460)
Returns the number of rows in the NSOutlineView (inherited from NSTableView).

`collapseItem` (page 1032)
Causes an item to be collapsed.

`expandItem` (page 1033)
Causes an item to be expanded.

`reloadItemAndChildren` (page 1035)
Informs the NSOutlineView that data for an item has changed and needs to be retrieved and redisplayed.

Overview

An NSOutlineView is a subclass of NSTableView that lets the user expand or collapse rows that contain hierarchical data.

Also see the NSOutlineView.DataSource (page 2013) interface, which declares the methods that an NSOutlineView uses to access the contents of its data source object.

Tasks

Constructors

`NSOutlineView` (page 1032)
Creates an NSOutlineView with a zero-sized frame rectangle.

Expanding and Collapsing the Outline

`isExpandable` (page 1034)
Returns `true` if *`item`* is expandable—that is, *`item`* can contain other items.

`expandItem` (page 1033)
Expands *`item`* if *`item`* is expandable and is not already expanded; otherwise, does nothing.

`expandItemAndChildren` (page 1033)

`collapseItem` (page 1032)
Collapses *`item`* if *`item`* is expanded and expandable; otherwise does nothing.

`collapseItemAndChildren` (page 1033)

`isItemExpanded` (page 1034)
Returns `true` if *`item`* is expanded.

Redisplaying Information

`reloadItem` (page 1035)
Reloads and redisplays the data for *`item`*.

`reloadItemAndChildren` (page 1035)

Converting Between Items and Rows

`itemAtRow` (page 1034)
Returns the item associated with *`row`*.

`rowForItem` (page 1036)
: Returns the row associated with *`item`*.

Setting the Outline Column

`setOutlineTableColumn` (page 1037)
: Sets the table column in which hierarchical data is displayed to *`outlineTableColumn`*.

`outlineTableColumn` (page 1035)
: Returns the table column in which hierarchical data is displayed.

`setAutoresizesOutlineColumn` (page 1036)
: Sets whether the receiver automatically resizes its outline column when the user expands or collapses an item to *`resize`*.

`autoresizesOutlineColumn` (page 1032)
: Returns whether the receiver automatically resizes its outline column when the user expands or collapses items.

Setting the Indentation

`levelForItem` (page 1034)
: Returns the indentation level for *`item`*.

`levelForRow` (page 1035)
: Returns the indentation level for *`row`*.

`setIndentationPerLevel` (page 1037)
: Sets indentation per level, in points, to *`newIndentLevel`*.

`indentationPerLevel` (page 1034)
: Returns the current indentation per level, in points.

`setIndentationMarkerFollowsCell` (page 1037)
: Sets whether the indentation marker symbol displayed in the outline column should be indented along with the cell contents, or always displayed left-justified in the column.

`indentationMarkerFollowsCell` (page 1033)
: Returns whether the indentation marker symbol displayed in the outline column should be indented along with the cell contents, or always displayed left-justified in the column.

Persistence

`autosaveExpandedItems` (page 1032)
: Returns whether the expanded items in the receiver are automatically saved.

`setAutosaveExpandedItems` (page 1036)
: Sets whether the expanded items in the receiver are automatically saved.

Dragging and Dropping

`setDropItemAndDropChildIndex` (page 1036)
Used to “retarget” a proposed drop.

`shouldCollapseAutoExpandedItemsForDeposited` (page 1037)
Returns `true` to indicate that autoexpanded items should return to their original collapsed state.

Collapsing and expanding items

`outlineViewShouldCollapseItem` (page 1041) *delegate method*
Returns `true` to permit *`outlineView`* to collapse *`item`*, `false` to deny permission.

`outlineViewItemWillCollapse` (page 1040) *delegate method*
Invoked when *`notification`* is posted—that is, whenever the user is about to collapse an item in the outline view.

`outlineViewItemDidCollapse` (page 1039) *delegate method*
Invoked when *`notification`* is posted—that is, whenever the user collapses an item in the outline view.

`outlineViewShouldExpandItem` (page 1041) *delegate method*
Returns `true` to permit *`outlineView`* to expand *`item`*, `false` to deny permission.

`outlineViewItemWillExpand` (page 1040) *delegate method*
Invoked when *`notification`* is posted—that is, whenever the user is about to expand an item in the outline view.

`outlineViewItemDidExpand` (page 1039) *delegate method*
Invoked when *`notification`* is posted—that is, whenever the user expands an item in the outline view.

Selecting

`outlineViewShouldSelectTableColumn` (page 1041) *delegate method*
Returns `true` to permit *`outlineView`* to select *`tableColumn`*, `false` to deny permission.

`outlineViewShouldSelectItem` (page 1041) *delegate method*
Returns `true` to permit *`outlineView`* to select *`item`*, `false` to deny permission.

`selectionShouldChangeInOutlineView` (page 1042) *delegate method*
Returns `true` to permit *`outlineView`* to change its selection (typically a row being edited), `false` to deny permission.

`outlineViewSelectionIsChanging` (page 1040) *delegate method*
Invoked when *`notification`* is posted—that is, whenever the outline view’s selection changes.

`outlineViewSelectionDidChange` (page 1040) *delegate method*
Invoked when *`notification`* is posted—that is, immediately after the outline view’s selection has changed.

Displaying cells

outlineViewWillDisplayCell (page 1042) *delegate method*
Informs the delegate that *outlineView* is about to display the cell specified by *tableColumn* and *item*.

outlineViewWillDisplayOutlineCellForTableColumn (page 1042) *delegate method*
Informs the delegate that *outlineView* is about to display *cell* (the cell used to draw the expansion symbol) for the column and item specified by *tableColumn* and *item*.

outlineViewToolTipForCell (page 1042) *delegate method*
When the cursor pauses over a cell, identified by *cell*, the value returned from this method is displayed in a tooltip.

Moving and resizing columns

outlineViewColumnDidMove (page 1038) *delegate method*
Invoked when *notification* is posted—that is, whenever the user moves a column in the outline view.

outlineViewColumnDidResize (page 1038) *delegate method*
Invoked when *notification* is posted—that is, whenever the user resizes a column in the outline view.

Editing columns

outlineViewShouldEditTableColumn (page 1041) *delegate method*
Returns true to permit *outlineView* to edit the cell specified by *tableColumn* and *item*, false to deny permission.

Working with table columns

outlineViewMouseDownInHeaderOfTableColumn (page 1040) *delegate method*
Sent to the delegate whenever the mouse button is clicked in *outlineView* while the cursor is in a column header *tableColumn*.

outlineViewDidClickTableColumn (page 1038) *delegate method*
Sent at the time the mouse button subsequently goes up in *outlineView* and *tableColumn* has been "clicked" without having been dragged anywhere.

outlineViewDidDragTableColumn (page 1039) *delegate method*
Sent at the time the mouse button goes up in *outlineView* and *tableColumn* has been dragged during the time the mouse button was down.

Returning row information

outlineViewHeightOfRowForItem (page 1039) *delegate method*
Returns the height in points of the row containing *item*.

Constructors

NSOutlineView

Creates an NSOutlineView with a zero-sized frame rectangle.

```
public NSOutlineView()
```

Creates an NSOutlineView with *frameRect* as its frame rectangle.

```
public NSOutlineView(NSRect frameRect)
```

Instance Methods

autoresizesOutlineColumn

Returns whether the receiver automatically resizes its outline column when the user expands or collapses items.

```
public boolean autoresizesOutlineColumn()
```

Discussion
The outline column contains the cells with the expansion symbols and is generally the first column. The default is `true` (the outline column is automatically resized).

See Also
`setAutoresizesOutlineColumn` (page 1036)

autosaveExpandedItems

Returns whether the expanded items in the receiver are automatically saved.

```
public boolean autosaveExpandedItems()
```

Discussion
The outline view information is saved separately for each user and for each application that user uses. Note that if `autosaveName` (page 1449) returns `null`, this setting is ignored, and outline information isn't saved.

See Also
`autosaveName` (page 1449) (NSTableView)
`autosaveTableColumns` (page 1450) (NSTableView)
`setAutosaveExpandedItems` (page 1036)

collapseItem

Collapses *item* if *item* is expanded and expandable; otherwise does nothing.

```
public void collapseItem(Object item)
```

Discussion
If collapsing takes place, posts item collapse notification.

See Also
`expandItem` (page 1033)

collapseItemAndChildren

```
public void collapseItemAndChildren(Object item, boolean collapseChildren)
```

Discussion
If *collapseChildren* is set to `false`, collapses *item* only (identical to `collapseItem` (page 1032)). If *collapseChildren* is set to `true`, recursively collapses item and its children. For each item collapsed, posts item collapsed notification.

See Also
`collapseItem` (page 1032)

expandItem

Expands *item* if *item* is expandable and is not already expanded; otherwise, does nothing.

```
public void expandItem(Object item)
```

Discussion
If expanding takes place, posts item expanded notification.

See Also
`collapseItem` (page 1032)

expandItemAndChildren

```
public void expandItemAndChildren(Object item, boolean expandChildren)
```

Discussion
If *expandChildren* is set to `false`, expands *item* only (identical to `expandItem` (page 1033)). If *expandChildren* is set to `true`, recursively expands *item* and its children. For each item expanded, posts item expanded notification.

See Also
`collapseItemAndChildren` (page 1033)

indentationMarkerFollowsCell

Returns whether the indentation marker symbol displayed in the outline column should be indented along with the cell contents, or always displayed left-justified in the column.

```
public boolean indentationMarkerFollowsCell()
```

Discussion
The default is `true`, the indentation marker is indented along with the cell contents.

See Also
`setIndentationMarkerFollowsCell` (page 1037)

indentationPerLevel

Returns the current indentation per level, in points.

```
public float indentationPerLevel()
```

See Also
`setIndentationPerLevel` (page 1037)

isExpandable

Returns `true` if *`item`* is expandable—that is, *`item`* can contain other items.

```
public boolean isExpandable(Object item)
```

See Also
`expandItem` (page 1033)
`isItemExpanded` (page 1034)

isItemExpanded

Returns `true` if *`item`* is expanded.

```
public boolean isItemExpanded(Object item)
```

See Also
`expandItem` (page 1033)
`isExpandable` (page 1034)

itemAtRow

Returns the item associated with *`row`*.

```
public Object itemAtRow(int row)
```

See Also
`rowForItem` (page 1036)

levelForItem

Returns the indentation level for *`item`*.

```
public int levelForItem(Object item)
```

Discussion

The levels are zero-based—that is, the first level of displayed items is level 0. If *item* is `null` (which is the root item), –1 is returned.

See Also

`indentationPerLevel` (page 1034)

`levelForRow` (page 1035)

levelForRow

Returns the indentation level for *row*.

```
public int levelForRow(int row)
```

Discussion

The levels are zero-based—that is, the first level of displayed items is level 0. For an invalid row, –1 is returned.

See Also

`indentationPerLevel` (page 1034)

`levelForItem` (page 1034)

outlineTableColumn

Returns the table column in which hierarchical data is displayed.

```
public NSTableColumn outlineTableColumn()
```

See Also

`setOutlineTableColumn` (page 1037)

reloadItem

Reloads and redisplays the data for *item*.

```
public void reloadItem(Object item)
```

See Also

`reloadItemAndChildren` (page 1035)

reloadItemAndChildren

```
public void reloadItemAndChildren(Object item, boolean reloadChildren)
```

Discussion

If *reloadChildren* is set to `false`, reloads and redisplays the data for *item* only (identical to `reloadItem` (page 1035)). If *reloadChildren* is set to `true`, recursively reloads and redisplays the data for item and its children. It is not necessary, or efficient, to reload children if the item is not expanded.

See Also

`reloadItem` (page 1035)

rowForItem

Returns the row associated with *item*.

```
public int rowForItem(Object item)
```

Discussion
Returns –1 if *item* is `null` or cannot be found.

See Also
`itemAtRow` (page 1034)

setAutoresizesOutlineColumn

Sets whether the receiver automatically resizes its outline column when the user expands or collapses an item to *resize*.

```
public void setAutoresizesOutlineColumn(boolean resize)
```

Discussion
The outline column contains the cells with the expansion symbols and is generally the first column. The default is `true` (the outline column is automatically resized).

See Also
`autoresizesOutlineColumn` (page 1032)

setAutosaveExpandedItems

Sets whether the expanded items in the receiver are automatically saved.

```
public void setAutosaveExpandedItems(boolean flag)
```

Discussion
If *flag* is different from the current value, this method also reads in the saved information and sets the outline view's options to match.

The outline information is saved separately for each user and for each application that user uses.

If `autosaveName` (page 1449) returns `null` this setting is ignored, and expanded item information isn't saved.

Note that you can have separate settings for `autosaveExpandedItems` (page 1032) and `autosaveTableColumns` (page 1450), so you could, for example, save expanded item information, but not table column positions.

See Also
`autosaveExpandedItems` (page 1032)
`setAutosaveTableColumns` (page 1469) (NSTableView)

setDropItemAndDropChildIndex

Used to "retarget" a proposed drop.

```
public void setDropItemAndDropChildIndex(Object item, int index)
```

Discussion

For example, to specify a drop on an item I, one would specify `item` as I and `index` as `DropOnItemIndex`. To specify a drop between child 2 and 3 of item I, one would specify `item` as I and `index` as 3 (children are a zero-based index). To specify a drop on an unexpandable item I, one would specify `item` as I and `index` as `DropOnItemIndex`.

setIndentationMarkerFollowsCell

Sets whether the indentation marker symbol displayed in the outline column should be indented along with the cell contents, or always displayed left-justified in the column.

```
public void setIndentationMarkerFollowsCell(boolean drawInCell)
```

Discussion

The default is `true`, the indentation marker is indented along with the cell contents.

See Also

`indentationMarkerFollowsCell` (page 1033)

setIndentationPerLevel

Sets indentation per level, in points, to *newIndentLevel*.

```
public void setIndentationPerLevel(float newIndentLevel)
```

See Also

`indentationPerLevel` (page 1034)

setOutlineTableColumn

Sets the table column in which hierarchical data is displayed to *outlineTableColumn*.

```
public void setOutlineTableColumn(NSTableColumn outlineTableColumn)
```

See Also

`outlineTableColumn` (page 1035)

shouldCollapseAutoExpandedItemsForDeposited

Returns `true` to indicate that autoexpanded items should return to their original collapsed state.

```
public boolean shouldCollapseAutoExpandedItemsForDeposited(boolean deposited)
```

Discussion

Override this method to provide custom behavior. *deposited* tells whether or not the drop terminated due to a successful drop.

This method is called in a variety of situations. For example, it is sent shortly after `outlineViewAcceptDrop` (page 2014) is processed and also if the drag exits the outline view (exiting the view is treated the same as a failed drop).

Constants

The following constant is provided by NSOutlineView:

Constant	Description
`DropOnItemIndex`	May be used as a valid child index of a drop target item. In this case, the drop will happen directly on the target item.

Delegate Methods

outlineViewColumnDidMove

Invoked when *notification* is posted—that is, whenever the user moves a column in the outline view.

```
public abstract void outlineViewColumnDidMove(NSNotification notification)
```

Discussion
This method is invoked as a result of posting an `OutlineViewColumnDidMoveNotification` (page 1043).

outlineViewColumnDidResize

Invoked when *notification* is posted—that is, whenever the user resizes a column in the outline view.

```
public abstract void outlineViewColumnDidResize(NSNotification notification)
```

Discussion
This method is invoked as a result of posting an `OutlineViewColumnDidResizeNotification` (page 1043).

outlineViewDidClickTableColumn

Sent at the time the mouse button subsequently goes up in *outlineView* and *tableColumn* has been “clicked” without having been dragged anywhere.

```
public abstract void outlineViewDidClickTableColumn(NSOutlineView outlineView,
    NSTableColumn tableColumn)
```

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

outlineViewDidDragTableColumn

Sent at the time the mouse button goes up in *outlineView* and *tableColumn* has been dragged during the time the mouse button was down.

```
public abstract void outlineViewDidDragTableColumn(NSOutlineView outlineView,
    NSTableColumn tableColumn)
```

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

outlineViewHeightOfRowForItem

Returns the height in points of the row containing *item*.

```
public abstract float outlineViewHeightOfRowForItem(NSOutlineView outlineView,
    Object item)
```

Discussion
Values returned by this method should not include intercell spacing and must be greater than `0`. Implement this method to support an outline view with varying row heights.

Performance Considerations: For large tables in particular, you should make sure that this method is efficient. NSTableView may cache the values this method returns. So if you would like to change a row's height make sure to invalidate the row height by calling -noteHeightOfRowsWithIndexesChanged:. NSTableView automatically invalidates its entire row height cache in -reloadData, and -noteNumberOfRowsChanged..

Availability
This method is available in Mac OS X 10.4 and later.

outlineViewItemDidCollapse

Invoked when *notification* is posted—that is, whenever the user collapses an item in the outline view.

```
public abstract void outlineViewItemDidCollapse(NSNotification notification)
```

Discussion
This method is invoked as a result of posting an `OutlineViewItemDidCollapseNotification` (page 1043).

outlineViewItemDidExpand

Invoked when *notification* is posted—that is, whenever the user expands an item in the outline view.

```
public abstract void outlineViewItemDidExpand(NSNotification notification)
```

Discussion
This method is invoked as a result of posting an `OutlineViewItemDidExpandNotification` (page 1043).

outlineViewItemWillCollapse

Invoked when *notification* is posted—that is, whenever the user is about to collapse an item in the outline view.

```
public abstract void outlineViewItemWillCollapse(NSNotification notification)
```

Discussion
This method is invoked as a result of posting an OutlineViewItemWillCollapseNotification (page 1044).

outlineViewItemWillExpand

Invoked when *notification* is posted—that is, whenever the user is about to expand an item in the outline view.

```
public abstract void outlineViewItemWillExpand(NSNotification notification)
```

Discussion
This method is invoked as a result of posting an OutlineViewItemWillExpandNotification (page 1044).

outlineViewMouseDownInHeaderOfTableColumn

Sent to the delegate whenever the mouse button is clicked in *outlineView* while the cursor is in a column header *tableColumn*.

```
public abstract void outlineViewMouseDownInHeaderOfTableColumn(NSOutlineView
    outlineView, NSTableColumn tableColumn)
```

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

outlineViewSelectionDidChange

Invoked when *notification* is posted—that is, immediately after the outline view's selection has changed.

```
public abstract void outlineViewSelectionDidChange(NSNotification notification)
```

Discussion
This method is invoked as a result of posting an OutlineViewSelectionDidChangeNotification (page 1044).

outlineViewSelectionIsChanging

Invoked when *notification* is posted—that is, whenever the outline view's selection changes.

```
public abstract void outlineViewSelectionIsChanging(NSNotification notification)
```

Discussion
This method is invoked as a result of posting an OutlineViewSelectionIsChangingNotification (page 1044).

outlineViewShouldCollapseItem

Returns `true` to permit *`outlineView`* to collapse *`item`*, `false` to deny permission.

```
public abstract boolean outlineViewShouldCollapseItem(NSOutlineView outlineView,
    Object item)
```

Discussion
The delegate can implement this method to disallow collapsing of specific items.

outlineViewShouldEditTableColumn

Returns `true` to permit *`outlineView`* to edit the cell specified by *`tableColumn`* and *`item`*, `false` to deny permission.

```
public abstract boolean outlineViewShouldEditTableColumn(NSOutlineView outlineView,
     NSTableColumn tableColumn, Object item)
```

Discussion
The delegate can implement this method to disallow editing of specific cells.

outlineViewShouldExpandItem

Returns `true` to permit *`outlineView`* to expand *`item`*, `false` to deny permission.

```
public abstract boolean outlineViewShouldExpandItem(NSOutlineView outlineView,
    Object item)
```

Discussion
The delegate can implement this method to disallow expanding of specific items.

outlineViewShouldSelectItem

Returns `true` to permit *`outlineView`* to select *`item`*, `false` to deny permission.

```
public abstract boolean outlineViewShouldSelectItem(NSOutlineView outlineView,
    Object item)
```

Discussion
The delegate can implement this method to disallow selection of particular items.

outlineViewShouldSelectTableColumn

Returns `true` to permit *`outlineView`* to select *`tableColumn`*, `false` to deny permission.

```
public abstract boolean outlineViewShouldSelectTableColumn(NSOutlineView outlineView,
     NSTableColumn tableColumn)
```

Discussion
The delegate can implement this method to disallow selection of specific columns.

outlineViewToolTipForCell

When the cursor pauses over a cell, identified by *cell*, the value returned from this method is displayed in a tooltip.

```
public String outlineViewToolTipForCell(NSOutlineView ov, NSCell cell, NSMutableRect
     rect, NSTableColumn tc, Object item, NSPoint mouseLocation)
```

Discussion

point represents the current mouse location in view coordinates. If you don't want a tooltip at that location, return null or the empty string. On entry, *rect* represents the proposed active area of the tooltip. By default, *rect* is computed as `[cell drawingRectForBounds:cellFrame]`. To control the default active area, you can modify the *rect* parameter.

Availability

This method is available in Mac OS X 10.4 and later.

outlineViewWillDisplayCell

Informs the delegate that *outlineView* is about to display the cell specified by *tableColumn* and *item*.

```
public abstract void outlineViewWillDisplayCell(NSOutlineView outlineView, Object
     cell, NSTableColumn tableColumn, Object item)
```

Discussion

The delegate can modify *cell* to alter its display attributes—for example, making uneditable values display in italic or gray text.

outlineViewWillDisplayOutlineCellForTableColumn

Informs the delegate that *outlineView* is about to display *cell* (the cell used to draw the expansion symbol) for the column and item specified by *tableColumn* and *item*.

```
public abstract void outlineViewWillDisplayOutlineCellForTableColumn(NSOutlineView
     outlineView, Object cell, NSTableColumn tableColumn, Object item)
```

Discussion

The delegate can modify *cell* to alter its display attributes.

selectionShouldChangeInOutlineView

Returns `true` to permit *outlineView* to change its selection (typically a row being edited), `false` to deny permission.

```
public abstract boolean selectionShouldChangeInOutlineView(NSOutlineView outlineView)
```

Discussion

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. The delegate can implement this method for complex validation of edited rows based on the values of any of their cells.

Notifications

OutlineViewColumnDidMoveNotification

Posted whenever a column is moved by user action in an NSOutlineView.

The notification object is the NSOutlineView in which a column moved. The `userInfo` dictionary contains the following information:

Key	Value
`"NSOldColumn"`	The integer value of the column's original index
`"NSNewColumn"`	The integer value of the column's present index

See Also
`moveColumnToColumn` (page 1459) (NSTableView)

OutlineViewColumnDidResizeNotification

Posted whenever a column is resized in an NSOutlineView.

The notification object is the NSOutlineView in which a column was resized. The `userInfo` dictionary contains the following information:

Key	Value
`"NSTableColumn"`	The column that was resized.
`"NSOldWidth"`	The integer value of the column's original width

OutlineViewItemDidCollapseNotification

Posted whenever an item is collapsed in an NSOutlineView.

The notification object is the NSOutlineView in which an item was collapsed. A collapsed item's children lose their status as being selected. The `userInfo` dictionary contains the following information:

Key	Value
`"NSObject"`	The item that was collapsed

OutlineViewItemDidExpandNotification

Posted whenever an item is expanded in an NSOutlineView.

The notification object is the NSOutlineView in which an item was expanded. The `userInfo` dictionary contains the following information:

Key	Value
`"NSObject"`	The item that was expanded

OutlineViewItemWillCollapseNotification

Posted before an item is collapsed (after the user clicks the arrow but before the item is collapsed).

The notification object is the NSOutlineView object that contains the item about to be collapsed. A collapsed item's children will lose their status as being selected. The *userInfo* dictionary contains the following information:

Key	Value
`"NSObject"`	The item about to be collapsed

OutlineViewItemWillExpandNotification

Posted before an item is expanded (after the user clicks the arrow but before the item is collapsed).

The notification object is the NSOutlineView that contains an item about to be expanded. The *userInfo* dictionary contains the following information:

Key	Value
`"NSObject"`	The item that is to be expanded

OutlineViewSelectionDidChangeNotification

Posted after the NSOutlineView's selection changes.

The notification object is the NSOutlineView whose selection changed. This notification does not contain a *userInfo* dictionary.

OutlineViewSelectionIsChangingNotification

Posted as the NSOutlineView's selection changes (while the mouse button is still down).

The notification object is the NSOutlineView whose selection is changing. This notification does not contain a *userInfo* dictionary.

NSPageLayout

Inherits from	NSObject
Package:	com.apple.cocoa.application
Companion guide	Printing Programming Topics for Cocoa

Overview

NSPageLayout is a panel that queries the user for information such as paper type and orientation. It is normally displayed in response to the user selecting the Page Setup menu item. You obtain an instance with the `pageLayout` (page 1046) class method. The pane can then be run as a sheet using `beginSheetWithPrintInfo` (page 1047) or modally using `runModal` (page 1048) or `runModalWithPrintInfo` (page 1049).

Tasks

Constructors

`NSPageLayout` (page 1046)
Creates an empty NSPageLayout.

Creating an NSPageLayout

`pageLayout` (page 1046)
Returns a newly created NSPageLayout object.

Running an NSPageLayout

`beginSheetWithPrintInfo` (page 1047)
Presents a page layout sheet for *`printInfo`*, document modal relative to *`docWindow`*.

`runModal` (page 1048)
Displays the receiver and begins the modal loop.

`runModalWithPrintInfo` (page 1049)
Displays the receiver and begins the modal loop.

Customizing an NSPageLayout

`accessoryView` (page 1047)
Returns the receiver's accessory view (used to customize the receiver).

`setAccessoryView` (page 1049)
Adds an NSView to the receiver.

Accessing the NSPrintInfo

`printInfo` (page 1048)
Returns the NSPrintInfo object used when the receiver is run.

`readPrintInfo` (page 1048)
Sets the receiver's values to those stored in the NSPrintInfo object used when the receiver is run.

`writePrintInfo` (page 1049)
Writes the receiver's values to the NSPrintInfo object used when the receiver is run.

Deprecated Methods

`pickedButton` (page 1047)

`pickedOrientation` (page 1047)

`pickedPaperSize` (page 1048)

`pickedUnits` (page 1048)

Constructors

NSPageLayout

Creates an empty NSPageLayout.

```
public NSPageLayout()
```

Static Methods

pageLayout

Returns a newly created NSPageLayout object.

```
public static NSPageLayout pageLayout()
```

Instance Methods

accessoryView

Returns the receiver's accessory view (used to customize the receiver).

```
public NSView accessoryView()
```

See Also
`setAccessoryView` (page 1049)

beginSheetWithPrintInfo

Presents a page layout sheet for *printInfo*, document modal relative to *docWindow*.

```
public void beginSheetWithPrintInfo(NSPrintInfo printInfo, NSWindow docWindow,
    Object delegate, NSSelector didEndSelector, Object contextInfo)
```

Discussion
When the modal session ends, if neither *delegate* nor *didEndSelector* is `null`, *didEndSelector* is invoked on *delegate*, passing *contextInfo*, among others, as an argument.

The *didEndSelector* argument must have the same signature as:

```
public void pageLayoutDidEnd (NSPageLayout pageLayout, int returnCode,  void
contextInfo)
```

The value passed as *returnCode* will be either `PLCancelButton` or `PLOKButton`.

pickedButton

```
public void pickedButton(Object sender)
```

Discussion
This method has been deprecated.

pickedOrientation

```
public void pickedOrientation(Object sender)
```

Discussion
This method has been deprecated.

pickedPaperSize

```
public void pickedPaperSize(Object sender)
```

Discussion
This method has been deprecated.

pickedUnits

```
public void pickedUnits(Object sender)
```

Discussion
This method has been deprecated.

printInfo

Returns the NSPrintInfo object used when the receiver is run.

```
public NSPrintInfo printInfo()
```

Discussion
The NSPrintInfo object is set using the `beginSheetWithPrintInfo` (page 1047) or `runModalWithPrintInfo` (page 1049) method. The shared NSPrintInfo object is used if the receiver is run using `runModal` (page 1048).

See Also
`readPrintInfo` (page 1048)
`writePrintInfo` (page 1049)

readPrintInfo

Sets the receiver's values to those stored in the NSPrintInfo object used when the receiver is run.

```
public void readPrintInfo()
```

Discussion
Do not invoke this method directly; it is invoked automatically before the receiver is displayed.

See Also
`printInfo` (page 1048)
`writePrintInfo` (page 1049)
`runModal` (page 1048)
`runModalWithPrintInfo` (page 1049)

runModal

Displays the receiver and begins the modal loop.

```
public int runModal()
```

Discussion

The receiver's values are recorded in the shared NSPrintInfo object. Returns `PLCancelButton` if the user clicks the Cancel button; otherwise returns `PLOKButton`.

See Also

`pickedButton` (page 1047)
`runModalWithPrintInfo` (page 1049)

runModalWithPrintInfo

Displays the receiver and begins the modal loop.

```
public int runModalWithPrintInfo(NSPrintInfo printInfo)
```

Discussion

The receiver's values are recorded in *printInfo*. Returns `PLCancelButton` if the user clicks the Cancel button; otherwise returns `PLOKButton`.

See Also

`pickedButton` (page 1047)
`runModal` (page 1048)

setAccessoryView

Adds an NSView to the receiver.

```
public void setAccessoryView(NSView aView)
```

Discussion

Invoke this method to add a custom view containing your controls. *aView* is added to the receiver's Settings popup menu with your application's name as its menu item. The receiver is automatically resized to accommodate *aView*. This method can be invoked repeatedly to change the accessory view depending on the situation. If *aView* is `null`, then the receiver's current accessory view, if any, is removed.

See Also

`accessoryView` (page 1047)

writePrintInfo

Writes the receiver's values to the NSPrintInfo object used when the receiver is run.

```
public void writePrintInfo()
```

Discussion

Do not invoke this method directly; it is invoked automatically when the receiver is dismissed.

See Also

`printInfo` (page 1048)
`readPrintInfo` (page 1048)
`runModal` (page 1048)
`runModalWithPrintInfo` (page 1049)

NSPanel

Inherits from	NSWindow : NSResponder : NSObject
Implements	NSCoding (NSResponder)
Package:	com.apple.cocoa.application
Companion guide	Window Programming Guide for Cocoa

Overview

A panel is a special kind of window, typically serving an auxiliary function in an application.

Tasks

Constructors

NSPanel (page 1052)
Creates an empty NSPanel.

Configuring Panel Behavior

setFloatingPanel (page 1054)
Controls whether the receiver floats above normal windows.

isFloatingPanel (page 1053)
Returns true if the receiver is set to float above normal windows, false otherwise.

setBecomesKeyOnlyIfNeeded (page 1053)
Controls whether the receiver becomes the key window only when the user clicks a view object that edits text or otherwise accepts keyboard input.

becomesKeyOnlyIfNeeded (page 1053)

setWorksWhenModal (page 1054)
Controls whether the receiver receives keyboard and mouse events even when some other window is being run modally.

worksWhenModal (page 1054)
Returns true if the receiver is able to receive keyboard and mouse events even when some other window is being run modally, false otherwise.

Constructors

NSPanel

Creates an empty NSPanel.

```
public NSPanel()
```

Creates a new NSPanel.

```
public NSPanel(NSRect contentRect, int styleMask, int backingType, boolean defer)
```

Discussion
The `contentRect` argument specifies the location and size of the panel's content area in screen coordinates. Note that the window server limits window position coordinates to ±16,000 and sizes to 10,000.

The `styleMask` argument specifies the panel's style. Either it can be `NSWindow.BorderlessWindowMask`, or it can contain any of the options described in NSWindow's "Constants" (page 1875), combined using the C bitwise OR operator.

Borderless windows display none of the usual peripheral elements and are generally useful only for display or caching purposes; you should normally not need to create them. Also, note that an NSWindow's style mask should include `NSWindow.TitledWindowMask` if it includes any of the others.

The `backingType` argument specifies how the drawing done in the panel is buffered by the object's window device, and possible values are described in NSWindow's "Constants" (page 1875).

The `defer` argument determines whether the window server creates a window device for the new panel immediately. If `defer` is `true`, it defers creating the window until the panel is moved onscreen. All display messages sent are postponed until the panel 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 new panel creates an instance of NSView to be its default content view. You can replace it with your own object by using the `setContentView` (page 1858) method.

Creates a new NSPanel.

```
public NSPanel(NSRect contentRect, int styleMask, int bufferingType, boolean defer,
     NSScreen aScreen)
```

Discussion
This constructor is equivalent to the preceding one, except `contentRect` is specified relative to the lower-left corner of `aScreen`.

If `aScreen` is `null`, `contentRect` 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.

Instance Methods

becomesKeyOnlyIfNeeded

```
public boolean becomesKeyOnlyIfNeeded()
```

Discussion

Returns `true` if the receiver becomes the key window only when the user clicks a view object that needs to be first responder to receive event and action messages—for example, if it edits text or otherwise accepts keyboard input. Returns `false` if it becomes the key window whenever clicked. NSPanel by default returns `false`, indicating that panels become key as other windows do.

See Also

`setBecomesKeyOnlyIfNeeded` (page 1053)

`needsPanelToBecomeKey` (page 1761) (NSView)

isFloatingPanel

Returns `true` if the receiver is set to float above normal windows, `false` otherwise.

```
public boolean isFloatingPanel()
```

Discussion

A floating panel's window level is `NSWindow.FloatingWindowLevel`. NSPanel by default returns `false`, indicating that the receiver inhabits the normal window level.

See Also

`setFloatingPanel` (page 1054)

`level` (page 1841) (NSWindow)

setBecomesKeyOnlyIfNeeded

Controls whether the receiver becomes the key window only when the user clicks a view object that edits text or otherwise accepts keyboard input.

```
public void setBecomesKeyOnlyIfNeeded(boolean flag)
```

Discussion

If *`flag`* is `true`, the receiver becomes the key window only when keyboard input is needed; if *`flag`* is `false`, it becomes the key window whenever clicked. This behavior is not set by default. You should consider setting it only if most controls in the NSPanel aren't text fields, and if the choices that can be made by entering text can also be made in another way (such as by clicking an item in a list).

If the receiver is a non-activating panel, then it becomes key only if the hit view returns `true` from `needsPanelToBecomeKey` (page 1761). In this way, a non-activating panel can control whether it takes keyboard focus.

See Also

`becomesKeyOnlyIfNeeded` (page 1053)

`needsPanelToBecomeKey` (page 1761) (NSView)

setFloatingPanel

Controls whether the receiver floats above normal windows.

```
public void setFloatingPanel(boolean flag)
```

Discussion
If `flag` is `true`, sets the receiver's window level to `NSWindow.FloatingWindowLevel`; if `flag` is `false`, sets the receiver's window level to `NSWindow.NormalWindowLevel`. The default is `false`. It's appropriate for an NSPanel to float above other windows only if all of the following conditions are true:

- It's small enough not to obscure whatever's behind it.
- It's oriented more to the mouse than to the keyboard—that is, if it doesn't become the key window or becomes so only when needed.
- It needs to remain visible while the user works in the application's normal windows—for example, if the user must frequently move the cursor back and forth between a normal window and the panel (such as a tool palette), or if the panel gives information relevant to the user's actions in a normal window.
- It hides when the application is deactivated (the default behavior for panels).

See Also
`isFloatingPanel` (page 1053)
`setLevel` (page 1863) (NSWindow)

setWorksWhenModal

Controls whether the receiver receives keyboard and mouse events even when some other window is being run modally.

```
public void setWorksWhenModal(boolean flag)
```

Discussion
If `flag` is `true`, the application object sends events to the receiver even during a modal loop or session; if `flag` is `false`, the receiver gets no events while a modal loop or session is running. See "How Modal Windows Work" for more information on modal windows and panels.

See Also
`worksWhenModal` (page 1054)
`runModalForWindow` (page 119) (NSApplication)
`runModalSession` (page 120) (NSApplication)

worksWhenModal

Returns `true` if the receiver is able to receive keyboard and mouse events even when some other window is being run modally, `false` otherwise.

```
public boolean worksWhenModal()
```

Discussion
NSPanels by default return `false`, indicating their ineligibility for events during a modal loop or session. See "How Modal Windows Work" for more information on modal windows and panels.

See Also

setWorksWhenModal (page 1054)

runModalForWindow (page 119) (NSApplication)

runModalSession (page 120) (NSApplication)

Constants

These constants define the possible return values for such methods as the runModal... methods of NSOpenPanel, which tell which button (OK or Cancel) the user has clicked on an open panel:

Constant	Description
CancelButton	The Cancel button
OKButton	The OK button

NSPanel defines the following constants for panel styles:

Constant	Description
DocModalWindowMask	The panel is created as a modal sheet.
UtilityWindowMask	The panel is created as a floating window.
NonactivatingPanelMask	The panel can receive keyboard input without activating the owning application. Valid only for an NSPanel or its subclasses; not valid for an NSWindow.

NSParagraphStyle

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.application
Companion guide	Rulers and Paragraph Styles

Overview

NSParagraphStyle and its subclass NSMutableParagraphStyle encapsulate the paragraph or ruler attributes used by the NSAttributedString classes. Instances of these classes are often referred to as paragraph style objects or, when no confusion will result, paragraph styles.

The mutable subclass of NSParagraphStyle is NSMutableParagraphStyle (page 993).

Tasks

Constructors

`NSParagraphStyle` (page 1059)
Creates an empty NSParagraphStyle.

Creating an NSParagraphStyle

`defaultParagraphStyle` (page 1059)
Returns the default paragraph style.

Accessing Style Information

`alignment` (page 1060)
Returns the text alignment of the receiver.

`firstLineHeadIndent` (page 1061)
Returns the distance in points from the leading margin of a text container to the beginning of the paragraph's first line.

headIndent (page 1062)
Returns the distance in points from the leading margin of a text container to the beginning of lines other than the first.

tailIndent (page 1065)
Returns the distance in points from the margin of a text container to the end of lines.

tabStops (page 1064)
Returns the NSTextTab objects, sorted by location, that define the tab stops for the paragraph style.

defaultTabInterval (page 1061)
Returns the document-wide default tab interval.

lineHeightMultiple (page 1062)
Returns the line height multiple.

maximumLineHeight (page 1063)
Returns the maximum height that any line in the receiver will occupy, regardless of the font size or size of any attached graphic.

minimumLineHeight (page 1063)
Returns the minimum height that any line in the receiver will occupy, regardless of the font size or size of any attached graphic.

lineSpacing (page 1063)
Returns the space in points added between lines within the paragraph (commonly known as leading).

paragraphSpacing (page 1064)
Returns the space added at the end of the paragraph to separate it from the following paragraph.

paragraphSpacingBefore (page 1064)
Returns the distance between the paragraph's top and the beginning of its text content.

Getting Text Block and List Information

textBlocks (page 1065)
Returns an array specifying the text blocks containing the paragraph, nested from outermost to innermost.

textLists (page 1065)
Returns an array specifying the text lists containing the paragraph, nested from outermost to innermost.

Getting Line Breaking Information

lineBreakMode (page 1062)
Returns the mode that should be used to break lines when laying out the paragraph's text.

hyphenationFactor (page 1062)
Returns the paragraph's threshold for hyphenation.

tighteningFactorForTruncation (page 1065)
Returns the threshold for using tightening as an alternative to truncation.

Getting HTML Header Level

`headerLevel` (page 1061)
Specifies whether the paragraph is to be treated as a header for purposes of HTML generation.

Writing Direction

`defaultWritingDirectionForLanguage` (page 1060)
Returns the default writing direction for *languageName*.

`baseWritingDirection` (page 1060)
Returns the base writing direction for the receiver. Possible return values are described in "Constants" (page 1066).

Constructors

NSParagraphStyle

Creates an empty NSParagraphStyle.

```
public NSParagraphStyle()
```

Static Methods

defaultParagraphStyle

Returns the default paragraph style.

```
public static NSParagraphStyle defaultParagraphStyle()
```

Discussion
The default paragraph style has the following default values:

Subattribute	Default Value
Alignment	`NSText.NaturalTextAlignment`
Tab stops	12 left-aligned tabs, spaced by 28.0 points
Line break mode	`LineBreakByWordWrapping`
All others	0.0

See individual method descriptions for explanations of each subattribute.

defaultWritingDirectionForLanguage

Returns the default writing direction for *languageName*.

```
public static int defaultWritingDirectionForLanguage(String languageName)
```

Discussion
languageName is in ISO language region format. Possible return values are described in "Constants" (page 1066).

Availability
Available in Mac OS X v10.2 and later.

See Also
`baseWritingDirection` (page 1060)
`setBaseWritingDirection` (page 996) (NSMutableParagraphStyle)

Instance Methods

alignment

Returns the text alignment of the receiver.

```
public int alignment()
```

Discussion
Text alignment is one of:

```
NSText.LeftTextAlignment
NSText.RightTextAlignment
NSText.CenterTextAlignment
NSText.JustifiedTextAlignment
NSText.NaturalTextAlignment
```

Natural text alignment is realized as left or right alignment depending on the line sweep direction of the first script contained in the paragraph.

See Also
`setAlignment` (page 995) (NSMutableParagraphStyle)

baseWritingDirection

Returns the base writing direction for the receiver. Possible return values are described in "Constants" (page 1066).

```
public int baseWritingDirection()
```

Availability
Available in Mac OS X v10.2 and later.

See Also

defaultWritingDirectionForLanguage (page 1060)

setBaseWritingDirection (page 996) (NSMutableParagraphStyle)

defaultTabInterval

Returns the document-wide default tab interval.

```
public float defaultTabInterval()
```

Discussion

Tabs after the last specified in tabStops (page 1064) are placed at integer multiples of this distance (if positive). Default return value is 0.0.

Availability

Available in Mac OS X v10.3 and later.

See Also

setDefaultTabInterval (page 996) (NSMutableParagraphStyle)

firstLineHeadIndent

Returns the distance in points from the leading margin of a text container to the beginning of the paragraph's first line.

```
public float firstLineHeadIndent()
```

Discussion

This value is always nonnegative.

See Also

headIndent (page 1062)

tailIndent (page 1065)

setFirstLineHeadIndent (page 997) (NSMutableParagraphStyle)

headerLevel

Specifies whether the paragraph is to be treated as a header for purposes of HTML generation.

```
public int headerLevel()
```

Discussion

Returns 0 (the default value), if the paragraph is not a header, or from 1 through 6 if the paragraph is to be treated as a header.

Availability

Available in Mac OS X v10.4 and later.

See Also

setHeaderLevel (page 997) (NSMutableParagraphStyle)

headIndent

Returns the distance in points from the leading margin of a text container to the beginning of lines other than the first.

```
public float headIndent()
```

Discussion
This value is always nonnegative.

See Also
`firstLineHeadIndent` (page 1061)
`tailIndent` (page 1065)
`setHeadIndent` (page 997) (NSMutableParagraphStyle)

hyphenationFactor

Returns the paragraph's threshold for hyphenation.

```
public float hyphenationFactor()
```

Discussion
Valid values lie between 0.0 and 1.0 inclusive. The default value is 0.0. Hyphenation is attempted when the ratio of the text width (as broken without hyphenation) to the width of the line fragment is less than the hyphenation factor. When the paragraph's hyphenation factor is 0.0, the layout manager's hyphenation factor is used instead. When both are 0.0, hyphenation is disabled.

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

See Also
`setHyphenationFactor` (page 997) (NSMutableParagraphStyle)

lineBreakMode

Returns the mode that should be used to break lines when laying out the paragraph's text.

```
public int lineBreakMode()
```

Discussion
See "Constants" (page 1066) for a list of values.

See Also
`setLineBreakMode` (page 998) (NSMutableParagraphStyle)

lineHeightMultiple

Returns the line height multiple.

```
public float lineHeightMultiple()
```

Discussion

The natural line height of the receiver is multiplied by this factor (if positive) before being constrained by minimum and maximum line height. Default return value is 0.0.

Availability

Available in Mac OS X v10.3 and later.

See Also

`maximumLineHeight` (page 1063)
`minimumLineHeight` (page 1063)
`setLineHeightMultiple` (page 998) (NSMutableParagraphStyle)

lineSpacing

Returns the space in points added between lines within the paragraph (commonly known as leading).

```
public float lineSpacing()
```

Discussion

This value is always nonnegative.

See Also

`maximumLineHeight` (page 1063)
`minimumLineHeight` (page 1063)
`paragraphSpacing` (page 1064)
`setLineSpacing` (page 998) (NSMutableParagraphStyle)

maximumLineHeight

Returns the maximum height that any line in the receiver will occupy, regardless of the font size or size of any attached graphic.

```
public float maximumLineHeight()
```

Discussion

Glyphs and graphics exceeding this height will overlap neighboring lines; however, a maximum height of 0 implies no line height limit. This value is always nonnegative. The default value is 0.

Although this limit applies to the line itself, line spacing adds extra space between adjacent lines.

See Also

`minimumLineHeight` (page 1063)
`lineSpacing` (page 1063)
`lineHeightMultiple` (page 1062)
`setMaximumLineHeight` (page 999) (NSMutableParagraphStyle)

minimumLineHeight

Returns the minimum height that any line in the receiver will occupy, regardless of the font size or size of any attached graphic.

```
public float minimumLineHeight()
```

Discussion

This value is always nonnegative.

See Also

`maximumLineHeight` (page 1063)

`lineSpacing` (page 1063)

`lineHeightMultiple` (page 1062)

`setMinimumLineHeight` (page 999) (NSMutableParagraphStyle)

paragraphSpacing

Returns the space added at the end of the paragraph to separate it from the following paragraph.

```
public float paragraphSpacing()
```

Discussion

This value is always nonnegative. Determined by adding the previous paragraph's `paragraphSpacing` and the current paragraph's `paragraphSpacingBefore` (page 1064).

See Also

`lineSpacing` (page 1063)

`setParagraphSpacing` (page 999) (NSMutableParagraphStyle)

paragraphSpacingBefore

Returns the distance between the paragraph's top and the beginning of its text content.

```
public float paragraphSpacingBefore()
```

Discussion

Default return value is 0.0.

Availability

Available in Mac OS X v10.3 and later.

See Also

`paragraphSpacing` (page 1064)

`setParagraphSpacingBefore` (page 1000) (NSMutableParagraphStyle)

tabStops

Returns the NSTextTab objects, sorted by location, that define the tab stops for the paragraph style.

```
public NSArray tabStops()
```

See Also

`location` (page 1597) (NSTextTab)

`setTabStops` (page 1000) (NSMutableParagraphStyle)

`addTabStop` (page 995) (NSMutableParagraphStyle) removeTabStop (page 995) (NSMutableParagraphStyle)

tailIndent

Returns the distance in points from the margin of a text container to the end of lines.

```
public float tailIndent()
```

Discussion
If positive, this value is the distance from the leading margin (for example, the left margin in left-to-right text). If 0 or negative, it's the distance from the trailing margin.

For example, a paragraph style designed to fit exactly in a 2-inch wide container has a head indent of 0.0 and a tail indent of 0.0. One designed to fit with a quarter-inch margin has a head indent of 0.25 and a tail indent of –0.25.

See Also
headIndent (page 1062)
firstLineHeadIndent (page 1061)
setTailIndent (page 1000) (NSMutableParagraphStyle)

textBlocks

Returns an array specifying the text blocks containing the paragraph, nested from outermost to innermost.

```
public NSArray textBlocks()
```

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

See Also
setTextBlocks (page 1001) (NSMutableParagraphStyle)

textLists

Returns an array specifying the text lists containing the paragraph, nested from outermost to innermost.

```
public NSArray textLists()
```

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

See Also
setTextLists (page 1001) (NSMutableParagraphStyle)

tighteningFactorForTruncation

Returns the threshold for using tightening as an alternative to truncation.

```
public float tighteningFactorForTruncation()
```

Discussion
When the line break mode specifies truncation, the text system attempts to tighten intercharacter spacing as an alternative to truncation, provided that the ratio of the text width to the line fragment width does not exceed 1.0 + the tightening factor returned by this method. Otherwise the text is truncated at a location determined by the line break mode. The default value is 0.05.

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

See Also
`setTighteningFactorForTruncation` (page 1001) (NSMutableParagraphStyle)

Constants

These constants specify what happens when a line is too long for its container:

Constant	Description
`LineBreakByWordWrapping`	Wrapping occurs at word boundaries, unless the word itself doesn't fit on a single line.
`LineBreakByCharWrapping`	Wrapping occurs before the first character that doesn't fit.
`LineBreakByClipping`	Lines are simply not drawn past the edge of the text container.
`LineBreakByTruncatingHead`	Each line is displayed so that the end fits in the container and the missing text is indicated by some kind of ellipsis glyph.
`LineBreakByTruncatingTail`	Each line is displayed so that the beginning fits in the container and the missing text is indicated by some kind of ellipsis glyph.
`LineBreakByTruncatingMiddle`	Each line is displayed so that the beginning and end fit in the container and the missing text is indicated by some kind of ellipsis glyph in the middle.

These constants specify the types of tab stops:

Constant	Description
`LeftTabStopType`	A left-aligned tab stop.
`RightTabStopType`	A right-aligned tab stop.
`CenterTabStopType`	A center-aligned tab stop.
`DecimalTabStopType`	Aligns columns of numbers by the decimal point.

These constants specify the writing directions:

Constant	Description
`WritingDirectionLeftToRight`	The writing direction is left to right.
`WritingDirectionRightToLeft`	The writing direction is right to left.

NSPasteboard

Inherits from	NSObject
Package:	com.apple.cocoa.application
Companion guides	Pasteboard Programming Topics for Cocoa Drag and Drop Programming Topics for Cocoa System Services

Class at a Glance

An NSPasteboard object is an interface to a pasteboard server that allows you to transfer data between applications, as in copy, cut, and paste operations. The data can be placed in the pasteboard server in a variety of representations.

Principal Attributes

- Owners
- Change count
- Data types
- Name

`generalPasteboard` (page 1072)
Returns the general NSPasteboard.

`pasteboardWithName` (page 1073)
Returns a named NSPasteboard.

Commonly Used Methods

`types` (page 1079)
Returns an NSArray of pasteboard data types.

`declareTypes` (page 1075)
Prepares NSPasteboard to receive new data.

`dataForType` (page 1075)
Reads data from a pasteboard.

`setDataForType` (page 1077)
Writes data to a pasteboard.

`stringForType` (page 1078)
Reads a String from a pasteboard.

`setStringForType` (page 1078)
Writes a String to a pasteboard.

Overview

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, 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.

The drag pasteboard (`NSDragPboard`) is used to transfer data that is being dragged by the user.

NSPasteboard is also used to transfer data between applications and service providers listed in each application's Services menu.

Tasks

Constructors

`NSPasteboard` (page 1072)
Not implemented. Use `pasteboardWithName` (page 1073) or `pasteboardWithUniqueName` (page 1074) instead to create new pasteboards.

Creating and Releasing an NSPasteboard Object

`generalPasteboard` (page 1072)
Returns the general NSPasteboard.

`pasteboardByFilteringData` (page 1072)
Creates and returns a new pasteboard with a unique name that supplies *`data`* in as many types as possible given the available filter services.

`pasteboardByFilteringFile` (page 1073)
Creates and returns a new pasteboard with a unique name that supplies the data in *`filename`* in as many types as possible given the available filter services.

`pasteboardByFilteringTypesInPasteboard` (page 1073)
Creates and returns a new pasteboard with a unique name that supplies the data on *`pasteboard`* in as many types as possible given the available filter services.

`pasteboardWithName` (page 1073)
Returns the pasteboard for the name *`name`*.

`pasteboardWithUniqueName` (page 1074)
Creates and returns a new pasteboard with a name that is guaranteed to be unique with respect to other pasteboards on the computer.

`typesFilterableTo` (page 1074)
: Returns an array listing the types of data that can be converted to *`type`* by available filter services.

`releaseGlobally` (page 1077)
: Releases the receiver's resources in the pasteboard server.

Referring to a Pasteboard by Name

`name` (page 1076)
: Returns the receiver's name.

Writing Data

`addTypes` (page 1074)
: Adds the data types in *`newTypes`* to the receiver and declares a new owner *`newOwner`*.

`declareTypes` (page 1075)
: Prepares the receiver for a change in its contents by declaring the new types of data it will contain and a new owner.

`setDataForType` (page 1077)
: Writes *`data`* to the pasteboard server.

`setPropertyListForType` (page 1078)
: Writes *`propertyList`* to the pasteboard server.

`setStringForType` (page 1078)
: Writes *`string`* to the pasteboard server.

`writeFileContents` (page 1079)

`writeFileWrapper` (page 1079)

Determining Types

`availableTypeFromArray` (page 1074)
: Scans *`types`* and returns the first type that matches a type declared on the receiver.

`types` (page 1079)
: Returns an array of the receiver's data types.

Reading Data

`changeCount` (page 1075)
: Returns the receiver's change count.

`dataForType` (page 1075)
: Reads the *`dataType`* representation of the current contents of the receiver.

`propertyListForType` (page 1076)
: Returns a property list object using the type specified by *`dataType`*.

`readFileContentsTypeToFile` (page 1077)
: Reads data representing a file's contents from the receiver and writes it to the file *filename*.

`readFileWrapper` (page 1077)
: Reads data representing a file's contents from the receiver and returns it as a file wrapper.

`stringForType` (page 1078)
: Returns a String using the type specified by *dataType*.

Methods implemented by the owner

`pasteboardChangedOwner` (page 1081) *delegate method*
: Notifies a prior owner of the *sender* pasteboard (and owners of representations on the pasteboard) that the pasteboard has changed owners.

`pasteboardProvideDataForType` (page 1081) *delegate method*
: Implemented by the owner (previously declared in a `declareTypes` (page 1075) message) to provide promised data.

Constructors

NSPasteboard

Not implemented. Use `pasteboardWithName` (page 1073) or `pasteboardWithUniqueName` (page 1074) instead to create new pasteboards.

```
public NSPasteboard()
```

Static Methods

generalPasteboard

Returns the general NSPasteboard.

```
public static NSPasteboard generalPasteboard()
```

Discussion
Invokes `pasteboardWithName` (page 1073) to obtain the pasteboard.

pasteboardByFilteringData

Creates and returns a new pasteboard with a unique name that supplies *data* in as many types as possible given the available filter services.

```
public static NSPasteboard pasteboardByFilteringData(NSData data, String type)
```

Discussion

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.

pasteboardByFilteringFile

Creates and returns a new pasteboard with a unique name that supplies the data in *filename* in as many types as possible given the available filter services.

```
public static NSPasteboard pasteboardByFilteringFile(String filename)
```

Discussion

No filter service is invoked until the data is actually requested, so invoking this method is reasonably inexpensive.

pasteboardByFilteringTypesInPasteboard

Creates and returns a new pasteboard with a unique name that supplies the data on *pasteboard* in as many types as possible given the available filter services.

```
public static NSPasteboard pasteboardByFilteringTypesInPasteboard(NSPasteboard
    pasteboard)
```

Discussion

This process can be thought of as expanding the pasteboard, because the new pasteboard generally contains more representations of the data than *pasteboard*.

This method returns *pasteboard* if pasteboard was returned by one of the `pasteboardByFiltering...` methods, so a pasteboard cannot 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 does 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.

pasteboardWithName

Returns the pasteboard for the name *name*.

```
public static NSPasteboard pasteboardWithName(String name)
```

Discussion

A new object is created only if the application does not yet have an NSPasteboard object for the specified *name*; otherwise, the existing one is returned. To get a standard pasteboard, *name* should be one of the following variables:

```
GeneralPboard
FontPboard
RulerPboard
```

```
    FindPboard
    DragPboard
```

Other names can be assigned to create private pasteboards for other purposes.

pasteboardWithUniqueName

Creates and returns a new pasteboard with a name that is guaranteed to be unique with respect to other pasteboards on the computer.

```
public static NSPasteboard pasteboardWithUniqueName()
```

Discussion
This method is useful for applications that implement their own interprocess communication using pasteboards.

typesFilterableTo

Returns an array listing the types of data that can be converted to *type* by available filter services.

```
public static NSArray typesFilterableTo(String type)
```

Discussion
The array contains the original type.

Instance Methods

addTypes

Adds the data types in *newTypes* to the receiver and declares a new owner *newOwner*.

```
public int addTypes(NSArray newTypes, Object newOwner)
```

Discussion
This method can be useful when multiple entities (such as a combination of application and library methods) contribute data for a single copy command. This method should be invoked only after a `declareTypes` (page 1075) message has been sent for the same types. The owner for the new types may be different from the owner(s) of the previously declared types.

Returns the new change count, or 0 in case of an error.

See Also
`changeCount` (page 1075)

availableTypeFromArray

Scans *types* and returns the first type that matches a type declared on the receiver.

```
public String availableTypeFromArray(NSArray types)
```

Discussion

A `types` (page 1079) or `availableTypeFromArray` message should be sent before reading any data from an NSPasteboard.

changeCount

Returns the receiver's change count.

```
public int changeCount()
```

Discussion

The change count starts at zero when a client creates the receiver and becomes the first owner. The change count increments for reasons other than ownership changes. Basically any modification of the pasteboard increments the change count. For example, when an owner converts a promise to actual data, the change count is incremented.

See Also

`declareTypes` (page 1075)

dataForType

Reads the *`dataType`* representation of the current contents of the receiver.

```
public NSData dataForType(String dataType)
```

Discussion

`dataType` should be one of the types returned by the `types` (page 1079) method. A `types` or `availableTypeFromArray` (page 1074) message should be sent before invoking `dataForType`.

If the data is successfully read, this method returns the data. It returns `null` if the contents of the pasteboard have changed (if the change count has been incremented by a `declareTypes` (page 1075) message) since they were last checked with the `types` (page 1079) method. It also returns `null` if the pasteboard server cannot supply the data in time—for example, if the pasteboard's owner is slow in responding to a `pasteboardProvideDataForType` (page 1081) message and the interprocess communication times out. The integer seconds timeout value is obtained from `PasteboardPromiseTimeout`, or is 120 seconds if the default does not exist. All other errors throw an `PasteboardCommunicationException`.

If `null` is returned, the application should put up a panel informing the user that it was unable to carry out the paste operation.

See Also

`setDataForType` (page 1077)

declareTypes

Prepares the receiver for a change in its contents by declaring the new types of data it will contain and a new owner.

```
public int declareTypes(NSArray newTypes, Object newOwner)
```

Discussion

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` message essentially changes the contents of the receiver: It invalidates the current contents of the receiver and increments its change count.

The *newTypes* argument is an array of Strings that specifies the types the 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).

The *newOwner* argument is the object responsible for writing data to the pasteboard in all the types listed in *newTypes*. You can write the data immediately after declaring the types or wait until it is required for a paste operation. If you wait, the owner will receive a `pasteboardProvideDataForType` (page 1081) message requesting the data in a particular type when it is 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* argument can be `null` if data is provided for all types immediately. Otherwise, the owner should be an object that will not be released. It should not, for example, be the NSView that displays the data if that NSView is in a window that might be closed.

Returns the receiver's new change count.

See Also

`setStringForType` (page 1078)
`addTypes` (page 1074)
`changeCount` (page 1075)

name

Returns the receiver's name.

```
public String name()
```

See Also

`pasteboardWithName` (page 1073)

propertyListForType

Returns a property list object using the type specified by *dataType*.

```
public Object propertyListForType(String dataType)
```

Discussion

A property list is an object of NSArray, NSData, NSDictionary, or String objects—or any combination thereof.

A `types` (page 1079) or `availableTypeFromArray` (page 1074) message should be sent before invoking `propertyListForType`.

This method invokes `dataForType` (page 1075).

See Also

`setPropertyListForType` (page 1078)

readFileContentsTypeToFile

Reads data representing a file's contents from the receiver and writes it to the file *filename*.

```
public String readFileContentsTypeToFile(String type, String filename)
```

Discussion

An availableTypeFromArray (page 1074) or types (page 1079) message should be sent before invoking readFileContentsTypeToFile.

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 the extension of *filename* is substituted. If data matching *type* is not found on the NSPasteboard, data of type FileContentsPboardType is requested. Returns the name of the file the data was actually written to.

See Also

writeFileContents (page 1079)

readFileWrapper

Reads data representing a file's contents from the receiver and returns it as a file wrapper.

```
public NSFileWrapper readFileWrapper()
```

Discussion

If there is no data of type FileContentsPboardType on the receiver, this method returns null.

releaseGlobally

Releases the receiver's resources in the pasteboard server.

```
public void releaseGlobally()
```

Discussion

After this method is invoked, no other application can use the named pasteboard. A temporary, privately named pasteboard can be released this way when it is no longer needed, but a standard pasteboard should never be released globally.

setDataForType

Writes *data* to the pasteboard server.

```
public boolean setDataForType(NSData data, String dataType)
```

Discussion

dataType gives the type of data being written; it must be a type declared in the previous declareTypes (page 1075) message. *data* points to the data to be sent to the pasteboard server.

Returns true if the data is successfully written or false if ownership of the pasteboard has changed. Any other error throws a PasteboardCommunicationException.

See Also

dataForType (page 1075)

`setPropertyListForType` (page 1078)
`setStringForType` (page 1078)

setPropertyListForType

Writes *`propertyList`* to the pasteboard server.

```
public boolean setPropertyListForType(Object propertyList, String dataType)
```

Discussion

`dataType` gives the type of data being written; it must be a type declared in the previous `declareTypes` (page 1075) message. *`propertyList`* points to the data to be sent to the pasteboard server.

This method invokes `setDataForType` (page 1077) with a serialized property list parameter.

Returns `true` if the data is successfully written or `false` if ownership of the pasteboard has changed. Any other error throws a `PasteboardCommunicationException`.

See Also

`propertyListForType` (page 1076)
`setStringForType` (page 1078)

setStringForType

Writes *`string`* to the pasteboard server.

```
public boolean setStringForType(String string, String dataType)
```

Discussion

`dataType` gives the type of data being written; it must be a type declared in the previous `declareTypes` (page 1075) message. *`string`* points to the data to be sent to the pasteboard server.

This method invokes `setDataForType` (page 1077) to perform the write.

Returns `true` if the data is successfully written or `false` if ownership of the pasteboard has changed. Any other error throws a `PasteboardCommunicationException`.

See Also

`stringForType` (page 1078)
`setDataForType` (page 1077)
`setPropertyListForType` (page 1078)

stringForType

Returns a String using the type specified by *`dataType`*.

```
public String stringForType(String dataType)
```

Discussion

`types` (page 1079) or `availableTypeFromArray` (page 1074) should be sent before invoking `stringForType`.

This method invokes `dataForType` to obtain the string. If the string cannot be obtained, `stringForType` will return `null`. See `dataForType` (page 1075) for a description of what will cause `null` to be returned.

See Also

`setStringForType` (page 1078)

types

Returns an array of the receiver's data types.

```
public NSArray types()
```

Discussion

Returns an array of the types declared for the current contents of the NSPasteboard. The array is an array of Strings holding the type names. Types are listed in the order they were declared.

The `types` or `availableTypeFromArray` (page 1074) method should be sent before reading any data from the receiver.

See Also

`declareTypes` (page 1075)

`dataForType` (page 1075)

writeFileContents

```
public boolean writeFileContents(String filename)
```

Discussion

Writes the contents of the file *`filename`* to the receiver and declares the data to be of type `FileContentsPboardType` and also of a type appropriate for the file's extension, if it has one. Returns `true` if the data from *`filename`* was successfully written to the receiver and `false` otherwise.

See Also

`readFileContentsTypeToFile` (page 1077)

writeFileWrapper

```
public boolean writeFileWrapper(NSFileWrapper wrapper)
```

Discussion

Writes the serialized contents of the file wrapper *`wrapper`* to the receiver and declares the data to be of type `FileContentsPboardType` and also of a type appropriate for the file's extension, if it has one. If *`wrapper`* does not have a preferred filename, this method throws an exception. Returns `true` if it could successfully write the data from *`wrapper`* to the receiver and `false` otherwise.

Constants

NSPasteboard defines the following named pasteboards:

Constant	Description
`GeneralPboard`	The 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.
`FontPboard`	The pasteboard that holds font and character information and supports Copy Font and Paste Font commands that may be implemented in a text editor.
`RulerPboard`	The pasteboard that holds information about paragraph formats in support of the Copy Ruler and Paste Ruler commands that may be implemented in a text editor.
`FindPboard`	The 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 another search.
`DragPboard`	The pasteboard that stores data to be moved as the result of a drag operation. For additional information on working with the drag pasteboard, see *Drag and Drop Programming Topics for Cocoa*.

NSPasteboard defines the following common pasteboard data types:

Constant	Description
`ColorPboardType`	NSColor data
`FileContentsPboardType`	A representation of a file's contents
`FilenamesPboardType`	Array of Strings designating one or more filenames
`FontPboardType`	Font and character information
`HTMLPboardType`	HTML (which NSTextView can read from, but not write to)
`PDFPboardType`	PDF data
`PICTPboardType`	QuickDraw picture data
`PostScriptPboardType`	Encapsulated PostScript (EPS) code
`RulerPboardType`	Paragraph formatting information
`RTFPboardType`	Rich Text Format (RTF)
`RTFDPboardType`	RTFD formatted file contents
`StringPboardType`	String data
`TabularTextPboardType`	String containing tab-separated fields of text
`TIFFPboardType`	Tag Image File Format (TIFF)
`URLPboardType`	URL data for one file or resource
`VCardPboardType`	VCard data

Constant	Description
FilesPromisePboardType	Promised files

Delegate Methods

pasteboardChangedOwner

Notifies a prior owner of the *sender* pasteboard (and owners of representations on the pasteboard) that the pasteboard has changed owners.

```
public abstract void pasteboardChangedOwner(NSPasteboard sender)
```

Discussion

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 (page 1075) method used to declare ownership.

See Also

changeCount (page 1075)

pasteboardProvideDataForType

Implemented by the owner (previously declared in a declareTypes (page 1075) message) to provide promised data.

```
public abstract void pasteboardProvideDataForType(NSPasteboard sender, String type)
```

Discussion

The owner receives a pasteboardProvideDataForType 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 setDataForType (page 1077), setPropertyListForType (page 1078), or setStringForType (page 1078) methods.

pasteboardProvideDataForType messages may also be sent to the owner when the application is shut down through Application's terminate (page 125) method. This is the method that's 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 pasteboardProvideDataForType message is sent only if *type* data has not 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 NSPasteboard in the richest, and therefore most preferred, type at the time of a cut or copy operation, its pasteboardProvideDataForType method can simply read that data from the pasteboard, convert it to the requested type, and write it back to the pasteboard as the new type.

NSPDFImageRep

Inherits from	NSImageRep : NSObject
Implements	NSCoding (NSImageRep)
Package:	com.apple.cocoa.application
Companion guide	Drawing and Images

Overview

An NSPDFImageRep is an object that can render an image from a PDF format data stream.

Tasks

Constructors

`NSPDFImageRep` (page 1084)
Creates an empty NSPDFImageRep.

Creating an NSPDFImageRep

`imageRep` (page 1084)
Creates a new NSPDFImageRep instance and then invokes a constructor to initialize it with the contents of *`pdfData`*, a PDF format data stream.

Getting Image Data

`bounds` (page 1084)
Returns the rectangle that bounds the receiver.

`currentPage` (page 1085)
Gets the currently displayed page.

`pageCount` (page 1085)
Returns the number of pages in the receiver.

`PDFRepresentation` (page 1085)
Returns the PDF representation of the receiver.

`setCurrentPage` (page 1085)
Sets the page to display to *`page`*.

Constructors

NSPDFImageRep

Creates an empty NSPDFImageRep.

```
public NSPDFImageRep()
```

Creates a new NSPDFImageRep, with the contents of *`pdfData`*, a PDF format data stream.

```
public NSPDFImageRep(NSData pdfData)
```

Discussion
If the new image rep can't be initialized for any reason (for example, *`pdfData`* doesn't conform to the PDF file format), this method returns `null`.

Static Methods

imageRep

Creates a new NSPDFImageRep instance and then invokes a constructor to initialize it with the contents of *`pdfData`*, a PDF format data stream.

```
public static Object imageRep(NSData pdfData)
```

Discussion
If the new object can't be initialized for any reason (for example, *`pdfData`* doesn't conform to the PDF file format), this method returns `null`. Otherwise, it returns a new instance of NSPDFImageRep.

See Also
`PDFRepresentation` (page 1085)

Instance Methods

bounds

Returns the rectangle that bounds the receiver.

```
public NSRect bounds()
```

Discussion
The rectangle is obtained from the PDF format using its crop box.

currentPage

Gets the currently displayed page.

```
public int currentPage()
```

Discussion

The page index is zero-based.

See Also

`setCurrentPage` (page 1085)

pageCount

Returns the number of pages in the receiver.

```
public int pageCount()
```

PDFRepresentation

Returns the PDF representation of the receiver.

```
public NSData PDFRepresentation()
```

Discussion

The returned PDF data is a copy of the data used to create the receiver.

setCurrentPage

Sets the page to display to *page*.

```
public void setCurrentPage(int page)
```

Discussion

The page index is zero-based.

See Also

`currentPage` (page 1085)

NSPICTImageRep

Inherits from	NSImageRep : NSObject
Implements	NSCoding (NSImageRep)
Package:	com.apple.cocoa.application
Companion guide	Drawing and Images

Overview

An NSPICTImageRep is an object that can render an image from a PICT format data stream as described in the Carbon QuickDraw Manager documentation including PICT format version 1, version 2, and extended version 2 pictures.

Warning: There is no guarantee that the image will render exactly the same as it would under QuickDraw because of the differences between the display medium and QuickDraw. In particular, some transfer modes and region operations may not be supported.

Tasks

Constructors

`NSPICTImageRep` (page 1088)
Throws an exception. Use the other constructor instead.

Creating an NSPICTImageRep

`imageRep` (page 1088)
Creates a new NSPICTImageRep instance and then initializes it with the contents of *pictData*, a PICT format data stream.

Getting Image Data

`boundingBox` (page 1088)
Returns the rectangle that bounds the receiver.

`PICTRepresentation` (page 1089)
Returns the PICT representation of the receiver.

Constructors

NSPICTImageRep

Throws an exception. Use the other constructor instead.

```
public NSPICTImageRep()
```

Creates a new NSPICTImageRep, with the contents of *pictData*, a PICT format data stream.

```
public NSPICTImageRep(NSData pictData)
```

Discussion
If *pictData* is obtained directly from a PICT file or document, it contains a 512-byte header before the actual picture data starts. This constructor simply ignores that header. If the new image rep can't be initialized for any reason (for example, *pictData* doesn't conform to the PICT file format), this method returns `null`.

Static Methods

imageRep

Creates a new NSPICTImageRep instance and then initializes it with the contents of *pictData*, a PICT format data stream.

```
public static NSPICTImageRep imageRep(NSData pictData)
```

Discussion
If the new object can't be initialized for any reason (for example, *pictData* doesn't conform to the PICT file format), this method returns `null`. Otherwise, it returns a new instance of NSPICTImageRep.

See Also
`PICTRepresentation` (page 1089)

Instance Methods

boundingBox

Returns the rectangle that bounds the receiver.

```
public NSRect boundingBox()
```

Discussion

The rectangle is obtained from the PICT format data, specifically the `picFrame` field in the picture header. See the Carbon QuickDraw Manager documentation for information on the picture header.

PICTRepresentation

Returns the PICT representation of the receiver.

```
public NSData PICTRepresentation()
```

Discussion

The returned PICT data is a copy of the data minus the 512-byte header, if it is present. PICT files or documents contain a 512-byte header, so if you wish to save the returned data to a file you need to precede the data with 512 bytes (all zero) to conform to the PICT document format.

NSPopUpButton

Inherits from	NSButton : NSControl : NSView : NSResponder : NSObject
Implements	NSCoding (NSResponder)
Package:	com.apple.cocoa.application
Companion guide	Application Menu and Pop-up List Programming Topics for Cocoa

Class at a Glance

An NSPopUpButton object controls a pop-up menu or a pull-down menu from which a user can select an item.

Principal Attributes

- An NSMenu (page 909)

Interface Builder
Use Interface Builder to add a pop-up or pull-down menu to a window or panel.

Commonly Used Methods

`selectedItem` (page 1101)
Returns the currently selected item.

`indexOfSelectedItem` (page 1097)
Returns an integer identifying the currently selected item.

`titleOfSelectedItem` (page 1104)
Returns a string identifying the currently selected item.

Overview

The NSPopUpButton class defines objects that implement the pop-up and pull-down menus of the graphical user interface.

NSPopUpButton uses NSPopUpButtonCell (page 1107) to implement its user interface.

Note that while a menu is tracking, adding, removing, or changing items on the menu is not reflected.

Tasks

Constructors

`NSPopUpButton` (page 1095)
: Creates an empty NSPopUpButton with a zero-sized frame rectangle.

Setting the Type of Menu

`setPullsDown` (page 1104)
: Controls whether the receiver behaves as a pull-down or pop-up menu.

`pullsDown` (page 1100)
: Returns `true` if the receiver is configured as a pull-down menu or `false` if it's configured as a pop-up menu.

`setAutoenablesItems` (page 1103)
: Sets whether the receiver automatically enables and disables its items every time a user event occurs.

`autoenablesItems` (page 1096)
: Returns whether the receiver automatically enables and disables its items every time a user event occurs.

Inserting and Deleting Items

`addItem` (page 1095)
: Adds an item named *`title`* to the end of the menu.

`addItemsWithTitles` (page 1095)
: Adds multiple items to the end of the menu.

`insertItemAtIndex` (page 1097)
: Inserts an item with title *`title`* at position *`index`* in the menu.

`removeAllItems` (page 1100)
: Removes all items in the receiver's item menu.

`removeItemWithTitle` (page 1101)
: Removes the first item named *`title`*.

`removeItemAtIndex` (page 1101)
: Removes the item at *`index`*.

Getting the User's Selection

`selectedItem` (page 1101)
: Returns the item last selected by the user (the item that was highlighted when the user released the mouse button).

`titleOfSelectedItem` (page 1104)
Returns the title of the item last selected by the user or the empty string if there's no such item.

`indexOfSelectedItem` (page 1097)
Returns the index of the item last selected by the user or –1 if there's no selected item.

`objectValue` (page 1100)
Returns the index of the selected item.

Setting the Current Selection

`selectItem` (page 1102)
Selects the menu item *anObject* in the pop-up menu.

`selectItemAtIndex` (page 1102)
Selects the item in the menu at *index*.

`selectItemWithTag` (page 1102)
Selects the menu item with the specified *tag* and returns `true` if the item was successfully selected.

`selectItemWithTitle` (page 1102)
Selects the first item with *title*.

`setObjectValue` (page 1103)
Attempts to select the item at an index of *object* if the receiver responds to `intValue` and *object* is a valid index.

Getting Menu Items

`menu` (page 1099)
Returns the pop-up button's associated menu.

`setMenu` (page 1103)
Sets the pop-up button's associated menu to *menu*.

`numberOfItems` (page 1100)
Returns the number of items in the menu.

`itemArray` (page 1098)
Returns the NSArray that holds the menu's items.

`itemAtIndex` (page 1098)
Returns the menu item at *index*.

`itemTitleAtIndex` (page 1098)
Returns the title of the item at *index*.

`itemTitles` (page 1099)
Returns an NSArray object that holds the titles of all of the items in the menu.

`itemWithTitle` (page 1099)
Returns the first item whose title is *title*.

`lastItem` (page 1099)
Returns the last item in the menu.

Getting the Indices of Menu Items

`indexOfItem` (page 1096)

Returns the index of menu item *anObject* in the pop-up menu or –1 if the menu item is not found.

`indexOfItemWithTag` (page 1097)

Returns the index of the first menu item in the pop-up menu that has the tag value *tag* or –1 if the item is not found.

`indexOfItemWithTitle` (page 1097)

Returns the index of the first item whose title matches *title* or –1 if no match is found.

`indexOfItemWithRepresentedObject` (page 1096)

Returns the index of the first menu item in the pop-up menu that holds the represented object *anObject*, or –1 if no menu item with this object is found.

`indexOfItemWithTargetAndAction` (page 1097)

Returns the index of the first menu item in the pop-up menu that has the target *target* and the action *actionSelector*.

Setting the Cell Edge to Pop out in Restricted Situations

`preferredEdge` (page 1100)

Returns the edge of the receiver next to which the pop-up menu is displayed under restrictive screen conditions.

`setPreferredEdge` (page 1103)

Sets the edge of the receiver next to which the pop-up menu should appear under restrictive screen conditions to *edge*.

Setting the Title

`setTitle` (page 1104)

Sets the string displayed in the receiver when the user isn't pressing the mouse button.

Setting the Image

`setImage` (page 1103)

This method has no effect.

Setting the State

`synchronizeTitleAndSelectedItem` (page 1104)

Ensures that the item being displayed by the receiver agrees with the selected item (see `indexOfSelectedItem` (page 1097)).

Constructors

NSPopUpButton

Creates an empty NSPopUpButton with a zero-sized frame rectangle.

```
public NSPopUpButton()
```

Creates an NSPopUpButton, giving it the dimensions specified by `frameRect`.

```
public NSPopUpButton(NSRect frameRect)
```

Discussion
The new button is initialized to operate as a pop-up menu.

Creates an NSPopUpButton, giving it the dimensions specified by `frameRect`.

```
public NSPopUpButton(NSRect frameRect, boolean flag)
```

Discussion
If `flag` is `true`, the new button is initialized to operate as a pull-down menu; otherwise, it operates as a pop-up menu.

Instance Methods

addItem

Adds an item named `title` to the end of the menu.

```
public void addItem(String title)
```

Discussion
If an item with the name `title` already exists in the menu, it's removed, and the new one is added. This method then calls `synchronizeTitleAndSelectedItem` (page 1104) to make sure the item displayed matches the currently selected item.

Since this method searches for duplicate items, it should not be used if you are adding an item to an already populated menu with more than a few hundred items. Add items directly to the receiver's menu instead.

See Also
`insertItemAtIndex` (page 1097)
`removeItemWithTitle` (page 1101)
`setTitle` (page 1104)

addItemsWithTitles

Adds multiple items to the end of the menu.

```
public void addItemsWithTitles(NSArray itemTitles)
```

Discussion

The titles for the new items are taken from the *itemTitles* array. The titles within *itemTitles* should be unique. If an item with a title already exists in the menu, it's removed, and the new one is added. Once the items are added, this method uses `synchronizeTitleAndSelectedItem` (page 1104) to make sure the item displayed matches the currently selected item.

Since this method searches for duplicate items, it should not be used if you are adding items to an already populated menu with more than a few hundred items. Add items directly to the receiver's menu instead.

See Also

`insertItemAtIndex` (page 1097)

`removeAllItems` (page 1100)

`removeItemWithTitle` (page 1101)

autoenablesItems

Returns whether the receiver automatically enables and disables its items every time a user event occurs.

```
public boolean autoenablesItems()
```

Discussion

Autoenabling is turned on unless you send the message `setAutoenablesItems` (page 1103) with an argument of `false` to the NSPopUpButton. See the NSMenu.MenuValidation (page 2011) interface specification for more information.

See Also

`setAutoenablesItems` (page 1103)

indexOfItem

Returns the index of menu item *anObject* in the pop-up menu or –1 if the menu item is not found.

```
public int indexOfItem(NSMenuItem anObject)
```

Discussion

This method invokes the method of the same name of its NSPopUpButtonCell.

indexOfItemWithRepresentedObject

Returns the index of the first menu item in the pop-up menu that holds the represented object *anObject*, or –1 if no menu item with this object is found.

```
public int indexOfItemWithRepresentedObject(Object anObject)
```

Discussion

Represented objects bear some direct relation to the title or image of a menu item; for example, an item entitled "100" might have a Number encapsulating that value as its represented object. This method invokes the method of the same name of its NSPopUpButtonCell.

indexOfItemWithTag

Returns the index of the first menu item in the pop-up menu that has the tag value *tag* or –1 if the item is not found.

```
public int indexOfItemWithTag(int tag)
```

Discussion

This method invokes the method of the same name of its NSPopUpButtonCell.

indexOfItemWithTargetAndAction

Returns the index of the first menu item in the pop-up menu that has the target *target* and the action *actionSelector*.

```
public int indexOfItemWithTargetAndAction(Object target, NSSelector actionSelector)
```

Discussion

If *actionSelector* is `NULL`, the index of the first menu item in the pop-up menu that has target *target* is returned. If no menu item matching the above criteria is found, –1 is returned. This method invokes the method of the same name of its NSPopUpButtonCell.

indexOfItemWithTitle

Returns the index of the first item whose title matches *title* or –1 if no match is found.

```
public int indexOfItemWithTitle(String title)
```

indexOfSelectedItem

Returns the index of the item last selected by the user or –1 if there's no selected item.

```
public int indexOfSelectedItem()
```

See Also

`selectedItem` (page 1101)

`titleOfSelectedItem` (page 1104)

insertItemAtIndex

Inserts an item with title *title* at position *index* in the menu.

```
public void insertItemAtIndex(String title, int index)
```

Discussion

index 0 indicates the top item. Once the item is inserted, this method uses `synchronizeTitleAndSelectedItem` (page 1104) to make sure the item displayed matches the currently selected item.

If an item with the name *title* already exists in the menu, it's removed, and the new one is added. This action essentially moves *title* to a new position. If you want to move an item, it's better to invoke `removeItemWithTitle` (page 1101) explicitly and then send this method.

Since this method searches for duplicate items, it should not be used if you are adding an item to an already populated menu with more than a few hundred items. Add items directly to the receiver's menu instead.

See Also

`addItem` (page 1095)
`addItemsWithTitles` (page 1095)
`indexOfItemWithTitle` (page 1097)
`removeItemWithTitle` (page 1101)

itemArray

Returns the NSArray that holds the menu's items.

```
public NSArray itemArray()
```

Discussion

Usually you access the menu's items and modify the menu by sending messages directly to the NSPopUpButton rather than accessing the array of items.

See Also

`itemAtIndex` (page 1098)
`insertItemAtIndex` (page 1097)
`removeItemAtIndex` (page 1101)

itemAtIndex

Returns the menu item at *index*.

```
public NSMenuItem itemAtIndex(int index)
```

Discussion

If there is no item at *index*, this method returns `null`.

See Also

`itemWithTitle` (page 1099)
`lastItem` (page 1099)

itemTitleAtIndex

Returns the title of the item at *index*.

```
public String itemTitleAtIndex(int index)
```

Discussion

If there is no item at *index*, this method returns an empty string.

See Also

`itemTitles` (page 1099)

itemTitles

Returns an NSArray object that holds the titles of all of the items in the menu.

```
public NSArray itemTitles()
```

Discussion

The titles appear in the order in which the items appear in the menu.

See Also

`itemTitleAtIndex` (page 1098)

`itemWithTitle` (page 1099)

`numberOfItems` (page 1100)

itemWithTitle

Returns the first item whose title is *title*.

```
public NSMenuItem itemWithTitle(String title)
```

Discussion

If there is no item with *title*, this method returns `null`.

See Also

`addItem` (page 1095)

`selectItemWithTitle` (page 1102)

`itemAtIndex` (page 1098)

`indexOfItemWithTitle` (page 1097)

lastItem

Returns the last item in the menu.

```
public NSMenuItem lastItem()
```

See Also

`itemAtIndex` (page 1098)

menu

Returns the pop-up button's associated menu.

```
public NSMenu menu()
```

numberOfItems

Returns the number of items in the menu.

```
public int numberOfItems()
```

See Also
`lastItem` (page 1099)

objectValue

Returns the index of the selected item.

```
public Object objectValue()
```

Discussion
It is equivalent to `indexOfSelectedItem` (page 1115).

See Also
`setObjectValue` (page 1103)

preferredEdge

Returns the edge of the receiver next to which the pop-up menu is displayed under restrictive screen conditions.

```
public int preferredEdge()
```

Discussion
For pull-down menus, the default behavior is to display the menu under the receiver. For most pop-up menus, NSPopUpButton attempts to show the selected item directly over the button.

See Also
`setPreferredEdge` (page 1103)

pullsDown

Returns `true` if the receiver is configured as a pull-down menu or `false` if it's configured as a pop-up menu.

```
public boolean pullsDown()
```

See Also
`setPullsDown` (page 1104)

removeAllItems

Removes all items in the receiver's item menu.

```
public void removeAllItems()
```

Discussion

This method then uses `synchronizeTitleAndSelectedItem` (page 1104) to refresh the menu.

See Also

`numberOfItems` (page 1100)

`removeItemAtIndex` (page 1101)

`removeItemWithTitle` (page 1101)

removeItemAtIndex

Removes the item at *`index`*.

```
public void removeItemAtIndex(int index)
```

Discussion

This method then uses `synchronizeTitleAndSelectedItem` (page 1104) to make sure the title displayed matches the currently selected item.

See Also

`insertItemAtIndex` (page 1097)

`removeAllItems` (page 1100)

`removeItemWithTitle` (page 1101)

removeItemWithTitle

Removes the first item named *`title`*.

```
public void removeItemWithTitle(String title)
```

Discussion

This method then uses `synchronizeTitleAndSelectedItem` (page 1104) to refresh the menu. An assertion is triggered if the string in *`title`* does not correspond to an existing menu item.

See Also

`addItem` (page 1095)

`removeAllItems` (page 1100)

`removeItemAtIndex` (page 1101)

selectedItem

Returns the item last selected by the user (the item that was highlighted when the user released the mouse button).

```
public NSMenuItem selectedItem()
```

Discussion

If there is no selected item, this method returns `null`. It is possible for a pull-down menu's selected item to be its first item.

selectItem

Selects the menu item `anObject` in the pop-up menu.

```
public void selectItem(NSMenuItem anObject)
```

Discussion

If `anObject` is `null`, all items in the menu are deselected (this is a technique for obtaining a pop-up menu with no items selected).

selectItemAtIndex

Selects the item in the menu at `index`.

```
public void selectItemAtIndex(int index)
```

Discussion

If `index` is –1, all items in the menu are deselected.

See Also

`indexOfSelectedItem` (page 1097)

selectItemWithTag

Selects the menu item with the specified `tag` and returns `true` if the item was successfully selected.

```
public boolean selectItemWithTag(int tag)
```

Discussion

If no item with the specified `tag` is found, returns `false` and leaves the menu state unchanged.

You typically assign tags to menu items from Interface Builder, but you can also assign them programmatically using the `setTag` (page 1931) method of NSMenuItem.

Availability

Available in Mac OS X v10.4 and later.

See Also

`indexOfItemWithTag` (page 1097)

selectItemWithTitle

Selects the first item with `title`.

```
public void selectItemWithTitle(String title)
```

Discussion

If `title` is `null` or contains an empty string, all items in the menu are deselected.

See Also

`indexOfItemWithTitle` (page 1097)
`addItem` (page 1095)

`setTitle` (page 1104)

setAutoenablesItems

Sets whether the receiver automatically enables and disables its items every time a user event occurs.

```
public void setAutoenablesItems(boolean flag)
```

Discussion
Autoenabling is turned on unless you specify `false` as the value for *`flag`*.

See Also
`autoenablesItems` (page 1096)

setImage

This method has no effect.

```
public void setImage(NSImage anImage)
```

Discussion
The image displayed in a pop up button cell is taken from the selected menu item (in the case of a pop up menu) or from the first menu item (in the case of a pull-down menu).

setMenu

Sets the pop-up button's associated menu to *`menu`*.

```
public void setMenu(NSMenu menu)
```

setObjectValue

Attempts to select the item at an index of *`object`* if the receiver responds to `intValue` and *`object`* is a valid index.

```
public void setObjectValue(Object object)
```

Discussion
Otherwise, the selected item is cleared.

See Also
`objectValue` (page 1100)

setPreferredEdge

Sets the edge of the receiver next to which the pop-up menu should appear under restrictive screen conditions to *`edge`*.

```
public void setPreferredEdge(int edge)
```

Discussion
For pull-down menus, the default behavior is to display the menu under the receiver. For most pop-up menus, NSPopUpButton attempts to show the selected item directly over the button.

See Also
`preferredEdge` (page 1100)

setPullsDown

Controls whether the receiver behaves as a pull-down or pop-up menu.

```
public void setPullsDown(boolean flag)
```

Discussion
If *flag* is `true`, the receiver is configured as a pull-down menu. If *flag* is `false`, the receiver is configured as a pop-up menu.

See Also
`pullsDown` (page 1100)

setTitle

Sets the string displayed in the receiver when the user isn't pressing the mouse button.

```
public void setTitle(String aString)
```

Discussion
If this menu is a pop-up menu, it changes the current item to be the item named *aString*, adding a new item by that name if it doesn't already exist. If this menu is a pull-down list, it sets its title to be *aString*.

synchronizeTitleAndSelectedItem

Ensures that the item being displayed by the receiver agrees with the selected item (see `indexOfSelectedItem` (page 1097)).

```
public void synchronizeTitleAndSelectedItem()
```

Discussion
If there's no selected item, this method selects the first item in the item menu and sets the receiver's item to match. For pull-down menus, this method makes sure that the first item is being displayed (the NSPopUpButtonCell must be set to use the selected menu item, which happens by default).

See Also
`itemArray` (page 1098)

titleOfSelectedItem

Returns the title of the item last selected by the user or the empty string if there's no such item.

```
public String titleOfSelectedItem()
```

See Also

`indexOfSelectedItem` (page 1097)

NSPopUpButtonCell

Inherits from NSMenuItemCell : NSButtonCell : NSActionCell : NSCell : NSObject

Implements NSCoding (NSCell)

Package: com.apple.cocoa.application

Companion guide Application Menu and Pop-up List Programming Topics for Cocoa

Overview

NSPopUpButtonCell defines the visual appearance of pop-up buttons that display pop-up or pull-down menus. Pop-up menus present the user with a set of choices, much the way radio buttons do, but using much less space. Pull-down menus also provide a set of choices but present the information in a slightly different way, usually to provide a set of commands from which the user can choose.

NSPopUpButtonCell implements the user interface of NSPopUpButton (page 1091).

Note that while a menu is tracking, adding, removing, or changing items on the menu is not reflected.

Tasks

Constructors

`NSPopUpButtonCell` (page 1110)
Creates an empty NSPopUpButtonCell.

Getting and Setting Attributes

`setMenu` (page 1122)
Sets the menu object to be used by this pop-up button to *`menu`*.

`menu` (page 1117)
Returns the menu object associated with the pop-up button.

`setPullsDown` (page 1123)
Sets whether the pop-up button uses a pop-up or a pull-down menu.

`pullsDown` (page 1118)
Returns `true` if the menu is a pull-down menu; otherwise returns `false`.

setAutoenablesItems (page 1121)
: Controls whether the pop-up button's menu automatically enables and disables its menu items.

autoenablesItems (page 1112)
: Returns whether the pop-up button's menu automatically enables and disables its menu items.

setPreferredEdge (page 1122)
: Sets the edge of the pop-up button to which menus are attached to *edge*.

preferredEdge (page 1117)
: Returns the preferred edge on which to attach the menu.

setUsesItemFromMenu (page 1123)
: Controls whether the pop-up button uses an item from the menu for its own purposes.

usesItemFromMenu (page 1124)
: Returns `true` if the pop-up button uses the title text of a menu item for its own title.

setAltersStateOfSelectedItem (page 1120)

altersStateOfSelectedItem (page 1112)
: Returns `true` if the receiver sets the state of the selected menu item to `NSCell.StateOn`.

setArrowPosition (page 1121)
: Sets the position of the arrow displayed on the receiver to *position*.

arrowPosition (page 1112)
: Returns the position of the arrow displayed on the receiver.

Adding and Removing Items

addItem (page 1111)
: Creates a new menu item with the specified *title* and adds it to the end of the menu.

addItemsWithTitles (page 1111)
: For each string in *itemTitles*, this method creates a new menu item and adds it to the end of the menu.

insertItemAtIndex (page 1115)
: Creates a new menu item with the specified *title* and inserts it into the array of menu items at *index*.

removeItemWithTitle (page 1118)
: Removes the first menu item with the specified title from the pop-up button's menu.

removeItemAtIndex (page 1118)
: Removes the menu item at *index* from the pop-up button's menu.

removeAllItems (page 1118)
: Removes all of the pop-up button's menu items.

Accessing the Items

itemArray (page 1115)
: Returns the array of menu items associated with the menu.

numberOfItems (page 1117)
: Returns the number of menu items in the pop-up button's menu.

`indexOfItem` (page 1113)
: Returns the index of *item*.

`indexOfItemWithTitle` (page 1114)
: Returns the index of the first menu item with the specified *title*.

`indexOfItemWithTag` (page 1114)
: Returns the index of the first menu item with the specified *tag*.

`indexOfItemWithRepresentedObject` (page 1113)
: Returns the index of the first menu item with the represented object specified by *obj*.

`indexOfItemWithTargetAndAction` (page 1114)
: Returns the index of the first menu item that invokes the specified *actionSelector* on the given *target*.

`itemAtIndex` (page 1116)
: Returns the menu item at *index*.

`itemWithTitle` (page 1116)
: Returns the first menu item whose title matches *title*, or `null` if no such item exists.

`lastItem` (page 1117)
: Returns the last menu item in the pop-up button's menu.

`setObjectValue` (page 1122)
: Attempts to select the item at an index of *object* if the receiver responds to `intValue` and *object* is a valid index.

`objectValue` (page 1117)
: Returns the index of the selected item.

Dealing with Selection

`selectItem` (page 1119)
: Makes *item* the currently selected menu item.

`selectItemAtIndex` (page 1119)
: Makes the item at *index* the current selection.

`selectItemWithTag` (page 1120)
: Makes the menu item with the specified *tag* the current selection.

`selectItemWithTitle` (page 1120)
: Makes the first menu item with the given title the currently selected item.

`setTitle` (page 1123)

`selectedItem` (page 1119)
: Returns the currently selected menu item or `null` if no menu item is selected.

`indexOfSelectedItem` (page 1115)
: Returns the index of the currently selected menu item.

`synchronizeTitleAndSelectedItem` (page 1124)
: For pop-up menus, this method sets the pop-up button's menu item to the currently selected menu item or to the first menu item if none is selected.

Title Conveniences

`itemTitleAtIndex` (page 1116)
: Returns the title of the menu item located at *index*.

`itemTitles` (page 1116)
: Returns a mutable array of strings containing the titles of this pop-up button's menu items.

`titleOfSelectedItem` (page 1124)
: Returns a string containing the title of the currently selected menu item or an empty string if no item is selected.

Setting the Image

`setImage` (page 1121)
: This method has no effect.

Handling Events and Action Messages

`attachPopUpWithFrameInView` (page 1112)
: Displays the pop-up button's menu, making adjustments as necessary to display the menu along the preferred edge of the cell if possible.

`dismissPopUp` (page 1113)
: Dismisses the pop-up button's menu by ordering its window out.

`performClickWithFrameInView` (page 1117)
: Displays the receiver's menu as a context menu over *controlView* in *frame*.

Constructors

NSPopUpButtonCell

Creates an empty NSPopUpButtonCell.

```
public NSPopUpButtonCell()
```

Creates an NSPopUpButtonCell initialized with *title*. If *title* contains a nonempty string, *title* is used to create the first menu item in the menu. This menu item is assigned the default pop-up button action that displays the menu.

```
public NSPopUpButtonCell(String title)
```

Creates an NSPopUpButtonCell initialized with *anImage*.

```
public NSPopUpButtonCell(NSImage anImage)
```

Discussion
If *anImage* is `null`, no image is set.

Creates a new button with the title *title*.

```
public NSPopUpButtonCell(String title, boolean pullDown)
```

Discussion
If *pullDown* is `true`, the menu style is a pull-down menu; otherwise the menu is a pop-up menu. If *title* contains a nonempty string, *title* is used to create the first menu item in the menu. This menu item is assigned the default pop-up button action that displays the menu. To set the action and target, use the `setAction` (page 1926) and `setTarget` (page 1932) methods of the item's corresponding NSMenuItem object.

Instance Methods

addItem

Creates a new menu item with the specified *title* and adds it to the end of the menu.

```
public void addItem(String title)
```

Discussion
If an item with the name *title* already exists in the menu, it's removed, and the new one is added. The menu item uses the pop-up button's default action and target, but you can change these using the `setAction` (page 1926) and `setTarget` (page 1932) methods of the corresponding NSMenuItem object.

Since this method searches for duplicate items, it should not be used if you are adding an item to an already populated menu with more than a few hundred items. Add items directly to the button's menu instead.

See Also
`addItemsWithTitles` (page 1111)
`setAction` (page 1926) (NSMenuItem)
`setTarget` (page 1932) (NSMenuItem)

addItemsWithTitles

For each string in *itemTitles*, this method creates a new menu item and adds it to the end of the menu.

```
public void addItemsWithTitles(NSArray itemTitles)
```

Discussion
The titles within *itemTitles* should be unique. If an item with a title already exists in the menu, it's removed, and the new one is added. The menu items use the pop-up button's default action and target, but you can change these using the `setAction` (page 1926) and `setTarget` (page 1932) methods of the corresponding NSMenuItem object.

Since this method searches for duplicate items, it should not be used if you are adding items to an already populated menu with more than a few hundred items. Add items directly to the button's menu instead.

See Also
`addItem` (page 1111)
`setAction` (page 1926) (NSMenuItem)
`setTarget` (page 1932) (NSMenuItem)

altersStateOfSelectedItem

Returns `true` if the receiver sets the state of the selected menu item to `NSCell.StateOn`.

```
public boolean altersStateOfSelectedItem()
```

Discussion

This option is usually used only by pop-up menus. You typically do not alter the state of menu items in a pull-down menu.

See Also

`selectItemAtIndex` (page 1119)

`selectItemWithTitle` (page 1120)

arrowPosition

Returns the position of the arrow displayed on the receiver.

```
public int arrowPosition()
```

Discussion

`PopUpNoArrow` means no arrow is displayed. `PopUpArrowAtCenter` means the arrow is vertically centered, pointing to the right, vertically centered. `PopUpArrowAtBottom` means the arrow is at the bottom, pointing downward.

See Also

`setArrowPosition` (page 1121)

attachPopUpWithFrameInView

Displays the pop-up button's menu, making adjustments as necessary to display the menu along the preferred edge of the cell if possible.

```
public void attachPopUpWithFrameInView(NSRect cellFrame, NSView controlView)
```

Discussion

The *`cellFrame`* parameter specifies the rectangle of the cell in the specified *`controlView`* to which the menu is to be attached. Before displaying the menu, this method sends a `PopUpButtonCellWillPopUpNotification` (page 1125) to both the *`controlView`* and this cell.

See Also

`dismissPopUp` (page 1113)

autoenablesItems

Returns whether the pop-up button's menu automatically enables and disables its menu items.

```
public boolean autoenablesItems()
```

Discussion

By default NSMenu objects do autoenable their menu items.

See Also

`setAutoenablesItems` (page 1121)

dismissPopUp

Dismisses the pop-up button's menu by ordering its window out.

```
public void dismissPopUp()
```

Discussion

If the pop-up button was not displaying its menu, this method does nothing.

See Also

`attachPopUpWithFrameInView` (page 1112)
`orderOut` (page 1845) (NSWindow)

indexOfItem

Returns the index of *item*.

```
public int indexOfItem(NSMenuItem item)
```

Discussion

If *item* is `null` or cannot be found, this method returns –1.

See Also

`indexOfItemWithRepresentedObject` (page 1113)
`indexOfItemWithTag` (page 1114)
`indexOfItemWithTargetAndAction` (page 1114)
`indexOfItemWithTitle` (page 1114)
`indexOfSelectedItem` (page 1115)

indexOfItemWithRepresentedObject

Returns the index of the first menu item with the represented object specified by *obj*.

```
public int indexOfItemWithRepresentedObject(Object obj)
```

Discussion

If *obj* is `null` or if a menu item with the given represented object cannot be found, this method returns –1.

See Also

`indexOfItem` (page 1113)
`indexOfItemWithTag` (page 1114)
`indexOfItemWithTargetAndAction` (page 1114)
`indexOfItemWithTitle` (page 1114)
`indexOfSelectedItem` (page 1115)
`representedObject` (page 1926) (NSMenuItem)
`setRepresentedObject` (page 1930) (NSMenuItem)

indexOfItemWithTag

Returns the index of the first menu item with the specified *tag*.

```
public int indexOfItemWithTag(int tag)
```

Discussion

If a menu item with *tag* cannot be found, this method returns –1.

Tags are values your application assigns to an object to identify it. You can assign tags to menu items using the `setTag` (page 1931) method of NSMenuItem.

See Also

`indexOfItem` (page 1113)
`indexOfItemWithRepresentedObject` (page 1113)
`indexOfItemWithTargetAndAction` (page 1114)
`indexOfItemWithTitle` (page 1114)
`indexOfSelectedItem` (page 1115)
`setTag` (page 1931) (NSMenuItem)

indexOfItemWithTargetAndAction

Returns the index of the first menu item that invokes the specified *actionSelector* on the given *target*.

```
public int indexOfItemWithTargetAndAction(Object target, NSSelector actionSelector)
```

Discussion

If a menu item with the given *actionSelector* and *target* cannot be found, this method returns –1.

NSPopUpButtonCell assigns a default action and target to each menu item, but you can change these values using the `setAction` (page 1926) and `setTarget` (page 1932) methods of NSMenuItem.

See Also

`indexOfItem` (page 1113)
`indexOfItemWithRepresentedObject` (page 1113)
`indexOfItemWithTag` (page 1114)
`indexOfItemWithTargetAndAction` (page 1114)
`indexOfItemWithTitle` (page 1114)
`indexOfSelectedItem` (page 1115)
`setAction` (page 1926) (NSMenuItem)
`setTarget` (page 1932) (NSMenuItem)

indexOfItemWithTitle

Returns the index of the first menu item with the specified *title*.

```
public int indexOfItemWithTitle(String title)
```

Discussion

title must not be `null`. If *title* contains a string that does not match the title of any menu item, this method returns –1.

See Also

`indexOfItem` (page 1113)
`indexOfItemWithRepresentedObject` (page 1113)
`indexOfItemWithTag` (page 1114)
`indexOfItemWithTargetAndAction` (page 1114)
`indexOfItemWithTitle` (page 1114)
`indexOfSelectedItem` (page 1115)

indexOfSelectedItem

Returns the index of the currently selected menu item.

```
public int indexOfSelectedItem()
```

Discussion

If no menu item is selected, this method returns –1.

See Also

`indexOfItem` (page 1113)
`indexOfItemWithRepresentedObject` (page 1113)
`indexOfItemWithTag` (page 1114)
`indexOfItemWithTargetAndAction` (page 1114)
`indexOfItemWithTitle` (page 1114)

insertItemAtIndex

Creates a new menu item with the specified *`title`* and inserts it into the array of menu items at *`index`*.

```
public void insertItemAtIndex(String title, int index)
```

Discussion

The value in *`index`* must represent a valid position in the array. The menu item at *`index`* and all those that follow it are shifted down one slot to make room for the new menu item.

This method assigns the pop-up button's default action and target to the new menu item. Use the menu item's `setAction` (page 1926) and `setTarget` (page 1932) methods to assign a new action and target.

Since this method searches for duplicate items, it should not be used if you are adding an item to an already populated menu with more than a few hundred items. Add items directly to the button's menu instead.

See Also

`insertObjectAtIndex` (NSMutableArray)
`setAction` (page 1926) (NSMenuItem)
`setTarget` (page 1932) (NSMenuItem)

itemArray

Returns the array of menu items associated with the menu.

```
public NSArray itemArray()
```

See Also
`itemArray` (page 918) (NSMenu)

itemAtIndex

Returns the menu item at *`index`*.

```
public NSMenuItem itemAtIndex(int index)
```

Discussion
The value in *`index`* must refer to an existing menu item.

See Also
`itemTitleAtIndex` (page 1116)
`itemAtIndex` (page 918) (NSMenu)

itemTitleAtIndex

Returns the title of the menu item located at *`index`*.

```
public String itemTitleAtIndex(int index)
```

See Also
`itemAtIndex` (page 1116)

itemTitles

Returns a mutable array of strings containing the titles of this pop-up button's menu items.

```
public NSArray itemTitles()
```

Discussion
If the menu contains separator items, the array contains an empty string ("") for each separator item.

See Also
`itemTitleAtIndex` (page 1116)

itemWithTitle

Returns the first menu item whose title matches *`title`*, or `null` if no such item exists.

```
public NSMenuItem itemWithTitle(String title)
```

See Also
`itemTitleAtIndex` (page 1116)
`itemAtIndex` (page 1116)

lastItem

Returns the last menu item in the pop-up button's menu.

```
public NSMenuItem lastItem()
```

menu

Returns the menu object associated with the pop-up button.

```
public NSMenu menu()
```

See Also

`setMenu` (page 1122)

numberOfItems

Returns the number of menu items in the pop-up button's menu.

```
public int numberOfItems()
```

See Also

`count` (NSArray)

objectValue

Returns the index of the selected item.

```
public Object objectValue()
```

Discussion

It is equivalent to `indexOfSelectedItem` (page 1115).

See Also

`setObjectValue` (page 1122)

performClickWithFrameInView

Displays the receiver's menu as a context menu over *controlView* in *frame*.

```
public void performClickWithFrameInView(NSRect frame, NSView controlView)
```

preferredEdge

Returns the preferred edge on which to attach the menu.

```
public int preferredEdge()
```

Discussion

If no edge was set using the `setPreferredEdge` (page 1122) method, this method sets the preferred edge to the bottom edge of the pop-up button and returns that value.

See Also

`setPreferredEdge` (page 1122)

pullsDown

Returns `true` if the menu is a pull-down menu; otherwise returns `false`.

```
public boolean pullsDown()
```

See Also

`setPullsDown` (page 1123)

removeAllItems

Removes all of the pop-up button's menu items.

```
public void removeAllItems()
```

See Also

`removeItemAtIndex` (page 1118)
`removeItemWithTitle` (page 1118)
`insertItemAtIndex` (page 1115)

removeItemAtIndex

Removes the menu item at *`index`* from the pop-up button's menu.

```
public void removeItemAtIndex(int index)
```

Discussion

`index` must be a valid index into the array of menu items and therefore must not be negative.

See Also

`removeAllItems` (page 1118)
`removeItemWithTitle` (page 1118)
`insertItemAtIndex` (page 1115)

removeItemWithTitle

Removes the first menu item with the specified title from the pop-up button's menu.

```
public void removeItemWithTitle(String title)
```

Discussion

An assertion is triggered if the string in *`title`* does not correspond to an existing menu item.

See Also

removeAllItems (page 1118)

removeItemAtIndex (page 1118)

insertItemAtIndex (page 1115)

selectedItem

Returns the currently selected menu item or `null` if no menu item is selected.

```
public NSMenuItem selectedItem()
```

See Also

selectItem (page 1119)

selectItemAtIndex (page 1119)

selectItemWithTitle (page 1120)

selectItem

Makes *item* the currently selected menu item.

```
public void selectItem(NSMenuItem item)
```

Discussion

If *item* is `null`, this method deselects the currently selected menu item.

By default, selecting or deselecting a menu item from a pop-up menu changes its state. Selecting a menu item from a pull-down menu does not automatically alter the state of the item. Use the `setAltersStateOfSelectedItem` (page 1120) method, passing it a value of `false`, to disassociate the current selection from the state of menu items.

See Also

selectedItem (page 1119)

selectItemAtIndex (page 1119)

selectItemWithTitle (page 1120)

setAltersStateOfSelectedItem (page 1120)

setState (page 1931) (NSMenuItem)

selectItemAtIndex

Makes the item at *index* the current selection.

```
public void selectItemAtIndex(int index)
```

Discussion

If *index* is –1, this method deselects the currently selected menu item.

By default, selecting or deselecting a menu item from a pop-up menu changes its state. Selecting a menu item from a pull-down menu does not automatically alter the state of the item. Use the `setAltersStateOfSelectedItem` (page 1120) method, passing it a value of `false`, to disassociate the current selection from the state of menu items.

Subclassers can override this method to be able to catch all select calls.

See Also

`selectedItem` (page 1119)

`selectItem` (page 1119)

`selectItemWithTitle` (page 1120)

`setAltersStateOfSelectedItem` (page 1120)

`setState` (page 1931) (NSMenuItem)

selectItemWithTag

Makes the menu item with the specified *`tag`* the current selection.

```
public boolean selectItemWithTag(int tag)
```

Discussion

Returns `true` if the item was successfully selected or `false` if it was not selected. If the specified *`tag`* could not be found, the menu state remains unchanged.

Availability

Available in Mac OS X v10.4 and later.

selectItemWithTitle

Makes the first menu item with the given title the currently selected item.

```
public void selectItemWithTitle(String title)
```

Discussion

If *`title`* is `null` or contains a string that does not match the title of any menu item, this method deselects the currently selected menu item.

By default, selecting or deselecting a menu item changes its state. Use the `setAltersStateOfSelectedItem` (page 1120) method, passing it a value of `false`, to disassociate the current selection from the state of menu items.

See Also

`selectedItem` (page 1119)

`selectItem` (page 1119)

`selectItemAtIndex` (page 1119)

`setAltersStateOfSelectedItem` (page 1120)

`setState` (page 1931) (NSMenuItem)

setAltersStateOfSelectedItem

```
public void setAltersStateOfSelectedItem(boolean flag)
```

Discussion

If *flag* is false, this method disassociates the current selection from the state of menu items. Before disassociating the selection from the menu item state, this method first sets the state of the currently selected menu item to NSCell.StateOff. If *flag* is true, this method sets the state of the currently selected menu item to NSCell.StateOn.

See Also

altersStateOfSelectedItem (page 1112)
selectedItem (page 1119)
selectItem (page 1119)
selectItemAtIndex (page 1119)
setState (page 1931) (NSMenuItem)

setArrowPosition

Sets the position of the arrow displayed on the receiver to *position*.

```
public void setArrowPosition(int position)
```

Discussion

PopUpNoArrow displays no arrow. PopUpArrowAtCenter displays the arrow centered horizontally within the cell. PopUpArrowAtBottom displays the arrow at the edge of the cell. This method works with setPreferredEdge: to determine the exact location and orientation of the arrow. For more information, see setPreferredEdge (page 1122).

This method is ignored unless the receiver is a pull-down list with a beveled border.

See Also

arrowPosition (page 1112)

setAutoenablesItems

Controls whether the pop-up button's menu automatically enables and disables its menu items.

```
public void setAutoenablesItems(boolean flag)
```

Discussion

If *flag* is true, menu items are automatically enabled and disabled. If *flag* is false, menu items are not automatically enabled or disabled.

See Also

autoenablesItems (page 1112)

setImage

This method has no effect.

```
public void setImage(NSImage anImage)
```

Discussion
The image displayed in a pop up button is taken from the selected menu item (in the case of a pop up menu) or from the first menu item (in the case of a pull-down menu).

setMenu

Sets the menu object to be used by this pop-up button to *menu*.

```
public void setMenu(NSMenu menu)
```

Discussion
If another menu was already associated with the pop-up button, this method releases the old menu. If you want to explicitly save the old menu, you should retain it before invoking this method.

See Also
`menu` (page 1117)

setObjectValue

Attempts to select the item at an index of *object* if the receiver responds to `intValue` and *object* is a valid index.

```
public void setObjectValue(Object object)
```

Discussion
Otherwise, the selected item is cleared.

See Also
`objectValue` (page 1117)

setPreferredEdge

Sets the edge of the pop-up button to which menus are attached to *edge*.

```
public void setPreferredEdge(int edge)
```

Discussion
At display time, if attaching the menu to the preferred edge would cause part of the menu to be obscured, the pop-up button may use a different edge. If no preferred edge is set, the pop-up button uses the bottom edge by default.

This method works with `setArrowPosition` (page 1121) to determine the exact location of the arrow:

- If the arrow position is `PopUpArrowAtCenter`, the arrow stays in the center of the button and this method determines which edge the arrow points to. `NSRect.MinXEdge` points to the left, `NSRect.MaxYEdge` points to the top, `NSRect.MaxXEdge` points to the right, and `NSRect.MinYEdge` points to the bottom.

- If the arrow position is PopUpArrowAtBottom, this method determines which edge the arrow is at. NSRect.MinXEdge places the arrow at the center of the left side, pointing to the left. NSRect.MinYEdge places the arrow at bottom right corner, pointing up. NSRect.MaxXEdge places the arrow at the center of the right side, pointing to the right. NSRect.MaxYEdge places the arrow at the bottom right corner, pointing down.

See Also
preferredEdge (page 1117)

setPullsDown

Sets whether the pop-up button uses a pop-up or a pull-down menu.

```
public void setPullsDown(boolean flag)
```

Discussion
If *flag* is true, the pop-up button displays a pull-down menu; otherwise the pop-up button displays a pop-up menu. This method does not change the contents of the menu; it changes only the style of the menu.

When changing the menu type to a pull-down menu, if the menu was a pop-up menu and the cell alters the state of its selected items, this method sets the state of the currently selected item to NSCell.StateOff before changing the menu type.

See Also
pullsDown (page 1118)
synchronizeTitleAndSelectedItem (page 1124)

setTitle

```
public void setTitle(String aString)
```

Discussion
For pull-down menus that get their titles from a menu item, this method simply sets the pop-up button cell's menu item to the first item in the menu. For pop-up menus, if a menu item whose title matches *aString* exists, this method makes that menu item the current selection; otherwise, it creates a new menu item with the title *aString*, adds it to the pop-up menu, and selects it.

setUsesItemFromMenu

Controls whether the pop-up button uses an item from the menu for its own purposes.

```
public void setUsesItemFromMenu(boolean flag)
```

Discussion
For pull-down menus, the pop-up button uses the first menu item as its own title if *flag* is true. For pop-up menus, the pop-up button uses the title of the currently selected menu item; if no menu item is selected, the pop-up button displays no item and is drawn empty.

See Also
usesItemFromMenu (page 1124)

synchronizeTitleAndSelectedItem

For pop-up menus, this method sets the pop-up button's menu item to the currently selected menu item or to the first menu item if none is selected.

```
public void synchronizeTitleAndSelectedItem()
```

Discussion
For pull-down menus, this method sets the item to the first menu item. If the pop-up button cell does not get its title from a menu item, this method does nothing.

If the pop-up button's menu does not contain any menu items, this method sets the pop-up button's item to `null`.

titleOfSelectedItem

Returns a string containing the title of the currently selected menu item or an empty string if no item is selected.

```
public String titleOfSelectedItem()
```

See Also
`selectItemWithTitle` (page 1120)

usesItemFromMenu

Returns `true` if the pop-up button uses the title text of a menu item for its own title.

```
public boolean usesItemFromMenu()
```

Discussion
If this option is set, pull-down menus use the title of the first menu item in the menu while pop-up menus use the title of the currently selected menu.

See Also
`setUsesItemFromMenu` (page 1123)

Constants

The following constants are defined by NSPopUpButtonCell and used by `arrowPosition` (page 1112) and `setArrowPosition` (page 1121):

Constant	Description
`PopUpNoArrow`	Does not display any arrow in the receiver.
`PopUpArrowAtCenter`	Arrow is centered vertically, pointing toward the `preferredEdge` (page 1117).
`PopUpArrowAtBottom`	Arrow is drawn at the edge of the button, pointing toward the `preferredEdge` (page 1117).

Notifications

PopUpButtonCellWillPopUpNotification

This notification is posted just before an pop-up menu is attached to its window frame. You can use this notification to lazily construct your part's menus, thus preventing unnecessary calculations until they are needed. The notification object can be either a pop-up button or its enclosed pop-up button cell. This notification does not contain a `userInfo` dictionary.

NSPrinter

Inherits from NSObject

Implements NSCoding

Package: com.apple.cocoa.application

Companion guide Printing Programming Topics for Cocoa

Overview

An NSPrinter object describes a printer's capabilities as defined in its PPD file. An NSPrinter object can be constructed by specifying either the printer name or the make and model of an available printer. You use NSPrinter to get information about printers, not to modify printer attributes or control a printing job.

Tasks

Constructors

`NSPrinter` (page 1129)
Creates an empty NSPrinter.

Creating an NSPrinter

`printerWithName` (page 1130)
Returns an NSPrinter that represents the printer with the given *name*.

`printerWithType` (page 1130)
Returns an NSPrinter object that represents the first available printer that has the make and model described by *type*.

Getting General Printer Information

`printerNames` (page 1129)
Returns the names of all available printers.

`printerTypes` (page 1129)
Returns descriptions of the makes and models of all available printers identified by `printerNames` (page 1129)

Getting Attributes

name (page 1133)
Returns the printer's name.

type (page 1135)
Returns a description of the printer's make and model.

Getting Specific Information

pageSizeForPaper (page 1133)
Returns the size of the page for the paper type *paperName*.

languageLevel (page 1132)
Returns the PostScript language level recognized by the printer.

Querying the Tables

isKeyInTable (page 1132)
Returns `true` if *key* is in *table*, `false` otherwise.

stringForKeyInTable (page 1134)
Returns the first occurrence of a value associated with *key* in *table*.

stringListForKeyInTable (page 1134)
Returns an array of strings, one for each occurrence, associated with *key* in *table*.

booleanForKeyInTable (page 1130)
Returns a `boolean` value associated with *key* in *table*.

floatForKeyInTable (page 1131)
Returns a floating-point value associated with *key* in *table*.

intForKeyInTable (page 1132)
Returns an integer value associated with *key* in *table*.

rectForKeyInTable (page 1133)
Returns the rectangle associated with *key* in *table*.

sizeForKeyInTable (page 1134)
Returns the size associated with *key* in *table*.

statusForTable (page 1134)
Returns the status of *table*.

deviceDescription (page 1131)
Returns a dictionary of keys and values describing the device.

Deprecated Methods

acceptsBinary (page 1130)
Deprecated.

domain (page 1131)

`host` (page 1131)
Deprecated.

`imageRectForPaper` (page 1131)
Deprecated.

`isColor` (page 1132)
Deprecated.

`isFontAvailable` (page 1132)
Deprecated.

`isOutputStackInReverseOrder` (page 1132)
Deprecated.

`note` (page 1133)
Deprecated.

Constructors

NSPrinter

Creates an empty NSPrinter.

```
public NSPrinter()
```

Static Methods

printerNames

Returns the names of all available printers.

```
public static NSArray printerNames()
```

Discussion
The user constructs the list of available printers with the Print Center application.

See Also
`printerTypes` (page 1129)
`name` (page 1133)

printerTypes

Returns descriptions of the makes and models of all available printers identified by `printerNames` (page 1129)

```
public static NSArray printerTypes()
```

Discussion
.

See Also

`type` (page 1135)

printerWithName

Returns an NSPrinter that represents the printer with the given *name*.

```
public static NSPrinter printerWithName(String name)
```

Discussion

Returns `null` if the specified printer is not available.

Deprecated.

```
public static NSPrinter printerWithName(String name, String domain, boolean
    includeUnavailable)
```

See Also

`printerWithType` (page 1130)

`printerNames` (page 1129)

printerWithType

Returns an NSPrinter object that represents the first available printer that has the make and model described by *type*.

```
public static NSPrinter printerWithType(String type)
```

Discussion

type is of the form returned by `printerTypes` (page 1129).

See Also

`printerWithName` (page 1130)

`type` (page 1135)

Instance Methods

acceptsBinary

Deprecated.

```
public boolean acceptsBinary()
```

booleanForKeyInTable

Returns a `boolean` value associated with *key* in *table*.

```
public boolean booleanForKeyInTable(String key, String table)
```

Discussion

Also returns `false` if *`key`* is not in *`table`* or if the receiver lacks a PPD file.

See Also

`isKeyInTable` (page 1132)

`stringForKeyInTable` (page 1134)

deviceDescription

Returns a dictionary of keys and values describing the device.

```
public NSDictionary deviceDescription()
```

Discussion

See NSGraphics "Constants" (page 727) for possible keys. The only key guaranteed to exist is `NSDeviceIsPrinter`.

domain

```
public String domain()
```

Discussion

Deprecated.

floatForKeyInTable

Returns a floating-point value associated with *`key`* in *`table`*.

```
public float floatForKeyInTable(String key, String table)
```

Discussion

Returns 0.0 if *`key`* is not in *`table`* or if the receiver lacks a PPD file.

See Also

`isKeyInTable` (page 1132)

`stringForKeyInTable` (page 1134)

host

Deprecated.

```
public String host()
```

imageRectForPaper

Deprecated.

```
public NSRect imageRectForPaper(String paperName)
```

Discussion

If used, it attempts to determine and return the bounds of the imagable area for a particular paper named *paperName*, but the result is not completely reliable.

See Also

`pageSizeForPaper` (page 1133)

intForKeyInTable

Returns an integer value associated with *key* in *table*.

```
public int intForKeyInTable(String key, String table)
```

Discussion

Returns 0 if *key* is not in *table* or if the receiver lacks a PPD file.

See Also

`isKeyInTable` (page 1132)
`stringForKeyInTable` (page 1134)

isColor

Deprecated.

```
public boolean isColor()
```

isFontAvailable

Deprecated.

```
public boolean isFontAvailable(String faceName)
```

isKeyInTable

Returns `true` if *key* is in *table*, `false` otherwise.

```
public boolean isKeyInTable(String key, String table)
```

isOutputStackInReverseOrder

Deprecated.

```
public boolean isOutputStackInReverseOrder()
```

languageLevel

Returns the PostScript language level recognized by the printer.

```
public int languageLevel()
```

Discussion
Returns 0 if the receiver is not a Postscript printer.

name

Returns the printer's name.

```
public String name()
```

See Also
`printerNames` (page 1129)
`printerWithName` (page 1130)

note

Deprecated.

```
public String note()
```

pageSizeForPaper

Returns the size of the page for the paper type *paperName*.

```
public NSSize pageSizeForPaper(String paperName)
```

Discussion
Possible values for *paperName* are contained in the printer's PPD file. Typical values are Letter and Legal. Returns a zero size if *paperName* is not recognized, or its entry in the PPD cannot be parsed.

See Also
`imageRectForPaper` (page 1131)

rectForKeyInTable

Returns the rectangle associated with *key* in *table*.

```
public NSRect rectForKeyInTable(String key, String table)
```

Discussion
Returns `NSRect.ZeroRect` if *key* is not in *table* or if the receiver lacks a PPD file.

See Also
`isKeyInTable` (page 1132)
`stringForKeyInTable` (page 1134)

sizeForKeyInTable

Returns the size associated with *key* in *table*.

```
public NSSize sizeForKeyInTable(String key, String table)
```

Discussion

The returned width and height are 0.0 if *key* is not in *table* or if the receiver lacks a PPD file.

See Also

`isKeyInTable` (page 1132)

`stringForKeyInTable` (page 1134)

statusForTable

Returns the status of *table*.

```
public int statusForTable(String table)
```

Discussion

The possible return values are described in "Constants" (page 1135).

stringForKeyInTable

Returns the first occurrence of a value associated with *key* in *table*.

```
public String stringForKeyInTable(String key, String table)
```

Discussion

If *key* is a main keyword only, and if that keyword has options in the PPD file, this method returns an empty string. Use `stringListForKeyInTable` (page 1134) to retrieve the values for all occurrences of a main keyword. Returns `null` if *key* is not in *table*.

See Also

`isKeyInTable` (page 1132)

`booleanForKeyInTable` (page 1130)

`floatForKeyInTable` (page 1131)

`intForKeyInTable` (page 1132)

`rectForKeyInTable` (page 1133)

`sizeForKeyInTable` (page 1134)

stringListForKeyInTable

Returns an array of strings, one for each occurrence, associated with *key* in *table*.

```
public NSArray stringListForKeyInTable(String key, String table)
```

Discussion

Returns `null` if *key* is not in *table*.

See Also

`isKeyInTable` (page 1132)

`stringForKeyInTable` (page 1134)

type

Returns a description of the printer's make and model.

```
public String type()
```

See Also

`printerTypes` (page 1129)

Constants

These constants describe the state of a printer information table stored by an NSPrinter object. They're used by `statusForTable` (page 1134).

Constant	Description
`PrinterTableOK`	Printer table was found and is valid.
`PrinterTableNotFound`	Printer table was not found.
`PrinterTableError`	Printer table is not valid.

NSPrintInfo

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.application
Companion guide	Printing Programming Topics for Cocoa

Overview

An NSPrintInfo object stores information that's used to generate output. A shared NSPrintInfo object is automatically created for an application and is used by default for all printing jobs for that application.

Tasks

Constructors

`NSPrintInfo` (page 1140)
Creates an empty NSPrintInfo.

Managing the Shared NSPrintInfo

`setSharedPrintInfo` (page 1140)
Sets the shared NSPrintInfo object to *`printInfo`*.

`sharedPrintInfo` (page 1141)
Returns the shared NSPrintInfo object.

Managing the Printing Rectangle

`bottomMargin` (page 1141)
Returns the height of the bottom margin in points.

`imageablePageBounds` (page 1142)
Returns the imageable area of a sheet of paper specified by the receiver, taking into account the current printer, paper size, and orientation settings, but not scaling.

`leftMargin` (page 1143)
Returns the width of the left margin in points.

`orientation` (page 1143)
Returns the orientation attribute.

`paperName` (page 1143)
Returns the paper name, such as Letter or Legal.

`localizedPaperName` (page 1143)
Returns the human-readable name of the currently selected paper size, suitable for presentation in user interfaces.

`paperSize` (page 1143)
Returns the size of the paper in points.

`rightMargin` (page 1144)
Returns the width of the right margin in points.

`setBottomMargin` (page 1144)
Sets the bottom margin to *margin* specified in points.

`setLeftMargin` (page 1145)
Sets the left margin to *margin* specified in points.

`setOrientation` (page 1145)
Sets the page orientation to *orientation* where *orientation* is either `PortraitOrientation` or `LandscapeOrientation`.

`setPaperName` (page 1146)
Sets the paper name to *name* (for example, Letter or Legal).

`setPaperSize` (page 1146)
Sets the width and height of the paper to *aSize* specified in points.

`setRightMargin` (page 1146)
Sets the right margin to *margin* specified in points.

`setTopMargin` (page 1147)
Sets the top margin to *margin* specified in points.

`topMargin` (page 1147)
Returns the top margin in points.

Pagination

`horizontalPagination` (page 1141)
Returns the horizontal pagination mode.

`setHorizontalPagination` (page 1144)
Sets the horizontal pagination to *mode*.

`setVerticalPagination` (page 1147)
Sets the vertical pagination to *mode*.

`verticalPagination` (page 1148)
Returns the vertical pagination mode.

Positioning the Image on the Page

`isHorizontallyCentered` (page 1142)
  Returns `true` if the image is centered horizontally, `false` otherwise.

`isVerticallyCentered` (page 1142)
  Returns `true` if the image is centered vertically, `false` otherwise.

`setHorizontallyCentered` (page 1144)

`setVerticallyCentered` (page 1147)

Specifying the Printer

`printer` (page 1144)
  Returns the NSPrinter to be used for printing.

`setPrinter` (page 1146)
  Sets the printer used in subsequent printing jobs to *`printer`*.

Controlling Printing

`jobDisposition` (page 1142)
  Returns the action specified for the job.

`setJobDisposition` (page 1145)
  Sets the action specified for the job to *`disposition`*.

`setUpPrintOperationDefaultValues` (page 1147)
  Validates the attributes encapsulated by the receiver.

Accessing the Dictionary

`dictionary` (page 1141)
  Returns the receiver’s dictionary that stores its attribute settings

Deprecated Methods

`defaultPrinter` (page 1140)
  Deprecated.

`setDefaultPrinter` (page 1140)
  Deprecated.

`sizeForPaperName` (page 1141)
  Deprecated.

Constructors

NSPrintInfo

Creates an empty NSPrintInfo.

```
public NSPrintInfo()
```

Creates a new NSPrintInfo object, assigning it the parameters specified in *aDictionary*.

```
public NSPrintInfo(NSDictionary aDictionary)
```

Discussion
The possible key-value pairs contained in *aDictionary* are described in "Constants" (page 1148).

See Also
`dictionary` (page 1141)

Static Methods

defaultPrinter

Deprecated.

```
public static NSPrinter defaultPrinter()
```

setDefaultPrinter

Deprecated.

```
public static void setDefaultPrinter(NSPrinter aPrinter)
```

setSharedPrintInfo

Sets the shared NSPrintInfo object to *printInfo*.

```
public static void setSharedPrintInfo(NSPrintInfo printInfo)
```

Discussion
The shared NSPrintInfo object defines the settings for the NSPageLayout panel and print operations that will be used if no NSPrintInfo object is specified for those operations. *printInfo* should never be `null`.

See Also
`sharedPrintInfo` (page 1141)

sharedPrintInfo

Returns the shared NSPrintInfo object.

```
public static NSPrintInfo sharedPrintInfo()
```

See Also
`setSharedPrintInfo` (page 1140)

sizeForPaperName

Deprecated.

```
public static NSSize sizeForPaperName(String name)
```

Discussion
Use NSPrinter's method `pageSizeForPaper` (page 1133) instead.

Instance Methods

bottomMargin

Returns the height of the bottom margin in points.

```
public float bottomMargin()
```

See Also
`setBottomMargin` (page 1144)

dictionary

Returns the receiver's dictionary that stores its attribute settings

```
public NSMutableDictionary dictionary()
```

Discussion
. The key-value pairs contained in the dictionary are described in "Constants" (page 1148). Modifying the returned dictionary changes the receiver's attributes.

horizontalPagination

Returns the horizontal pagination mode.

```
public int horizontalPagination()
```

Discussion
It can return one of the pagination modes described in "Constants" (page 1148).

See Also

`setVerticalPagination` (page 1147)

`verticalPagination` (page 1148)

imageablePageBounds

Returns the imageable area of a sheet of paper specified by the receiver, taking into account the current printer, paper size, and orientation settings, but not scaling.

```
public NSRect imageablePageBounds()
```

Discussion

"Imageable area" is the maximum area that can possibly be marked on by the printer hardware, not the area defined by the current margin settings. The rectangle is in a coordinate space measured by points, with (0, 0) being the lower-left corner of the oriented sheet and (*paperWidth*, *paperHeight*) being the upper-right corner of the oriented sheet. The imageable bounds may extend past the edges of the sheet when, for example, a printer driver specifies it so that borderless printing can be done reliably.

Availability

Available in Mac OS X v10.2 and later.

isHorizontallyCentered

Returns `true` if the image is centered horizontally, `false` otherwise.

```
public boolean isHorizontallyCentered()
```

See Also

`isVerticallyCentered` (page 1142)

`setHorizontallyCentered` (page 1144)

isVerticallyCentered

Returns `true` if the image is centered vertically, `false` otherwise.

```
public boolean isVerticallyCentered()
```

See Also

`isHorizontallyCentered` (page 1142)

`setVerticallyCentered` (page 1147)

jobDisposition

Returns the action specified for the job.

```
public String jobDisposition()
```

Discussion

See `setJobDisposition` (page 1145) for a description of return values.

leftMargin

Returns the width of the left margin in points.

```
public float leftMargin()
```

See Also
`setLeftMargin` (page 1145)

localizedPaperName

Returns the human-readable name of the currently selected paper size, suitable for presentation in user interfaces.

```
public String localizedPaperName()
```

Discussion
This is typically different from the name returned by `paperName` (page 1143), which is almost never suitable for presentation to the user.

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

orientation

Returns the orientation attribute.

```
public int orientation()
```

Discussion
See `setOrientation` (page 1145) for a description of return values.

paperName

Returns the paper name, such as Letter or Legal.

```
public String paperName()
```

Discussion
Paper names are implementation specific.

See Also
`setPaperName` (page 1146)
`localizedPaperName` (page 1143)

paperSize

Returns the size of the paper in points.

```
public NSSize paperSize()
```

See Also
`setPaperSize` (page 1146)

printer

Returns the NSPrinter to be used for printing.

```
public NSPrinter printer()
```

See Also
`setPrinter` (page 1146)

rightMargin

Returns the width of the right margin in points.

```
public float rightMargin()
```

See Also
`setRightMargin` (page 1146)

setBottomMargin

Sets the bottom margin to *`margin`* specified in points.

```
public void setBottomMargin(float margin)
```

See Also
`bottomMargin` (page 1141)

setHorizontallyCentered

```
public void setHorizontallyCentered(boolean flag)
```

Discussion
If *`flag`* is `true` the image will be centered horizontally.

See Also
`isHorizontallyCentered` (page 1142)
`isVerticallyCentered` (page 1142)
`setVerticallyCentered` (page 1147)

setHorizontalPagination

Sets the horizontal pagination to *`mode`*.

```
public void setHorizontalPagination(int mode)
```

Discussion

mode can be one of the pagination modes described in "Constants" (page 1148).

See Also

`horizontalPagination` (page 1141)
`setVerticalPagination` (page 1147)
`verticalPagination` (page 1148)

setJobDisposition

Sets the action specified for the job to *disposition*.

```
public void setJobDisposition(String disposition)
```

Discussion

disposition can be one of the following:

- `PrintSpoolJob` is a normal print job.
- `PrintPreviewJob` sends the print job to the Preview application.
- `PrintSaveJob` saves the print job to a file.
- `PrintCancelJob` aborts the print job.

See Also

`jobDisposition` (page 1142)

setLeftMargin

Sets the left margin to *margin* specified in points.

```
public void setLeftMargin(float margin)
```

See Also

`leftMargin` (page 1143)

setOrientation

Sets the page orientation to *orientation* where *orientation* is either `PortraitOrientation` or `LandscapeOrientation`.

```
public void setOrientation(int orientation)
```

Discussion

This method may change either the paper name or size for consistency. To avoid this side effect set the values in the dictionary directly.

See Also

`dictionary` (page 1141)
`orientation` (page 1143)

setPaperName

Sets the paper name to *name* (for example, Letter or Legal).

```
public void setPaperName(String name)
```

Discussion

Paper names are implementation specific. This method may change either the size or orientation for consistency. To avoid this side effect set the values in the dictionary directly.

See Also

`dictionary` (page 1141)

`paperName` (page 1143)

setPaperSize

Sets the width and height of the paper to *aSize* specified in points.

```
public void setPaperSize(NSSize aSize)
```

Discussion

This method may change either the paper name or orientation for consistency. To avoid this side effect set the values in the dictionary directly.

See Also

`dictionary` (page 1141)

`paperSize` (page 1143)

setPrinter

Sets the printer used in subsequent printing jobs to *printer*.

```
public void setPrinter(NSPrinter printer)
```

Discussion

This method iterates through the dictionary. If a feature in the dictionary is not supported by the new printer (as determined by a query to the PPD file), that feature is removed from the dictionary.

See Also

`printer` (page 1144)

setRightMargin

Sets the right margin to *margin* specified in points.

```
public void setRightMargin(float margin)
```

See Also

`rightMargin` (page 1144)

setTopMargin

Sets the top margin to *margin* specified in points.

```
public void setTopMargin(float margin)
```

See Also

`topMargin` (page 1147)

setUpPrintOperationDefaultValues

Validates the attributes encapsulated by the receiver.

```
public void setUpPrintOperationDefaultValues()
```

Discussion

Invoked when the print operation is about to start. Subclasses may override this method to set default values for any attributes that are not set.

setVerticallyCentered

```
public void setVerticallyCentered(boolean flag)
```

Discussion

If *flag* is `true`, the image will be vertically centered.

See Also

`isHorizontallyCentered` (page 1142)

`isVerticallyCentered` (page 1142)

`setHorizontallyCentered` (page 1144)

setVerticalPagination

Sets the vertical pagination to *mode*.

```
public void setVerticalPagination(int mode)
```

Discussion

mode can be one of the pagination modes described in "Constants" (page 1148).

See Also

`horizontalPagination` (page 1141)

`setHorizontalPagination` (page 1144)

`verticalPagination` (page 1148)

topMargin

Returns the top margin in points.

```
public float topMargin()
```

See Also

setTopMargin (page 1147)

verticalPagination

Returns the vertical pagination mode.

```
public int verticalPagination()
```

Discussion

It can return one of the pagination modes described in "Constants" (page 1148).

See Also

horizontalPagination (page 1141)
setHorizontalPagination (page 1144)

Constants

These constants specify the different ways in which an image is divided into pages. They're used by horizontalPagination (page 1141), setHorizontalPagination (page 1144), verticalPagination (page 1148), and setVerticalPagination (page 1147).

Pagination Constant	Meaning
AutoPagination	The image is divided into equal-sized rectangles and placed in one column of pages.
FitPagination	The image is scaled to produce one column or one row of pages.
ClipPagination	The image is clipped to produce one column or row of pages.

These constants specify page orientations. They're used by orientation (page 1143) and setOrientation (page 1145).

Orientation	Meaning
PortraitOrientation	Orientation is portrait (page is taller than it is wide).
LandscapeOrientation	Orientation is landscape (page is wider than it is tall).

These constants specify job dispositions. They're used by jobDisposition (page 1142), and setJobDisposition (page 1145).

Mode	Meaning
PrintSpoolJob	Normal print job.
PrintPreviewJob	Send to Preview application.

Mode	Meaning
PrintSaveJob	Save to a file.
PrintCancelJob	Cancel print job.

The following sets of constants specify the keys in the dictionary returned by dictionary (page 1141) or used to create an instance.

The following dictionary keys access page setup attributes.

Constant	Type	Description
PrintPaperName	String	The paper name.
PrintPaperSize	NSSize	Height and width of paper in points.
PrintOrientation	int	PortraitOrientation or LandscapeOrientation
PrintScalingFactor	float	Scale factor percentage before pagination.

The following dictionary keys access pagination attributes.

Constant	Type	Description
PrintLeftMargin	float	The left margin in points.
PrintRightMargin	float	The right margin in points.
PrintTopMargin	float	The top margin in points.
PrintBottomMargin	float	The bottom margin in points.
PrintHorizontallyCentered	boolean	If true, pages are centered horizontally.
PrintVerticallyCentered	boolean	If true, pages are centered vertically.
PrintHorizontalPagination	int	AutoPagination, FitPagination, or ClipPagination. See setHorizontalPagination (page 1144) for details.
PrintVerticalPagination	int	AutoPagination, FitPagination, or ClipPagination. See setVerticalPagination (page 1147) for details.

The following dictionary keys access print job attributes.

Constant	Type	Description
PrintPrinter	NSPrinter	The printer to use.
PrintCopies	int	Number of copies to spool.
PrintAllPages	boolean	If true, includes all pages in output.

Constant	Type	Description
PrintFirstPage	int	The first page in the print job.
PrintLastPage	int	The last page in the print job.
PrintReversePageOrder	boolean	If true, prints last page first.
PrintMustCollate	boolean	If true, collates output.
PrintJobDisposition	String	PrintSpoolJob, PrintPreviewJob, PrintSaveJob, or PrintCancelJob. See setJobDisposition (page 1145) for details.
PrintSavePath	String	Pathname to save as a file if job disposition is PrintSaveJob.
PrintPagesAcross	Number	The number of logical pages to be tiled horizontally on a physical sheet of paper. (Available in Mac OS X v10.4 and later.)
PrintPagesDown	Number	The number of logical pages to be tiled veritcally on a physical sheet of paper. (Available in Mac OS X v10.4 and later.)
PrintTime	Date	The time at which printing should begin. (Available in Mac OS X v10.4 and later.)
PrintDetailedError-Reporting	Number, containing aboolean	If true, produce detailed reports when an error occurs. (Available in Mac OS X v10.4 and later.)
PrintFaxNumber	String	A fax number. (Available in Mac OS X v10.4 and later.)
PrintPrinterName	String	The name of a printer. (Available in Mac OS X v10.4 and later.)

NSPrintOperation

Inherits from NSObject

Package: com.apple.cocoa.application

Companion guide Printing Programming Topics for Cocoa

Overview

An NSPrintOperation object controls operations that generate Encapsulated PostScript (EPS) code, Portable Document Format (PDF) code, or print jobs. NSPrintOperation works in conjunction with two other objects: an NSPrintInfo object, which specifies how the code should be generated, and an NSView object, which generates the actual code.

It is important to note that the majority of methods in NSPrintOperation copy the instance of NSPrintInfo passed into them. Future changes to that print info are not reflected in the print info retained by NSPrintOperation. All changes should be made to the print info before passing to NSPrintOperation methods. The only method in NSPrintOperation which does not copy the NSPrintInfo instance is `setPrintInfo` (page 1162).

Tasks

Constructors

`NSPrintOperation` (page 1154)
    Creates an empty NSPrintOperation.

Creating an NSPrintOperation

`EPSOperationWithViewInsideRect` (page 1154)
    Returns a new NSPrintOperation object that controls the copying of EPS graphics from the area specified by *rect* in *aView*.

`PDFOperationWithViewInsideRect` (page 1155)
    Returns a new NSPrintOperation object that controls the copying of PDF graphics from the area specified by *rect* in *aView*.

`printOperationWithView` (page 1156)
    Returns a new NSPrintOperation that controls the printing of *aView*.

Setting the Current NSPrintOperation for This Thread

`currentOperation` (page 1154)
Returns the current print operation for this thread.

`setCurrentOperation` (page 1156)
Sets the current print operation for this thread to *`operation`*.

Determining the Type of Operation

`isCopyingOperation` (page 1158)
Returns `true` if the receiver is an EPS or PDF copy operation.

`isEPSOperation` (page 1158)
Returns `true` if the receiver controls an EPS operation (initiated by a copy command), and `false` if the receiver controls a printing operation (initiated by a print command).

Modifying the NSPrintInfo Object

`printInfo` (page 1159)
Returns the receiver's NSPrintInfo object.

`setPrintInfo` (page 1162)
Sets the receiver's NSPrintInfo object to *`aPrintInfo`*.

Getting the NSView Object

`view` (page 1164)
Returns the NSView object that generates the actual PostScript or PDF code controlled by the receiver.

Running a Print Operation

`runOperation` (page 1160)
Runs the operation, either copying an EPS or PDF graphic or printing a job.

`runModalOperation` (page 1160)
Runs the print operation, with document-modal sheets attached to *`docWindow`*. When the modal session ends, if neither *`delegate`* or *`didRunSelector`* are `null`, *`didRunSelector`* is sent to *`delegate`* with *`contextInfo`* as an argument. The

`cleanUpOperation` (page 1157)
Invoked by `runOperation` (page 1160) at the end of an operation to remove the receiver as the current operation.

`deliverResult` (page 1158)
Delivers the results generated by `runOperation` (page 1160) to the intended destination (for example, the printer spool or preview application).

Modifying the User Interface

`showPanels` (page 1163)
Returns `true` if the NSPrintPanel will be used in the operation, otherwise `false`.

`setShowPanels` (page 1162)

`showsPrintPanel` (page 1163)
Returns `true` if the NSPrintPanel will be used in the operation, otherwise `false`.

`showsProgressPanel` (page 1164)
Returns `true` if a progress panel will be used in the operation, otherwise `false`.

`setShowsPrintPanel` (page 1162)
If *`flag`* is `true`, then the receiver's NSPrintPanel is used in the operation; otherwise, it is not.

`setShowsProgressPanel` (page 1163)
If *`flag`* is `true`, then the receiver's progress panel is used in the operation; otherwise, it is not.

`accessoryView` (page 1156)
Returns the accessory view used by the receiver's NSPrintPanel object.

`setAccessoryView` (page 1160)
Sets the custom accessory view *`aView`*, to be used by the receiver's NSPrintPanel object.

`jobStyleHint` (page 1159)
Returns the type of content that the print job is printing.

`setJobStyleHint` (page 1161)
Sets the type of content that the print job is printing.

`printPanel` (page 1159)
Returns the NSPrintPanel object used when running the operation.

`setPrintPanel` (page 1162)
Sets the receiver's NSPrintPanel used in the operation to *`panel`*.

Managing the Drawing Context

`context` (page 1157)
Returns the receiver's display context used for generating output.

`createContext` (page 1157)
Creates the display context for output generation, using the receiver's NSPrintInfo settings.

`destroyContext` (page 1158)
Destroys the receiver's display context.

Modifying Page Information

`currentPage` (page 1158)
Returns the page number of the page currently being printed.

`pageOrder` (page 1159)
Returns the order in which pages will be printed.

`setPageOrder` (page 1161)
Sets the order in which pages will be printed to *order*.

Managing Printing Threads

`canSpawnSeparateThread` (page 1157)
Returns whether the receiver is allowed to spawn a separate printing thread.

`setCanSpawnSeparateThread` (page 1161)
Sets whether the receiver is allowed to spawn a separate printing thread.

Constructors

NSPrintOperation

Creates an empty NSPrintOperation.

```
public NSPrintOperation()
```

Static Methods

currentOperation

Returns the current print operation for this thread.

```
public static NSPrintOperation currentOperation()
```

Discussion
Returns `null` if there isn't a current operation.

See Also
`setCurrentOperation` (page 1156)

EPSOperationWithViewInsideRect

Returns a new NSPrintOperation object that controls the copying of EPS graphics from the area specified by *rect* in *aView*.

```
public static NSPrintOperation EPSOperationWithViewInsideRect(NSView aView, NSRect
     rect, NSMutableData data)
```

Discussion
The new NSPrintOperation object uses the default NSPrintInfo object. The EPS code is written to *data*. Throws an exception if there is already a print operation in progress; otherwise the returned object is made the current print operation for this thread.

Returns a new NSPrintOperation object that controls the copying of EPS graphics from the area specified by *rect* in *aView*.

```
public static NSPrintOperation EPSOperationWithViewInsideRect(NSView aView, NSRect
      rect, NSMutableData data, NSPrintInfo aPrintInfo)
```

Discussion

The new NSPrintOperation object uses the settings stored in *aPrintInfo*. The code is written to *data*. Throws an exception if there is already a print operation in progress; otherwise the returned object is made the current print operation for this thread.

Creates and returns a new NSPrintOperation object that controls the copying of EPS graphics from the area specified by *rect* in *aView*.

```
public static NSPrintOperation EPSOperationWithViewInsideRect(NSView aView, NSRect
      rect, String path, NSPrintInfo aPrintInfo)
```

Discussion

The new NSPrintOperation object uses the settings stored in *aPrintInfo*. The code is written to *path*. Throws an exception if there is already a print operation in progress; otherwise the returned object is made the current print operation for this thread.

PDFOperationWithViewInsideRect

Returns a new NSPrintOperation object that controls the copying of PDF graphics from the area specified by *rect* in *aView*.

```
public static NSPrintOperation PDFOperationWithViewInsideRect(NSView aView, NSRect
      rect, NSMutableData data)
```

Discussion

The new NSPrintOperation object uses the default NSPrintInfo object. The PDF is written to *data*. Throws an exception if there is already a print operation in progress; otherwise the returned object is made the current print operation for this thread.

Returns a new NSPrintOperation object that controls the copying of PDF graphics from the area specified by *rect* in *aView*.

```
public static NSPrintOperation PDFOperationWithViewInsideRect(NSView aView, NSRect
      rect, NSMutableData data, NSPrintInfo aPrintInfo)
```

Discussion

The new NSPrintOperation object uses the settings stored in *aPrintInfo*. The PDF is written to *data*. Throws an exception if there is already a print operation in progress; otherwise the returned object is made the current print operation for this thread.

Creates and returns a new NSPrintOperation object that controls the copying of PDF graphics from the area specified by *rect* in *aView*.

```
public static NSPrintOperation PDFOperationWithViewInsideRect(NSView aView, NSRect
      rect, String path, NSPrintInfo aPrintInfo)
```

Discussion
The new NSPrintOperation object uses the settings stored in *aPrintInfo*. The PDF is written to *path*. Throws an exception if there is already a print operation in progress; otherwise the returned object is made the current print operation for this thread.

printOperationWithView

Returns a new NSPrintOperation that controls the printing of *aView*.

```
public static NSPrintOperation printOperationWithView(NSView aView)
```

Discussion
The new NSPrintOperation object uses the settings stored in the shared NSPrintInfo object. Throws an exception if there is already a print operation in progress; otherwise the returned object is made the current print operation for this thread.

Returns a new NSPrintOperation that controls the printing of *aView*.

```
public static NSPrintOperation printOperationWithView(NSView aView, NSPrintInfo
    aPrintInfo)
```

Discussion
The new NSPrintOperation object uses the settings stored in *aPrintInfo*. Throws an exception if there is already a print operation in progress; otherwise the returned object is made the current print operation for this thread.

setCurrentOperation

Sets the current print operation for this thread to *operation*.

```
public static void setCurrentOperation(NSPrintOperation operation)
```

Discussion
If *operation* is `null`, then there is no current print operation.

See Also
`currentOperation` (page 1154)

Instance Methods

accessoryView

Returns the accessory view used by the receiver's NSPrintPanel object.

```
public NSView accessoryView()
```

Discussion
You use `setAccessoryView` (page 1160) to customize the default NSPrintPanel object without having to subclass NSPrintPanel or specify your own NSPrintPanel object.

See Also

printPanel (page 1159)

setPrintPanel (page 1162)

setShowPanels (page 1162)

showPanels (page 1163)

canSpawnSeparateThread

Returns whether the receiver is allowed to spawn a separate printing thread.

```
public boolean canSpawnSeparateThread()
```

See Also

setCanSpawnSeparateThread (page 1161)

cleanUpOperation

Invoked by runOperation (page 1160) at the end of an operation to remove the receiver as the current operation.

```
public void cleanUpOperation()
```

Discussion

You typically do not invoke this method directly.

context

Returns the receiver’s display context used for generating output.

```
public NSGraphicsContext context()
```

See Also

createContext (page 1157)

destroyContext (page 1158)

createContext

Creates the display context for output generation, using the receiver’s NSPrintInfo settings.

```
public NSGraphicsContext createContext()
```

Discussion

Do not invoke this method directly—it is invoked before any output is generated.

See Also

context (page 1157)

destroyContext (page 1158)

currentPage

Returns the page number of the page currently being printed.

```
public int currentPage()
```

See Also

pageOrder (page 1159)

setPageOrder (page 1161)

deliverResult

Delivers the results generated by runOperation (page 1160) to the intended destination (for example, the printer spool or preview application).

```
public boolean deliverResult()
```

Discussion

Returns true if the operation was successful, false otherwise. Do not invoke this method directly—it is invoked automatically when the operation is done generating the output.

destroyContext

Destroys the receiver's display context.

```
public void destroyContext()
```

Discussion

Do not invoke this method directly—it is invoked at the end of a print operation.

See Also

context (page 1157)

createContext (page 1157)

isCopyingOperation

Returns true if the receiver is an EPS or PDF copy operation.

```
public boolean isCopyingOperation()
```

isEPSOperation

Returns true if the receiver controls an EPS operation (initiated by a copy command), and false if the receiver controls a printing operation (initiated by a print command).

```
public boolean isEPSOperation()
```

jobStyleHint

Returns the type of content that the print job is printing.

```
public String jobStyleHint()
```

Discussion

Returns `null` if no job style hint has been set.

Availability

Available in Mac OS X v10.2 and later.

See Also

`setJobStyleHint` (page 1161)

pageOrder

Returns the order in which pages will be printed.

```
public int pageOrder()
```

Discussion

See "Constants" (page 1164) for possible return values.

See Also

`currentPage` (page 1158)

printInfo

Returns the receiver's NSPrintInfo object.

```
public NSPrintInfo printInfo()
```

See Also

`setPrintInfo` (page 1162)

printPanel

Returns the NSPrintPanel object used when running the operation.

```
public NSPrintPanel printPanel()
```

See Also

`accessoryView` (page 1156)

`setAccessoryView` (page 1160)

`setPrintPanel` (page 1162)

`setShowPanels` (page 1162)

`showPanels` (page 1163)

runModalOperation

Runs the print operation, with document-modal sheets attached to *docWindow*. When the modal session ends, if neither *delegate* or *didRunSelector* are null, *didRunSelector* is sent to *delegate* with *contextInfo* as an argument. The

```
public void runModalOperation(NSWindow docWindow, Object delegate, NSSelector
    didRunSelector, Object contextInfo)
```

Discussion

didRunSelector argument must have the following signature:

```
public void printOperationDidRun(NSPrintOperation printOperation,  boolean
success, void  contextInfo)
```

The value of *success* is true if the print operation ran to completion without cancellation or error, and false otherwise.

If you send setCanSpawnSeparateThread (page 1161) to an NSPrintOperation object with an argument of true, then the delegate specified in a subsequent invocation of runModalOperation (page 1160) may be messaged in that spawned, non-main thread.

runOperation

Runs the operation, either copying an EPS or PDF graphic or printing a job.

```
public boolean runOperation()
```

Discussion

Returns true if successful, false otherwise. The operation runs to completion in the current thread, blocking the application. A separate thread is not spawned, even if canSpawnSeparateThread (page 1157) is true. Use runModalOperation (page 1160) to use document-modal sheets and to allow a separate thread to perform the operation.

See Also

cleanUpOperation (page 1157)
deliverResult (page 1158)

setAccessoryView

Sets the custom accessory view *aView*, to be used by the receiver's NSPrintPanel object.

```
public void setAccessoryView(NSView aView)
```

Discussion

By using this method you do not need to subclass NSPrintPanel or specify your own NSPrintPanel object. The NSPrintPanel is automatically resized to accommodate the new accessory view *aView*.

See Also

accessoryView (page 1156)
printPanel (page 1159)
setPrintPanel (page 1162)
setShowPanels (page 1162)

`showPanels` (page 1163)

setCanSpawnSeparateThread

Sets whether the receiver is allowed to spawn a separate printing thread.

```
public void setCanSpawnSeparateThread(boolean canSpawnSeparateThread)
```

Discussion

If *canSpawnSeparateThread* is `true`, an NSThread is detached when the print panel is dismissed (or immediately, if the panel is not to be displayed). The new thread performs the print operation, so that control can return to your application. A thread is detached only if the print operation is run using `runModalOperation` (page 1160). If *canSpawnSeparateThread* is `false`, the operation runs on the current thread, blocking the application until the operation completes.

If you send `setCanSpawnSeparateThread` (page 1161) to an NSPrintOperation object with an argument of `true`, then the delegate specified in a subsequent invocation of `runModalOperation` (page 1160) may be messaged in that spawned, non-main thread.

See Also

`canSpawnSeparateThread` (page 1157)

setJobStyleHint

Sets the type of content that the print job is printing.

```
public void setJobStyleHint(String hint)
```

Discussion

This controls the set of items that appear in the Presets menu of the simplified Print panel interface presented by this operation, if it presents one. The supported job style hints are described in the "Constants" (page 1170) section of NSPrintPanel. If *hint* is `null`, the standard interface is used.

Availability

Available in Mac OS X v10.2 and later.

See Also

`jobStyleHint` (page 1159)

setPageOrder

Sets the order in which pages will be printed to *order*.

```
public void setPageOrder(int order)
```

Discussion

order is one of the values described in "Constants" (page 1164).

See Also

`currentPage` (page 1158)
`pageOrder` (page 1159)

setPrintInfo

Sets the receiver's NSPrintInfo object to *aPrintInfo*.

```
public void setPrintInfo(NSPrintInfo aPrintInfo)
```

See Also

printInfo (page 1159)

setPrintPanel

Sets the receiver's NSPrintPanel used in the operation to *panel*.

```
public void setPrintPanel(NSPrintPanel panel)
```

See Also

accessoryView (page 1156)
printPanel (page 1159)
setAccessoryView (page 1160)
setShowPanels (page 1162)
showPanels (page 1163)

setShowPanels

```
public void setShowPanels(boolean flag)
```

Discussion

If *flag* is `true` then the receiver's NSPrintPanel will be used in the operation; otherwise it will not. This method also affects whether a progress panel is presented while the operation runs. If an EPS or PDF copy operation is being performed, neither panel is displayed, regardless of the value of *flag*.

See Also

accessoryView (page 1156)
printPanel (page 1159)
–setAccessoryView (page 1160)
setPrintPanel (page 1162)
showPanels (page 1163)

setShowsPrintPanel

If *flag* is `true`, then the receiver's NSPrintPanel is used in the operation; otherwise, it is not.

```
public void setShowsPrintPanel(boolean flag)
```

Discussion

This method does not affect the display of a progress panel; that operation is now controlled by setShowsProgressPanel (page 1163).

If an EPS or PDF copy operation is being performed, an NSPrintPanel is not displayed, regardless of the value of *flag*.

Availability

Available in Mac OS X v10.4 and later.

See Also

`setShowsProgressPanel` (page 1163)
`showsPrintPanel` (page 1163)

setShowsProgressPanel

If *flag* is `true`, then the receiver's progress panel is used in the operation; otherwise, it is not.

```
public void setShowsProgressPanel(boolean flag)
```

Discussion

This method does not affect the display of a print panel; that operation is now controlled by `setShowsPrintPanel` (page 1162). This method replaces the `setShowPanels` (page 1162) method.

Availability

Available in Mac OS X v10.4 and later.

See Also

`setShowsPrintPanel` (page 1162)
`showsProgressPanel` (page 1164)

showPanels

Returns `true` if the NSPrintPanel will be used in the operation, otherwise `false`.

```
public boolean showPanels()
```

See Also

`accessoryView` (page 1156)
`printPanel` (page 1159)
`setAccessoryView` (page 1160)
`setPrintPanel` (page 1162)
`setShowPanels` (page 1162)

showsPrintPanel

Returns `true` if the NSPrintPanel will be used in the operation, otherwise `false`.

```
public boolean showsPrintPanel()
```

Availability

Available in Mac OS X v10.4 and later.

See Also

`setShowsPrintPanel` (page 1162)

showsProgressPanel

Returns `true` if a progress panel will be used in the operation, otherwise `false`.

```
public boolean showsProgressPanel()
```

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

See Also
`setShowsProgressPanel` (page 1163)

view

Returns the NSView object that generates the actual PostScript or PDF code controlled by the receiver.

```
public NSView view()
```

Constants

These constants specify the page order. They're used by `pageOrder` (page 1159) and `setPageOrder` (page 1161).

Constant	Description
`AscendingPageOrder`	Ascending (back to front) page order.
`DescendingPageOrder`	Descending (front to back) page order.
`SpecialPageOrder`	The spooler does not rearrange pages—they are printed in the order received by the spooler.
`UnknownPageOrder`	No page order specified.

NSPrintPanel

Inherits from	NSObject
Package:	com.apple.cocoa.application
Companion guides	Printing Programming Topics for Cocoa Sheet Programming Topics for Cocoa

Overview

NSPrintPanel creates a Print panel used to query the user for information about a print job, such as which pages to print and how many copies, and to executes the Print command.

Print panels can display a simplified interface when printing certain types of data. The interface presents a list of presets for commonly used sets of print settings based on the type of data being printed. The user can select a preset instead of individually selecting the appropriate settings in each pane of the Print panel. The `setJobStyleHint` (page 1169) method activates the simplified interface and identifies which presets to present. Pass `null` to this method to deactivate the simplified interface.

Tasks

Constructors

`NSPrintPanel` (page 1166)
Creates an empty NSPrintPanel.

Creating an NSPrintPanel

`printPanel` (page 1167)
Returns a newly created NSPrintPanel object.

Customizing the Panel

`accessoryView` (page 1167)
Returns the receiver's accessory view (used to customize the receiver).

`setAccessoryView` (page 1169)
Adds an NSView to the receiver.

`jobStyleHint` (page 1168)
Returns the type of content that the Print panel is representing.

`setJobStyleHint` (page 1169)

Running the Panel

`beginSheetWithPrintInfo` (page 1167)
Presents a print sheet for *`printInfo`*, document-modal relative to *`docWindow`*.

`runModal` (page 1168)
Displays the receiver and begins the modal loop.

Communicating with the NSPrintInfo Object

`updateFromPrintInfo` (page 1169)
Reads the receiver's values from the NSPrintInfo object belonging to the current NSPrintOperation and updates the receiver accordingly.

`finalWritePrintInfo` (page 1167)
Writes the values of the receiver's printing attributes to the NSPrintInfo object belonging to the current NSPrintOperation.

Deprecated Methods

`pickedButton` (page 1168)

`pickedAllPages` (page 1168)

`pickedLayoutList` (page 1168)

Constructors

NSPrintPanel

Creates an empty NSPrintPanel.

```
public NSPrintPanel()
```

Static Methods

printPanel

Returns a newly created NSPrintPanel object.

```
public static NSPrintPanel printPanel()
```

Instance Methods

accessoryView

Returns the receiver's accessory view (used to customize the receiver).

```
public NSView accessoryView()
```

See Also
`setAccessoryView` (page 1169)

beginSheetWithPrintInfo

Presents a print sheet for *printInfo*, document-modal relative to *docWindow*.

```
public void beginSheetWithPrintInfo(NSPrintInfo printInfo, NSWindow docWindow,
    Object modalDelegate, NSSelector didEndSelector, Object contextInfo)
```

Discussion
When the modal session ends, if neither *modalDelegate* nor *didEndSelector* is `null`, *didEndSelector* is invoked on *modalDelegate*, passing *contextInfo*, among others, as an argument. *modalDelegate* is not the same as a delegate assigned to the panel. Modal delegates in sheets are temporary and the relationship only lasts until the sheet is dismissed.

The *didEndSelector* argument must have the following signature:

```
public void printPanelDidEnd (NSPrintPanel printPanel, int returnCode,  void
contextInfo)
```

The value passed as *returnCode* is either `NSPanel.CancelButton` or `NSPanel.OKButton`. `NSPanel.OKButton` is returned even if the user clicked the Preview button.

finalWritePrintInfo

Writes the values of the receiver's printing attributes to the NSPrintInfo object belonging to the current NSPrintOperation.

```
public void finalWritePrintInfo()
```

Discussion

Do not invoke this method directly—it is invoked automatically when the Print panel is dismissed by the user clicking the OK button.

See Also

`updateFromPrintInfo` (page 1169)

`currentOperation` (page 1154) (NSPrintOperation)

jobStyleHint

Returns the type of content that the Print panel is representing.

```
public String jobStyleHint()
```

Discussion

Returns `null` if no job style hint has been set.

Availability

Available in Mac OS X v10.2 and later.

See Also

`setJobStyleHint` (page 1169)

pickedAllPages

```
public void pickedAllPages(Object sender)
```

Discussion

Deprecated.

pickedButton

```
public void pickedButton(Object sender)
```

Discussion

Deprecated.

pickedLayoutList

```
public void pickedLayoutList(Object sender)
```

Discussion

Deprecated.

runModal

Displays the receiver and begins the modal loop.

```
public int runModal()
```

Discussion
Returns `NSPanel.CancelButton` if the user clicks the Cancel button; otherwise returns `NSPanel.OKButton`.

setAccessoryView

Adds an NSView to the receiver.

```
public void setAccessoryView(NSView aView)
```

Discussion
Invoke this method to add a custom view containing your controls. If you invoke this method prior to displaying the receiver, an item whose title is the same as your application's name is added to the pane-selection pull-down menu on the Print panel. When the user selects this item, your accessory view is shown. This method can be invoked repeatedly to change the accessory view depending on the situation. If *aView* is `null`, the receiver's current accessory view is removed.

See Also
`accessoryView` (page 1167)

setJobStyleHint

```
public void setJobStyleHint(String hint)
```

Discussion
Sets the type of content that the Print panel is representing. This controls the set of items that appear in the Presets menu of the simplified Print panel interface. The supported job style hints are described in "Constants" (page 1170). If *hint* is `null`, the standard interface is used.

Availability
Available in Mac OS X v10.2 and later.

See Also
`jobStyleHint` (page 1168)

updateFromPrintInfo

Reads the receiver's values from the NSPrintInfo object belonging to the current NSPrintOperation and updates the receiver accordingly.

```
public void updateFromPrintInfo()
```

Discussion
Do not invoke this method directly—it is invoked automatically before the Print panel is displayed.

See Also
`finalWritePrintInfo` (page 1167)
`currentOperation` (page 1154) (NSPrintOperation)

Constants

The following constant can be passed to `setJobStyleHint` (page 1169) to activate the simplified Print panel interface and identify which presets to present.

Constant	Description
`PrintPhotoJobStyleHint`	Output is of photographic data.

NSProgressIndicator

Inherits from	NSView : NSResponder : NSObject
Implements	NSCoding (NSResponder)
Package:	com.apple.cocoa.application
Companion guide	Progress Indicators

Overview

NSProgressIndicator causes an application to display a progress indicator to show that a lengthy task is under way. Some progress indicators are indeterminate and do nothing more than spin to show that the application is busy. Others are determinate and show the percentage of the task that has been completed.

Tasks

Constructors

`NSProgressIndicator` (page 1173)
Creates an empty NSProgressIndicator with a zero-sized frame rectangle.

Animating the Progress Indicator

`animate` (page 1173)
This action method advances the progress animation of an indeterminate progress animator by one step.

`animationDelay` (page 1174)
Returns the delay, in seconds, between animation steps for an indeterminate progress indicator.

`setAnimationDelay` (page 1176)
Sets the delay, in seconds, between animation steps for an indeterminate progress indicator to *`delay`*.

`setUsesThreadedAnimation` (page 1179)
Sets whether the receiver implements animation of the progress indicator in a separate thread to *`flag`*.

`startAnimation` (page 1179)
This action method starts the animation of an indeterminate progress indicator, which causes the barber pole to start spinning.

`stopAnimation` (page 1180)
: This action method stops the animation of an indeterminate progress indicator, which causes the barber pole to stop spinning.

`usesThreadedAnimation` (page 1180)
: Returns whether the receiver implements the animation of the progress indicator in a separate thread.

Advancing the Progress Bar

`incrementBy` (page 1175)
: Advances the progress bar of a determinate progress indicator by *`delta`*.

`setDoubleValue` (page 1178)
: Sets the value that indicates the current extent of the receiver to *`doubleValue`*.

`doubleValue` (page 1175)
: Returns a value that indicates the current extent of the progress bar of a determinate progress indicator.

`setMinValue` (page 1178)
: Sets the minimum value for the receiver to *`newMinimum`*.

`minValue` (page 1176)
: Returns the minimum value for the progress bar of a determinate progress indicator.

`setMaxValue` (page 1178)
: Sets the maximum value for the receiver to *`newMaximum`*.

`maxValue` (page 1176)
: Returns the maximum value for the progress bar of a determinate progress indicator.

Setting the Appearance

`setControlSize` (page 1177)
: Sets the size of the receiver.

`controlSize` (page 1174)
: Returns the size of the receiver.

`setControlTint` (page 1177)
: Sets the receiver's control tint.

`controlTint` (page 1174)
: Returns the receiver's control tint.

`setBezeled` (page 1177)
: Sets whether the receiver's frame has a three-dimensional bezel to *`flag`*.

`isBezeled` (page 1175)
: Returns `true` if the receiver's frame has a three-dimensional bezel.

`setIndeterminate` (page 1178)
: Sets whether the receiver is indeterminate to *`flag`*, if `style` (page 1180) returns `ProgressIndicatorBarStyle`.

`isIndeterminate` (page 1176)
: Returns `true` if the receiver is indeterminate, and `style` (page 1180) returns `ProgressIndicatorBarStyle`.

`setStyle` (page 1179)
  Sets whether to use a bar indicator or spinning.

`style` (page 1180)
  Returns whether the receiver is a bar indicator or spinning.

`sizeToFit` (page 1179)
  This action method resizes the receiver to an appropriate size depending on what `style` (page 1180) returns.

`setDisplayedWhenStopped` (page 1177)
  Sets whether the receiver hides itself when it isn't animating. By default, `isDisplayedWhenStopped` returns `true` if `style` (page 1180) is `ProgressIndicatorBarStyle`, and `isDisplayedWhenStopped` (page 1175) returns `false` if `style` (page 1180) is `ProgressIndicatorSpinningStyle`.

`isDisplayedWhenStopped` (page 1175)
  Returns `true` if the receiver shows itself even when it's not animating. By default, this returns `true` if `style` (page 1180) is `ProgressIndicatorBarStyle`, and this returns `false` if `style` (page 1180) is `ProgressIndicatorSpinningStyle`.

Constructors

NSProgressIndicator

Creates an empty NSProgressIndicator with a zero-sized frame rectangle.

```
public NSProgressIndicator()
```

Creates an NSProgressIndicator with *frameRect* as its frame rectangle

```
public NSProgressIndicator(NSRect frameRect)
```

Discussion

.

It's usually more convenient to use Interface Builder, which allows you to create an NSProgressIndicator and embed it in the superview of your choice.

Instance Methods

animate

This action method advances the progress animation of an indeterminate progress animator by one step.

```
public void animate(Object sender)
```

Discussion
Your application uses this method to control animation directly (as opposed to invoking `startAnimation` (page 1179) and `stopAnimation` (page 1180) for automatic animation). The more often you invoke `animate`, the faster the animation progresses. Determinate progress indicators do not use the same animation method; therefore, this method does nothing for a determinate progress indicator.

The `animate` method only invalidates the progress indicator, so it will be redrawn the next time through the event loop. To ensure immediate redrawing, invoke the `displayIfNeeded` (page 1748) method.

See Also
`animationDelay` (page 1174)
`setAnimationDelay` (page 1176)

animationDelay

Returns the delay, in seconds, between animation steps for an indeterminate progress indicator.

```
public double animationDelay()
```

Discussion
By default, the animation delay is set to 1/12 of a second (5.0/60.0). A determinate progress indicator does not use the animation delay value.

See Also
`animate` (page 1173)

controlSize

Returns the size of the receiver.

```
public int controlSize()
```

Discussion
Valid return values are described in "Constants" (page 1180).

See Also
`setControlSize` (page 1177)

controlTint

Returns the receiver's control tint.

```
public int controlTint()
```

Discussion
Valid return values are described in "Constants" (page 1180).

See Also
`setControlTint` (page 1177)

doubleValue

Returns a value that indicates the current extent of the progress bar of a determinate progress indicator.

```
public double doubleValue()
```

Discussion

For example, a determinate progress indicator goes from 0.0 to 100.0 by default. If the progress bar has advanced halfway across the view, the value returned by `doubleValue` would be 50.0. An indeterminate progress indicator does not use this value.

See Also

`incrementBy` (page 1175)
`setDoubleValue` (page 1178)

incrementBy

Advances the progress bar of a determinate progress indicator by *`delta`*.

```
public void incrementBy(double delta)
```

Discussion

For example, if you want to advance a progress bar from 0.0 to 100.0 in 20 steps, you would invoke `incrementBy` (page 1175) 20 times with a delta value of 5.0.

See Also

`doubleValue` (page 1175)

isBezeled

Returns `true` if the receiver's frame has a three-dimensional bezel.

```
public boolean isBezeled()
```

See Also

`setBezeled` (page 1177)

isDisplayedWhenStopped

Returns `true` if the receiver shows itself even when it's not animating. By default, this returns `true` if `style` (page 1180) is `ProgressIndicatorBarStyle`, and this returns `false` if `style` (page 1180) is `ProgressIndicatorSpinningStyle`.

```
public boolean isDisplayedWhenStopped()
```

Availability

Available in Mac OS X v10.2 and later.

See Also

`setDisplayedWhenStopped` (page 1177)

isIndeterminate

Returns `true` if the receiver is indeterminate, and `style` (page 1180) returns `ProgressIndicatorBarStyle`.

```
public boolean isIndeterminate()
```

Discussion

If `style` (page 1180) returns `ProgressIndicatorSpinningStyle`., the indicator is always indeterminate, regardless of what this method returns.

A determinate indicator displays how much of the task has been completed. An indeterminate indicator shows simply that the application is busy.

See Also

`setIndeterminate` (page 1178)

maxValue

Returns the maximum value for the progress bar of a determinate progress indicator.

```
public double maxValue()
```

Discussion

By default, a determinate progress indicator goes from 0.0 to 100.0, so the value returned would be 100.0. An indeterminate progress indicator does not use this value.

See Also

`minValue` (page 1176)

`setMaxValue` (page 1178)

minValue

Returns the minimum value for the progress bar of a determinate progress indicator.

```
public double minValue()
```

Discussion

By default, a determinate progress indicator goes from 0.0 to 100.0, so the value returned would be 0.0. An indeterminate progress indicator does not use this value.

See Also

`maxValue` (page 1176)

`setMinValue` (page 1178)

setAnimationDelay

Sets the delay, in seconds, between animation steps for an indeterminate progress indicator to *`delay`*.

```
public void setAnimationDelay(double delay)
```

Discussion

By default, the animation delay is set to 1/12 of a second (5.0/60.0). Setting the delay to a `double` value larger than 5.0/60.0 slows the animation, while setting the delay to a smaller value speeds it up. A determinate progress indicator does not use the animation delay value.

setBezeled

Sets whether the receiver's frame has a three-dimensional bezel to *flag*.

```
public void setBezeled(boolean flag)
```

See Also

`isBezeled` (page 1175)

setControlSize

Sets the size of the receiver.

```
public void setControlSize(int size)
```

Discussion

Valid values for *size* are described in "Constants" (page 1180).

See Also

`controlSize` (page 1174)

setControlTint

Sets the receiver's control tint.

```
public void setControlTint(int controlTint)
```

Discussion

Valid values for *controlTint* are described in "Constants" (page 1180).

See Also

`controlTint` (page 1174)

setDisplayedWhenStopped

Sets whether the receiver hides itself when it isn't animating. By default, `isDisplayedWhenStopped` returns `true` if `style` (page 1180) is `ProgressIndicatorBarStyle`, and `isDisplayedWhenStopped` (page 1175) returns `false` if `style` (page 1180) is `ProgressIndicatorSpinningStyle`.

```
public void setDisplayedWhenStopped(boolean isDisplayed)
```

Availability

Available in Mac OS X v10.2 and later.

See Also

`isDisplayedWhenStopped` (page 1175)

setDoubleValue

Sets the value that indicates the current extent of the receiver to *doubleValue*.

```
public void setDoubleValue(double doubleValue)
```

Discussion

An indeterminate progress indicator does not use this value.

See Also

`doubleValue` (page 1175)
`incrementBy` (page 1175)
`setMaxValue` (page 1178)
`setMinValue` (page 1178)

setIndeterminate

Sets whether the receiver is indeterminate to *flag*, if `style` (page 1180) returns `ProgressIndicatorBarStyle`.

```
public void setIndeterminate(boolean flag)
```

Discussion

If `style` (page 1180) returns `ProgressIndicatorSpinningStyle`, the indicator is always indeterminate, regardless of what you pass to this method.

See Also

`isIndeterminate` (page 1176)

setMaxValue

Sets the maximum value for the receiver to *newMaximum*.

```
public void setMaxValue(double newMaximum)
```

Discussion

An indeterminate progress indicator does not use this value.

See Also

`maxValue` (page 1176)

setMinValue

Sets the minimum value for the receiver to *newMinimum*.

```
public void setMinValue(double newMinimum)
```

Discussion

An indeterminate progress indicator does not use this value.

See Also

`minValue` (page 1176)

setStyle

Sets whether to use a bar indicator or spinning.

```
public void setStyle(int style)
```

Discussion
Possible values for *style* are described in "Constants" (page 1180).

Availability
Available in Mac OS X v10.2 and later.

See Also
`style` (page 1180)

setUsesThreadedAnimation

Sets whether the receiver implements animation of the progress indicator in a separate thread to *flag*.

```
public void setUsesThreadedAnimation(boolean flag)
```

Discussion
If the application becomes multithreaded as a result of an invocation of this method, the application's performance could become noticeably slower.

See Also
`usesThreadedAnimation` (page 1180)

sizeToFit

This action method resizes the receiver to an appropriate size depending on what `style` (page 1180) returns.

```
public void sizeToFit()
```

Discussion
Use this after you use `setStyle` (page 1179) to re-size the receiver.

Availability
Available in Mac OS X v10.2 and later.

startAnimation

This action method starts the animation of an indeterminate progress indicator, which causes the barber pole to start spinning.

```
public void startAnimation(Object sender)
```

Discussion
Does nothing for a determinate progress indicator.

See Also
`animationDelay` (page 1174)

`stopAnimation` (page 1180)

stopAnimation

This action method stops the animation of an indeterminate progress indicator, which causes the barber pole to stop spinning.

```
public void stopAnimation(Object sender)
```

Discussion
Does nothing for a determinate progress indicator.

See Also
`animationDelay` (page 1174)
`startAnimation` (page 1179)

style

Returns whether the receiver is a bar indicator or spinning.

```
public int style()
```

Discussion
Possible return values are described in "Constants" (page 1180).

Availability
Available in Mac OS X v10.2 and later.

See Also
`setStyle` (page 1179)

usesThreadedAnimation

Returns whether the receiver implements the animation of the progress indicator in a separate thread.

```
public boolean usesThreadedAnimation()
```

See Also
`setUsesThreadedAnimation` (page 1179)

Constants

These constants specify a cell's tint. They're used by `controlTint` (page 1174) and `setControlTint` (page 1177).

Constant	Description
`DefaultControlTint`	The current default tint setting.

Constant	Description
ClearControlTint	Clear control tint.
BlueControlTint	Aqua control tint
GraphiteControlTint	Graphite control tint

These constants specify a cell's size. They're used by controlSize (page 1174) and setControlSize (page 1177).

Constant	Description
RegularControlSize	A regular sized control
SmallControlSize	A small-sized control for use when space is tight.
MiniControlSize	The control has a smaller size than SmallControlSize.

The following constants specify the progress indicator's style and are used by style (page 1180) and setStyle (page 1179):

Constant	Description
ProgressIndicatorBarStyle	A rectangular indicator that can be determinate or indeterminate.
ProgressIndicatorSpinningStyle	A small square indicator that can be indeterminate only .

NSPureApplication

Inherits from NSObject

Package: com.apple.cocoa.application

Overview

NSPureApplication can be used to start a Java-based NSApplication from the `java` command-line tool. It contains one static method, `main` (page 1184), which performs the actual work.

Tasks

Constructors

`NSPureApplication` (page 1183)
Constructor for NSPureApplication. There's no reason to instantiate NSPureApplication, since it contains neither instance data nor instance methods.

Starting a Java Application

`main` (page 1184)

Constructors

NSPureApplication

Constructor for NSPureApplication. There's no reason to instantiate NSPureApplication, since it contains neither instance data nor instance methods.

```
public NSPureApplication()
```

Static Methods

main

```
public static void main(String[] args)
```

Discussion

The main method that starts an NSPureApplication using *args* as the argument list.

NSResponder

Inherits from NSObject

Implements NSCoding

Package: com.apple.cocoa.application

Companion guide Cocoa Event-Handling Guide

Overview

NSResponder is an abstract class that forms the basis of event and command processing in the Application Kit. The core classes—NSApplication, NSWindow, and NSView—inherit from NSResponder, as must any class that handles events. The responder model is built around three components: event messages, action messages, and the responder chain.

Starting with Mac OS X v10.4, NSResponder plays an important role in the presentation of error information. The default implementations of the `presentError` (page 1195) and `presentErrorModalForWindow` (page 1195) methods send `willPresentError` (page 1200) to `self`, thereby giving subclasses the opportunity to customize the localized information presented in error alerts. NSResponder then forwards the message to the next responder, passing it the customized NSError object. The exact path up the modified responder chain depends on the type of application window:

- Windows owned by document: view to superviews to window to window controller to document object to document controller to the application object
- Windows with window controllers but no documents: view to superviews to window to window controller to the application object
- Windows with no window controllers: view to superviews to window to the application object

Tasks

Constructors

`NSResponder` (page 1189)
Creates an empty NSResponder.

Changing the First Responder

acceptsFirstResponder (page 1189)
Overridden by subclasses to return true if the receiver can handle key events and action messages sent up the responder chain.

becomeFirstResponder (page 1190)
Notifies the receiver that it's about to become first responder in its NSWindow.

resignFirstResponder (page 1196)
Notifies the receiver that it's been asked to relinquish its status as first responder in its NSWindow.

Setting the Next Responder

setNextResponder (page 1198)
Sets the receiver's next responder to *aResponder*.

nextResponder (page 1193)
Returns the receiver's next responder, or null if it has none.

Event Methods

mouseDown (page 1192)
Informs the receiver that the user has pressed the left mouse button specified by *theEvent*.

mouseDragged (page 1192)
Informs the receiver that the user has moved the mouse with the left button pressed specified by *theEvent*.

mouseUp (page 1193)
Informs the receiver that the user has released the left mouse button specified by *theEvent*.

mouseMoved (page 1193)
Informs the receiver that the mouse has moved specified by *theEvent*.

mouseEntered (page 1192)
Informs the receiver that the cursor has entered a tracking rectangle specified by *theEvent*.

mouseExited (page 1193)
Informs the receiver that the cursor has exited a tracking rectangle.

rightMouseDown (page 1196)
Informs the receiver that the user has pressed the right mouse button specified by *theEvent*.

rightMouseDragged (page 1197)
Informs the receiver that the user has moved the mouse with the right button pressed specified by *theEvent*.

rightMouseUp (page 1197)
Informs the receiver that the user has released the right mouse button specified by *theEvent*.

otherMouseDown (page 1194)
Informs the receiver that the user has pressed a mouse button other than left or right specified by *theEvent*.

otherMouseDragged (page 1194)
: Informs the receiver that the user has moved the mouse with a button other than the left or right button pressed specified by *theEvent*.

otherMouseUp (page 1194)
: Informs the receiver that the user has released a mouse button other than the left or right specified by *theEvent*.

scrollWheel (page 1197)
: Informs the receiver that the mouse's scroll wheel has moved specified by *theEvent*.

keyDown (page 1191)
: Informs the receiver that the user has pressed a key.

keyUp (page 1192)
: Informs the receiver that the user has released a key event specified by *theEvent*.

flagsChanged (page 1190)
: Informs the receiver that the user has pressed or released a modifier key (Shift, Control, and so on) specified by *theEvent*.

helpRequested (page 1191)
: Displays context-sensitive help for the receiver if such exists; otherwise passes this message to the next responder.

tabletPoint (page 1198)
: Informs the receiver that tablet-point event *theEvent* has occurred.

tabletProximity (page 1199)
: Informs the receiver that the tablet-proximity event *theEvent* has occurred.

Special Key Event Methods

interpretKeyEvents (page 1191)
: Invoked by subclasses from their keyDown (page 1191) method to handle a series of key events.

performKeyEquivalent (page 1194)
: Overridden by subclasses to handle a key equivalent.

performMnemonic (page 1195)
: Overridden by subclasses to handle a mnemonic.

Clearing Key Events

flushBufferedKeyEvents (page 1190)
: Overridden by subclasses to clear any unprocessed key events.

Action Methods

showContextHelp (page 1198)
: Implemented by subclasses to invoke the help system, displaying information relevant to the receiver and its current state. The *sender* argument is typically the object that invoked this method.

Dispatch Methods

`tryToPerform` (page 1199)
: Attempts to perform the action method indicated by *`anAction`*.

Terminating the Responder Chain

`noResponderForSelector` (page 1193)
: Handles the case where an event or action message falls off the end of the responder chain.

Services Menu Updating

`validRequestorForTypes` (page 1200)
: Overridden by subclasses to determine what services are available.

Setting the Menu

`setMenu` (page 1197)
: Sets the receiver's menu to *`aMenu`*.

`menu` (page 1192)
: Returns the receiver's menu.

Setting the Interface Style

`setInterfaceStyle` (page 1197)
: Sets the receiver's style to the style specified by *`interfaceStyle`*, such as `NSInterfaceStyle.MacintoshInterfaceStyle` or `NSInterfaceStyle.Windows95InterfaceStyle`.

`interfaceStyle` (page 1191)
: Returns the receiver's interface style.

Testing Events

`shouldBeTreatedAsInkEvent` (page 1198)
: Returns `true` if *`theEvent`* should be treated as an ink event, `false` if *`theEvent`* should be treated as a mouse event.

Getting the Undo Manager

`undoManager` (page 1199)
: Returns the undo manager for this responder.

Presenting and Customizing Error Information

`presentError` (page 1195)
Presents an error alert to the user as an application-modal dialog.

`presentErrorModalForWindow` (page 1195)
Presents an error alert to the user as a document-modal sheet attached to document window.

`willPresentError` (page 1200)
Implemented by subclasses to return a custom version of error object *anError* that is more suitable for presentation in alert sheets and dialogs.

Binding

`bind` (page 1190)
Creates a relationship between the receiver's *binding* and the property of *observableController* specified by *keyPath*.

Constructors

NSResponder

Creates an empty NSResponder.

```
public NSResponder()
```

Instance Methods

acceptsFirstResponder

Overridden by subclasses to return `true` if the receiver can handle key events and action messages sent up the responder chain.

```
public boolean acceptsFirstResponder()
```

Discussion
NSResponder's implementation returns `false`, indicating that by default a responder object doesn't agree to become first responder. Objects that aren't first responder can receive mouse-down messages, but no other event or action messages.

See Also
`becomeFirstResponder` (page 1190)
`resignFirstResponder` (page 1196)
`needsPanelToBecomeKey` (page 1761) (NSView)

becomeFirstResponder

Notifies the receiver that it's about to become first responder in its NSWindow.

```
public boolean becomeFirstResponder()
```

Discussion

NSResponder's implementation returns `true`, accepting first responder status. Subclasses can override this method to update state or perform some action such as highlighting the selection, or to return `false`, refusing first responder status.

Use NSWindow's `makeFirstResponder` (page 1841), not this method, to make an object the first responder. Never invoke this method directly.

See Also

`resignFirstResponder` (page 1196)

`acceptsFirstResponder` (page 1189)

bind

Creates a relationship between the receiver's *`binding`* and the property of *`observableController`* specified by *`keyPath`*.

```
public void bind(String binding, Object observableController, String keyPath,
    NSDictionary options)
```

Discussion

The *`binding`* is the key path for a property of the receiver previously exposed. The *`options`* dictionary is optional. If present, it contains placeholder objects or an NSValueTransformer identifier as described in "Constants" (page 2005).

Availability

Available in Mac OS X v10.4 and later.

flagsChanged

Informs the receiver that the user has pressed or released a modifier key (Shift, Control, and so on) specified by *`theEvent`*.

```
public void flagsChanged(NSEvent theEvent)
```

Discussion

NSResponder's implementation simply passes this message to the next responder.

flushBufferedKeyEvents

Overridden by subclasses to clear any unprocessed key events.

```
public void flushBufferedKeyEvents()
```

helpRequested

Displays context-sensitive help for the receiver if such exists; otherwise passes this message to the next responder.

```
public void helpRequested(NSEvent theEvent)
```

Discussion

NSWindow invokes this method automatically when the user clicks for help—while processing *theEvent*. Subclasses need not override this method, and application code shouldn't directly invoke it.

See Also

`showContextHelp` (page 1198)

interfaceStyle

Returns the receiver's interface style.

```
public int interfaceStyle()
```

Discussion

`interfaceStyle` is an abstract method in NSResponder and just returns `NSInterfaceStyle.NoInterfaceStyle`. It is overridden in classes such as NSWindow and NSView to return the interface style, such as `NSInterfaceStyle.MacintoshInterfaceStyle`. A responder's style (if other than `NSInterfaceStyle.NoInterfaceStyle`) overrides all other settings, such as those established by the defaults system.

See Also

`setInterfaceStyle` (page 1197)

interpretKeyEvents

Invoked by subclasses from their `keyDown` (page 1191) method to handle a series of key events.

```
public void interpretKeyEvents(NSArray eventArray)
```

Discussion

This method sends the character input in *eventArray* to the system input manager for interpretation as text to insert or commands to perform. Subclasses shouldn't override this method.

See the NSInputManager (page 801) and NSTextInput (page 2025) class and interface specifications for more information on input management.

keyDown

Informs the receiver that the user has pressed a key.

```
public void keyDown(NSEvent theEvent)
```

Discussion

The receiver can interpret *theEvent* itself, or pass it to the system input manager using `interpretKeyEvents` (page 1191). NSResponder's implementation simply passes this message to the next responder.

keyUp

Informs the receiver that the user has released a key event specified by *theEvent*.

```
public void keyUp(NSEvent theEvent)
```

Discussion

NSResponder's implementation simply passes this message to the next responder.

menu

Returns the receiver's menu.

```
public NSMenu menu()
```

Discussion

For NSApplication this menu is the same as the menu returned by its `mainMenu` (page 113) method.

See Also

`setMenu` (page 1197)

`menuForEvent` (page 1760) (NSView)

`defaultMenu` (page 1738) (NSView)

mouseDown

Informs the receiver that the user has pressed the left mouse button specified by *theEvent*.

```
public void mouseDown(NSEvent theEvent)
```

Discussion

NSResponder's implementation simply passes this message to the next responder.

mouseDragged

Informs the receiver that the user has moved the mouse with the left button pressed specified by *theEvent*.

```
public void mouseDragged(NSEvent theEvent)
```

Discussion

NSResponder's implementation simply passes this message to the next responder.

mouseEntered

Informs the receiver that the cursor has entered a tracking rectangle specified by *theEvent*.

```
public void mouseEntered(NSEvent theEvent)
```

Discussion

NSResponder's implementation simply passes this message to the next responder.

mouseExited

Informs the receiver that the cursor has exited a tracking rectangle.

```
public void mouseExited(NSEvent theEvent)
```

Discussion

NSResponder's implementation simply passes this message to the next responder.

mouseMoved

Informs the receiver that the mouse has moved specified by *theEvent*.

```
public void mouseMoved(NSEvent theEvent)
```

Discussion

NSResponder's implementation simply passes this message to the next responder.

See Also

setAcceptsMouseMovedEvents (page 1853) (NSWindow)

mouseUp

Informs the receiver that the user has released the left mouse button specified by *theEvent*.

```
public void mouseUp(NSEvent theEvent)
```

Discussion

NSResponder's implementation simply passes this message to the next responder.

nextResponder

Returns the receiver's next responder, or null if it has none.

```
public NSResponder nextResponder()
```

See Also

setNextResponder (page 1198)
noResponderForSelector (page 1193)

noResponderForSelector

Handles the case where an event or action message falls off the end of the responder chain.

```
public void noResponderForSelector(NSSelector eventSelector)
```

Discussion

NSResponder's implementation beeps if *eventSelector* is `keyDown` (page 1191).

otherMouseDown

Informs the receiver that the user has pressed a mouse button other than left or right specified by *theEvent*.

```
public void otherMouseDown(NSEvent theEvent)
```

Discussion

NSResponder's implementation simply passes this message to the next responder.

otherMouseDragged

Informs the receiver that the user has moved the mouse with a button other than the left or right button pressed specified by *theEvent*.

```
public void otherMouseDragged(NSEvent theEvent)
```

Discussion

NSResponder's implementation simply passes this message to the next responder.

otherMouseUp

Informs the receiver that the user has released a mouse button other than the left or right specified by *theEvent*.

```
public void otherMouseUp(NSEvent theEvent)
```

Discussion

NSResponder's implementation simply passes this message to the next responder.

performKeyEquivalent

Overridden by subclasses to handle a key equivalent.

```
public boolean performKeyEquivalent(NSEvent theEvent)
```

Discussion

If the character code or codes in *theEvent* match the receiver's key equivalent, the receiver should respond to the event and return `true`. NSResponder's implementation does nothing and returns `false`.

Note: `performKeyEquivalent` (page 1194) takes an NSEvent as its argument, while `performMnemonic` (page 1195) takes a String containing the uninterpreted characters of the key event. You should extract the characters for a key equivalent using NSEvent's `charactersIgnoringModifiers` (page 612).

See Also

`performKeyEquivalent` (page 1763) (NSView)
`performKeyEquivalent` (page 259) (NSButton)

performMnemonic

Overridden by subclasses to handle a mnemonic.

```
public boolean performMnemonic(String aString)
```

Discussion
If the character code or codes in *aString* match the receiver's mnemonic, the receiver should perform the mnemonic and return `true`. NSResponder's implementation does nothing and returns `false`. Mnemonics are not supported in Mac OS X.

See Also
`performMnemonic` (page 1763) (NSView)

presentError

Presents an error alert to the user as an application-modal dialog.

```
public boolean presentError(NSError anError)
```

Discussion
The alert displays information found in the NSError object *anError*; this information can include error description, recovery suggestion, failure reason, and button titles (all localized). The method returns `true` if error recovery succeeded and `false` otherwise.

The default implementation of this method sends `willPresentError` (page 1200) to `self`. By doing this, NSResponder gives subclasses an opportunity to customize error presentation. It then forwards the message, passing any customized error object, to the next responder; if there is no next responder, it passes the error object to `NSApp`, which displays a document-modal error alert. When the user dismisses the alert, any recovery attempter associated with the error object is given a chance to recover from the error. See the class description for the precise route up the responder chain (plus document and controller objects) this message might travel.

It is not recommended that you attempt to override this method. If you wish to customize the error presentation, override `willPresentError` (page 1200) instead.

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

See Also
`presentErrorModalForWindow` (page 1195)

presentErrorModalForWindow

Presents an error alert to the user as a document-modal sheet attached to document window.

```
public void presentErrorModalForWindow(NSError error, NSWindow aWindow, Object
    delegate, NSSelector didPresentSelector, Object contextInfo)
```

Discussion
The *aWindow* parameter represents the window. The information displayed in the alert is extracted from the NSError object *error*; it may include a description, recovery suggestion, failure reason, and button titles (all localized). Once the user dismisses the alert and any recovery attempter associated with the error object has

had a chance to recover from it, the receiver sends a message identified by *didPresentSelector* to the modal delegate *delegate*. (A recovery attempter is an object that conforms to the NSErrorRecoveryAttempting informal protocol.) The *didPresentSelector* selector must have the signature:

```
void didPresentErrorWithRecovery(boolean didRecover, Object contextInfo
```

The modal delegate implements this method to perform any post-error processing if recovery failed or was not attempted (that is, *didRecover* is `NO`). Any supplemental data is passed to the modal delegate via *contextInfo*.

The default implementation of this method sends `willPresentError` (page 1200) to `self`. By doing this, NSResponder gives subclasses an opportunity to customize error presentation. It then forwards the message, passing any customized error, to the next responder or; if there is no next responder, it passes the error object to `NSApp`, which displays a document-modal error alert. When the user dismisses the alert, any recovery attempter associated with the error object is given a chance to recover from the error. See the class description for the precise route up the responder chain (plus document and controller objects) this message might travel.

It is not recommended that you attempt to override this method. If you wish to customize the error presentation, override `willPresentError` (page 1200) instead.

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

See Also
`presentError` (page 1195)

resignFirstResponder

Notifies the receiver that it's been asked to relinquish its status as first responder in its NSWindow.

```
public boolean resignFirstResponder()
```

Discussion
NSResponder's implementation returns `true`, resigning first responder status. Subclasses can override this method to update state or perform some action such as unhighlighting the selection, or to return `false`, refusing to relinquish first responder status.

Use NSWindow's `makeFirstResponder` (page 1841), not this method, to make an object the first responder. Never invoke this method directly.

See Also
`becomeFirstResponder` (page 1190)
`acceptsFirstResponder` (page 1189)

rightMouseDown

Informs the receiver that the user has pressed the right mouse button specified by *theEvent*.

```
public void rightMouseDown(NSEvent theEvent)
```

Discussion
NSResponder's implementation simply passes this message to the next responder.

rightMouseDragged

Informs the receiver that the user has moved the mouse with the right button pressed specified by *theEvent*.

```
public void rightMouseDragged(NSEvent theEvent)
```

Discussion
NSResponder's implementation simply passes this message to the next responder.

rightMouseUp

Informs the receiver that the user has released the right mouse button specified by *theEvent*.

```
public void rightMouseUp(NSEvent theEvent)
```

Discussion
NSResponder's implementation simply passes this message to the next responder.

scrollWheel

Informs the receiver that the mouse's scroll wheel has moved specified by *theEvent*.

```
public void scrollWheel(NSEvent theEvent)
```

Discussion
NSResponder's implementation simply passes this message to the next responder.

setInterfaceStyle

Sets the receiver's style to the style specified by *interfaceStyle*, such as `NSInterfaceStyle.MacintoshInterfaceStyle` or `NSInterfaceStyle.Windows95InterfaceStyle`.

```
public void setInterfaceStyle(int interfaceStyle)
```

Discussion
`setInterfaceStyle` is an abstract method in NSResponder, but is overridden in classes such as NSWindow and NSView to actually set the interface style. You should almost never need to invoke or override this method, but if you do override it, your version should always invoke the implementation in `super`.

See Also
`interfaceStyle` (page 1191)

setMenu

Sets the receiver's menu to *aMenu*.

```
public void setMenu(NSMenu aMenu)
```

Discussion
If the receiver is an NSApplication object, this method sets the main menu, typically set using `setMainMenu` (page 122).

See Also

`menu` (page 1192)

setNextResponder

Sets the receiver's next responder to *aResponder*.

```
public void setNextResponder(NSResponder aResponder)
```

See Also

`nextResponder` (page 1193)

shouldBeTreatedAsInkEvent

Returns `true` if *theEvent* should be treated as an ink event, `false` if *theEvent* should be treated as a mouse event.

```
public boolean shouldBeTreatedAsInkEvent(NSEvent theEvent)
```

Discussion

This provides the ability to distinguish when a pen-down should start inking versus when a pen-down should be treated as a mouse down event. This allows for a write-anywhere model for pen-based input.

The default implementation in NSApplication sends the method to the NSWindow under the pen. If the window is inactive, this method returns `true`, unless the pen-down is in the window drag region. If the window is active, this method is sent to the NSView under the pen.

The default implementation in NSView returns `true`, and NSControl overrides and returns `false`. This allows write-anywhere over most NSViews, but allows the pen to be used to track in controls and to move windows.

A custom view should override this method to get the correct behavior for a pen-down in the view.

Availability

Available in Mac OS X v10.2 and later.

showContextHelp

Implemented by subclasses to invoke the help system, displaying information relevant to the receiver and its current state. The *sender* argument is typically the object that invoked this method.

```
public void showContextHelp(Object sender)
```

See Also

`helpRequested` (page 1191)

tabletPoint

Informs the receiver that tablet-point event *theEvent* has occurred.

```
public void tabletPoint(NSEvent theEvent)
```

Discussion

Tablet events are represented by NSEvent objects of type `NSTabletPoint`. They describe the current state of a transducer (that is, a pointing device) that is in proximity to its tablet, reflecting changes such as location, pressure, tilt, and rotation. See "NSEvent" (page 603) for the methods that allow you to extract this and other information from *theEvent*. The default implementation of NSResponder passes the message to the next responder.

Availability

Available in Mac OS X v10.4 or later.

See Also

`tabletProximity` (page 1199)

tabletProximity

Informs the receiver that the tablet-proximity event *theEvent* has occurred.

```
public void tabletProximity(NSEvent theEvent)
```

Discussion

Tablet events are represented by NSEvent objects of type `NSTabletProximity`. Tablet devices generate proximity events when the transducer (pointing device) nears a tablet and when it moves away from a tablet. From an event object of this type you can extract information about the kind of device and its capabilities, as well as the relation of this tablet-proximity event to various tablet-point events; see "NSEvent" (page 603) for details. The default implementation of NSResponder passes the message to the next responder.

Availability

Available in Mac OS X v10.4 or later.

See Also

`tabletPoint` (page 1198)

tryToPerform

Attempts to perform the action method indicated by *anAction*.

```
public boolean tryToPerform(NSSelector anAction, Object anObject)
```

Discussion

The method should take a single argument of type `Object` and return `void`. If the receiver responds to *anAction*, it invokes the method with *anObject* as the argument and returns `true`. If the receiver doesn't respond, it sends this message to its next responder with the same selector and object. Returns `false` if no responder is found that responds to *anAction*.

See Also

`sendActionToTargetFromSender` (page 121) (NSApplication)

undoManager

Returns the undo manager for this responder.

```
public NSUndoManager undoManager()
```

Discussion
NSResponder's implementation simply passes this message to the next responder.

validRequestorForTypes

Overridden by subclasses to determine what services are available.

```
public Object validRequestorForTypes(String sendType, String returnType)
```

Discussion
With each event, and for each service in the Services menu, the application object sends this message up the responder chain with the send and return type for the service being checked. This method is therefore invoked many times per event. If the receiver can place data of *sendType* on the pasteboard and receive data of *returnType*, it should return `this`; otherwise it should return either `super.validRequestorForTypes()` or `nextResponder().validRequestorForTypes()`, which allows an object higher up in the responder chain to have an opportunity to handle the message. NSResponder's implementation simply forwards this message to the next responder, ultimately returning `null`.

Either *sendType* or *returnType*—but not both—may be empty. If *sendType* is empty, the service doesn't require input from the application requesting the service. If *returnType* is empty, the service doesn't return data.

See Also
`registerServicesMenuTypes` (page 117) (NSApplication)

willPresentError

Implemented by subclasses to return a custom version of error object *anError* that is more suitable for presentation in alert sheets and dialogs.

```
public NSError willPresentError(NSError anError)
```

Discussion
The default implementation of this method simply returns *anError* unchanged. When overriding this method, you can examine *anError* and, if its localized description or recovery information is unhelpfully generic, return an error object with more specific localized text. If you do this, always use the domain and error code of the NSError object to distinguish between errors whose presentation you want to customize and those you do not. Don't make decisions based on the localized description, recovery suggestion, or recovery options because parsing localized text is problematic. If you decide not to customize the error presentation, return by sending this message to `super`.

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

See Also
`presentError` (page 1195)

NSRulerMarker

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.application
Companion guide	Rulers and Paragraph Styles

Overview

An NSRulerMarker displays a symbol on an NSRulerView, indicating a location for whatever graphics element it represents in the client of the NSRulerView (for example, a margin or tab setting, or the edges of a graphic on the page).

Tasks

Constructors

`NSRulerMarker` (page 1203)
Creates an empty NSRulerMarker.

Getting the Ruler View

`ruler` (page 1205)
Returns the NSRulerView the receiver belongs to.

Setting the Image

`setImage` (page 1206)
Sets the receiver's image to *anImage*.

`image` (page 1204)
Returns the NSImage object displayed by the receiver.

`setImageOrigin` (page 1206)
Sets the point in the receiver's image positioned at the receiver's location on the NSRulerView to *aPoint*.

`imageOrigin` (page 1204)
: Returns the point in the receiver's image positioned at the receiver's location on the NSRulerView, expressed in the image's coordinate system.

`imageRectInRuler` (page 1204)
: Returns the rectangle occupied by the receiver's image, in the NSRulerView's coordinate system, accounting for whether the NSRulerView's coordinate system is flipped.

`thicknessRequiredInRuler` (page 1207)
: Returns the amount of the receiver's image that's displayed above or to the left of the NSRulerView's baseline, the height for a horizontal ruler or width for a vertical ruler.

Setting Movability

`setMovable` (page 1206)
: Controls whether the user can move the receiver in its NSRulerView.

`isMovable` (page 1204)
: Returns `true` if the user can move the receiver on its NSRulerView, `false` otherwise.

`setRemovable` (page 1207)
: Controls whether the user can remove the receiver from its NSRulerView.

`isRemovable` (page 1205)
: Returns `true` if the user can remove the receiver from its NSRulerView, `false` otherwise.

Setting the Location

`setMarkerLocation` (page 1206)
: Sets the location of the receiver in the coordinate system of the NSRulerView's client view to *`location`*.

`markerLocation` (page 1205)
: Returns the location of the receiver in the coordinate system of the NSRulerView's client view.

Setting the Represented Object

`setRepresentedObject` (page 1207)
: Sets the object the receiver represents to *`anObject`*.

`representedObject` (page 1205)
: Returns the object the receiver represents, as explained in the class description.

Drawing and Event Handling

`drawRect` (page 1203)
: Draws the part of the receiver's image that intersects *`aRect`* in the NSRulerView's coordinate system.

`isDragging` (page 1204)
: Returns `true` if the receiver is being dragged, `false` otherwise.

`trackMouseToAddMarker` (page 1207)
: Handles user manipulation of the receiver in its NSRulerView specified by *`theEvent`*.

Constructors

NSRulerMarker

Creates an empty NSRulerMarker.

```
public NSRulerMarker()
```

Creates an NSRulerMarker object, associating it with (but not adding it to) *aRulerView* and assigning the attributes provided.

```
public NSRulerMarker(NSRulerView aRulerView, float location, NSImage anImage,
    NSPoint imageOrigin)
```

Discussion

location is the x or y position of the marker in the client view's coordinate system, depending on whether the NSRulerView is horizontal or vertical. *anImage* is the image displayed at the marker location, and *imageOrigin* is the point within the image positioned at the marker location, expressed in pixels relative to the lower-left corner of the image. This method throws an exception if *aRulerView* or *anImage* is `null`.

The image used to draw the marker must be appropriate for the orientation of the ruler. Markers may need to look different on a horizontal ruler than on a vertical ruler, and the NSRulerView neither scales nor rotates the images.

To add the new ruler marker to *aRulerView*, use either of NSRulerView's `addMarker` (page 1215) or `trackMarker` (page 1222) methods. `addMarker` (page 1215) immediately puts the marker on the ruler, while `trackMarker` (page 1222) allows the client view to intercede in the addition and placement of the marker.

A new ruler marker can be moved on its NSRulerView, but not removed. Use `setMovable` (page 1206) and `setRemovable` (page 1207) to change these attributes. The new ruler marker also has no represented object; use `setRepresentedObject` (page 1207) to provide or change it.

See Also

`setMarkerLocation` (page 1206)

`setImage` (page 1206)

`setImageOrigin` (page 1206)

Instance Methods

drawRect

Draws the part of the receiver's image that intersects *aRect* in the NSRulerView's coordinate system.

```
public void drawRect(NSRect aRect)
```

See Also

`imageRectInRuler` (page 1204)

image

Returns the NSImage object displayed by the receiver.

```
public NSImage image()
```

See Also

`setImage` (page 1206)

imageOrigin

Returns the point in the receiver's image positioned at the receiver's location on the NSRulerView, expressed in the image's coordinate system.

```
public NSPoint imageOrigin()
```

Discussion

For a horizontal ruler, the x coordinate of the image origin is aligned with the location of the marker, and the y coordinate lies on the baseline of the ruler. For vertical rulers, the y coordinate of the image origin is the location, and the x coordinate lies on the baseline.

See Also

`setImageOrigin` (page 1206)

`imageRectInRuler` (page 1204)

imageRectInRuler

Returns the rectangle occupied by the receiver's image, in the NSRulerView's coordinate system, accounting for whether the NSRulerView's coordinate system is flipped.

```
public NSRect imageRectInRuler()
```

See Also

`drawRect` (page 1203)

`thicknessRequiredInRuler` (page 1207)

isDragging

Returns `true` if the receiver is being dragged, `false` otherwise.

```
public boolean isDragging()
```

See Also

`trackMouseToAddMarker` (page 1207)

isMovable

Returns `true` if the user can move the receiver on its NSRulerView, `false` otherwise.

```
public boolean isMovable()
```

Discussion

NSRulerMarkers are by default movable.

See Also

`setMovable` (page 1206)

`isRemovable` (page 1205)

isRemovable

Returns `true` if the user can remove the receiver from its NSRulerView, `false` otherwise.

```
public boolean isRemovable()
```

Discussion

NSRulerMarkers cannot by default be removed from their NSRulerViews.

See Also

`setRemovable` (page 1207)

`isMovable` (page 1204)

markerLocation

Returns the location of the receiver in the coordinate system of the NSRulerView's client view.

```
public float markerLocation()
```

Discussion

This is an x position for a horizontal ruler, a y position for a vertical ruler.

See Also

`setMarkerLocation` (page 1206)

representedObject

Returns the object the receiver represents, as explained in the class description.

```
public Object representedObject()
```

See Also

`setRepresentedObject` (page 1207)

ruler

Returns the NSRulerView the receiver belongs to.

```
public NSRulerView ruler()
```

See Also

`addMarker` (page 1215) (NSRulerView)

setImage

Sets the receiver's image to *anImage*.

```
public void setImage(NSImage anImage)
```

See Also

`image` (page 1204)

`setImageOrigin` (page 1206)

setImageOrigin

Sets the point in the receiver's image positioned at the receiver's location on the NSRulerView to *aPoint*.

```
public void setImageOrigin(NSPoint aPoint)
```

Discussion

This point is always expressed in pixels relative to the lower-left corner of the image.

For a horizontal ruler, the x coordinate of the image origin is aligned with the location of the marker, and the y coordinate lies on the baseline of the ruler. For vertical rulers, the y coordinate of the image origin is the location, and the x coordinate lies on the baseline.

See Also

`imageOrigin` (page 1204)

`setImage` (page 1206)

`setMarkerLocation` (page 1206)

setMarkerLocation

Sets the location of the receiver in the coordinate system of the NSRulerView's client view to *location*.

```
public void setMarkerLocation(float location)
```

Discussion

This location is an x position for a horizontal ruler, a y position for a vertical ruler.

See Also

`markerLocation` (page 1205)

`setImageOrigin` (page 1206)

setMovable

Controls whether the user can move the receiver in its NSRulerView.

```
public void setMovable(boolean flag)
```

Discussion

If *flag* is `true`, the user can drag the marker image in the ruler. If *flag* is `false`, the receiver is immovable. NSRulerMarkers are by default movable.

See Also

isMovable (page 1204)

setRemovable (page 1207)

setRemovable

Controls whether the user can remove the receiver from its NSRulerView.

```
public void setRemovable(boolean flag)
```

Discussion

If *flag* is true, the user can drag the marker image off of the ruler. If *flag* is false, the receiver can't be removed. NSRulerMarkers are by default not removable.

See Also

isRemovable (page 1205)

setMovable (page 1206)

setRepresentedObject

Sets the object the receiver represents to *anObject*.

```
public void setRepresentedObject(Object anObject)
```

Discussion

See the class description for more information on the represented object.

See Also

representedObject (page 1205)

thicknessRequiredInRuler

Returns the amount of the receiver's image that's displayed above or to the left of the NSRulerView's baseline, the height for a horizontal ruler or width for a vertical ruler.

```
public float thicknessRequiredInRuler()
```

See Also

imageOrigin (page 1204)

trackMouseToAddMarker

Handles user manipulation of the receiver in its NSRulerView specified by *theEvent*.

```
public boolean trackMouseToAddMarker(NSEvent theEvent, boolean flag)
```

Discussion

NSRulerView invokes this method automatically to add a new marker or to move or remove an existing marker. You should never need to invoke it directly.

If *flag* is true, the receiver is a new marker being added to its NSRulerView. Before the receiver actually adds itself to the NSRulerView, it queries the NSRulerView's client view using rulerViewShouldAddMarker (page 1224). If the client view responds to this method and returns false, this method immediately returns false, and the new marker isn't added.

If *flag* is false, this method attempts to move or remove an existing marker, once again based on responses from the NSRulerView's client view. If the receiver is neither movable nor removable, this method immediately returns false. Further, if the NSRulerView's client responds to rulerViewShouldMoveMarker (page 1224) and returns false, this method returns false, indicating the receiver can't be moved.

If the receiver is being added or moved, this method queries the client view using rulerViewWillAddMarker (page 1225) or rulerViewWillMoveMarker (page 1225), respectively. If the client responds to the method, the return value is used as the receiver's location. These methods are invoked repeatedly as the receiver is dragged within the NSRulerView.

If the receiver is an existing marker being removed (dragged off the ruler), this method queries the client view using rulerViewShouldRemoveMarker (page 1225). If the client responds to this method and returns false, the marker is pinned to the NSRulerView's baseline (following the cursor on the baseline if it's movable).

When the user releases the mouse button, this method informs the client view of the marker's new status using rulerViewDidAddMarker (page 1223), rulerViewDidMoveMarker (page 1223), or rulerViewDidRemoveMarker (page 1223) as appropriate. The client view can use this notification to set the marker's represented object, modify its state and redisplay (for example, adjusting text layout around a new tab stop), or take whatever other action it might need. If *flag* is true and the user dragged the new marker away from the ruler, the marker isn't added, no message is sent, and this method returns false.

See the NSRulerView (page 1209) class description for more information on these client methods.

See Also

isMovable (page 1204)

isRemovable (page 1205)

NSRulerView

Inherits from	NSView : NSResponder : NSObject
Implements	NSCoding (NSResponder)
Package:	com.apple.cocoa.application
Companion guide	Rulers and Paragraph Styles

Class at a Glance

An NSRulerView displays a ruler and markers above or to the side of an NSScrollView's document view. Views within the NSScrollView can become clients of the ruler view, having it display markers for their elements, and receiving messages from the ruler view when the user manipulates the markers.

Principal Attributes

- Displays markers that represent elements of the client view.
- Displays in arbitrary units.
- Provides for an accessory view containing extra controls.

Constructor
`setHasHorizontalRuler` (page 1279) (NSScrollView)
`setHasVerticalRuler` (page 1280) (NSScrollView)

Commonly Used Methods

`setClientView` (page 1219)
Changes the ruler's client view.

`setMarkers` (page 1220)
Sets the markers displayed by the ruler view.

`setAccessoryView` (page 1219)
Sets the accessory view.

`trackMarker` (page 1222)
Allows the user to add a new marker.

Overview

An NSRulerView resides in an NSScrollView, displaying a labeled ruler and markers for its client, the document view of the NSScrollView, or a subview of the document view.

Tasks

Constructors

`NSRulerView` (page 1213)
Creates an NSRulerView with a zero-sized frame rectangle.

Altering Measurement Units

`registerUnit` (page 1214)
Registers a new unit of measurement with the NSRulerView class, making it available to all instances of NSRulerView.

`setMeasurementUnits` (page 1220)
Sets the measurement units used by the ruler to *`unitName`*.

`measurementUnits` (page 1217)
Returns the full name of the measurement units in effect for the receiver.

Setting the Client View

`setClientView` (page 1219)
Sets the receiver's client view to *`aView`*, without retaining it, and removes its ruler markers, after informing the prior client of the change using `rulerViewWillSetClientView` (page 1225).

`clientView` (page 1215)
Returns the receiver's client view, if it has one.

Setting an Accessory View

`setAccessoryView` (page 1219)
Sets the receiver's accessory view to *`aView`*.

`accessoryView` (page 1214)
Returns the receiver's accessory view, if it has one.

Setting the Zero Mark Position

`setOriginOffset` (page 1220)
Sets the distance to the zero hash mark from the bounds origin of the NSScrollView's document view (not of the receiver's client view), in the document view's coordinate system.

`originOffset` (page 1217)
Returns the distance from the receiver's zero hash mark to the bounds origin of the NSScrollView's document view (not the receiver's client view), in the document view's coordinate system.

Adding and Removing Markers

`setMarkers` (page 1220)
Sets the receiver's ruler markers to *markers*, removing any existing ruler markers and not consulting with the client view about the new markers.

`markers` (page 1216)
Returns the receiver's NSRulerMarkers.

`addMarker` (page 1215)
Adds *aMarker* to the receiver, without consulting the client view for approval.

`removeMarker` (page 1218)
Removes *aMarker* from the receiver, without consulting the client view for approval.

`trackMarker` (page 1222)
Tracks the mouse to add *aMarker* based on the initial mouse-down or mouse-dragged event *theEvent*.

Drawing Temporary Ruler Lines

`moveRulerline` (page 1217)
Draws temporary lines in the ruler area.

Drawing

`drawHashMarksAndLabelsInRect` (page 1215)
Draws the receiver's hash marks and labels in *aRect*, which is expressed in the receiver's coordinate system.

`drawMarkersInRect` (page 1216)
Draws the receiver's markers in *aRect*, which is expressed in the receiver's coordinate system.

`invalidateHashMarks` (page 1216)
Forces recalculation of the hash mark spacing for the next time the receiver is displayed.

Ruler Layout

`setScrollView` (page 1222)
Sets the NSScrollView that owns the receiver to *scrollView*, without retaining it.

`scrollView` (page 1219)
: Returns the NSScrollView object that contains the receiver.

`setOrientation` (page 1220)
: Sets the orientation of the receiver to *`orientation`*.

`orientation` (page 1217)
: Returns the orientation of the receiver.

`setReservedThicknessForAccessoryView` (page 1221)
: Sets the room available for the receiver's accessory view to *`thickness`*.

`reservedThicknessForAccessoryView` (page 1218)
: Returns the thickness reserved to contain the receiver's accessory view, its height or width depending on the receiver's orientation.

`setReservedThicknessForMarkers` (page 1221)
: Sets the room available for ruler markers to *`thickness`*.

`reservedThicknessForMarkers` (page 1218)
: Returns the thickness reserved to contain the images of the receiver's ruler markers, the height or width depending on the receiver's orientation.

`setRuleThickness` (page 1221)
: Sets to *`thickness`* the thickness of the area where ruler hash marks and labels are drawn.

`ruleThickness` (page 1219)
: Returns the thickness of the receiver's ruler area (the area where hash marks and labels are drawn), its height or width depending on the receiver's orientation.

`requiredThickness` (page 1218)
: Returns the thickness needed for proper tiling of the receiver within an NSScrollView.

`baselineLocation` (page 1215)
: Returns the location of the receiver's baseline, in its own coordinate system.

`isFlipped` (page 1216)
: Returns `true` if the NSRulerView's coordinate system is flipped, `false` otherwise.

Adding markers

`rulerViewShouldAddMarker` (page 1224) *delegate method*
: Requests permission for *`aRulerView`* to add *`aMarker`*, an NSRulerMarker being dragged onto the ruler by the user.

`rulerViewWillAddMarker` (page 1225) *delegate method*
: Informs the client that *`aRulerView`* will add the new NSRulerMarker, *`aMarker`*.

`rulerViewDidAddMarker` (page 1223) *delegate method*
: Informs the client that *`aRulerView`* allowed the user to add *`aMarker`*.

Moving markers

`rulerViewShouldMoveMarker` (page 1224) *delegate method*
: Requests permission for *`aRulerView`* to move *`aMarker`*.

`rulerViewWillMoveMarker` (page 1225) *delegate method*
: Informs the client that *`aRulerView`* will move *`aMarker`*, an NSRulerMarker already on the ruler view.

`rulerViewDidMoveMarker` (page 1223) *delegate method*
Informs the client that *aRulerView* allowed the user to move *aMarker*.

Removing markers

`rulerViewShouldRemoveMarker` (page 1225) *delegate method*
Requests permission for *aRulerView* to remove *aMarker*.

`rulerViewDidRemoveMarker` (page 1223) *delegate method*
Informs the client that *aRulerView* allowed the user to remove *aMarker*.

Handling mouse events

`rulerViewHandleMouseDown` (page 1224) *delegate method*
Informs the client that the user has pressed the mouse button while the cursor is in the ruler area of *aRulerView*.

Changing client view

`rulerViewWillSetClientView` (page 1225) *delegate method*
Informs the client view that *aRulerView* is about to be appropriated by *newClient*.

Constructors

NSRulerView

Creates an NSRulerView with a zero-sized frame rectangle.

```
public NSRulerView()
```

Discussion
The orientation is horizontal (`HorizontalRuler`).

Creates an NSRulerView with *aRect* as its frame rectangle.

```
public NSRulerView(NSRect aRect)
```

Discussion
The orientation is horizontal (`HorizontalRuler`).

Creates an NSRulerView with *orientation* (`HorizontalRuler` or `VerticalRuler`) within *aScrollView*.

```
public NSRulerView(NSScrollView aScrollView, int orientation)
```

Discussion
The new ruler view displays the user's preferred measurement units and has no client, markers, or accessory view. Unlike most subclasses of NSView, no initial frame rectangle is given for NSRulerView; its containing NSScrollView adjusts its frame rectangle as needed.

Static Methods

registerUnit

Registers a new unit of measurement with the NSRulerView class, making it available to all instances of NSRulerView.

```
public static void registerUnit(String unitName, String abbreviation, float
    conversionFactor, NSArray stepUpCycle, NSArray stepDownCycle)
```

Discussion

unitName is the name of the unit in English, in plural form and capitalized by convention—"Inches", for example. The unit name is used as a key to identify the measurement units and so shouldn't be localized. *abbreviation* is a localized short form of the unit name, such as "in" for Inches. *conversionFactor* is the number of PostScript points in the specified unit; there are 72.0 points per inch, for example. *stepUpCycle* and *stepDownCycle* are arrays of Numbers that specify how hash marks are calculated, as explained in "Setting Up a Ruler View". All numbers in *stepUpCycle* should be greater than 1.0, those in *stepDownCycle* should be less than 1.0.

NSRulerView supports these units by default:

Unit Name	Abbreviation	Points/Unit	Step-Up Cycle	Step-Down Cycle
Inches	in	72.0	2.0	0.5
Centimeters	cm	28.35	2.0	0.5, 0.2
Points	pt	1.0	10.0	0.5
Picas	pc	12.0	10.0	0.5

See Also

`setMeasurementUnits` (page 1220)

Instance Methods

accessoryView

Returns the receiver's accessory view, if it has one.

```
public NSView accessoryView()
```

See Also

`setAccessoryView` (page 1219)
`reservedThicknessForAccessoryView` (page 1218)

addMarker

Adds *aMarker* to the receiver, without consulting the client view for approval.

```
public void addMarker(NSRulerMarker aMarker)
```

Discussion

Throws an `InternalInconsistencyException` if the receiver has no client view.

See Also

`setMarkers` (page 1220)

`removeMarker` (page 1218)

`markers` (page 1216)

`trackMarker` (page 1222)

baselineLocation

Returns the location of the receiver's baseline, in its own coordinate system.

```
public float baselineLocation()
```

Discussion

This is a y position for horizontal rulers and an x position for vertical ones.

See Also

`ruleThickness` (page 1219)

clientView

Returns the receiver's client view, if it has one.

```
public NSView clientView()
```

See Also

`setClientView` (page 1219)

drawHashMarksAndLabelsInRect

Draws the receiver's hash marks and labels in *aRect*, which is expressed in the receiver's coordinate system.

```
public void drawHashMarksAndLabelsInRect(NSRect aRect)
```

Discussion

This method is invoked by `drawRect` (page 1203)—you should never need to invoke it directly. You can define custom measurement units using the class method `registerUnit` (page 1214). Override this method if you want to customize the appearance of the hash marks themselves.

See Also

`invalidateHashMarks` (page 1216)

`drawMarkersInRect` (page 1216)

drawMarkersInRect

Draws the receiver's markers in *aRect*, which is expressed in the receiver's coordinate system.

```
public void drawMarkersInRect(NSRect aRect)
```

Discussion

This method is invoked by drawRect (page 1203); you should never need to invoke it directly, but you might want to override it if you want to do something different when drawing markers.

See Also

reservedThicknessForMarkers (page 1218)

drawHashMarksAndLabelsInRect (page 1215)

invalidateHashMarks

Forces recalculation of the hash mark spacing for the next time the receiver is displayed.

```
public void invalidateHashMarks()
```

Discussion

You should never need to invoke this method directly, but might need to override it if you override drawHashMarksAndLabelsInRect (page 1215).

See Also

drawHashMarksAndLabelsInRect (page 1215)

isFlipped

Returns `true` if the NSRulerView's coordinate system is flipped, `false` otherwise.

```
public boolean isFlipped()
```

Discussion

A vertical ruler takes into account whether the coordinate system of the NSScrollView's document view—not the receiver's client view—is flipped. A horizontal ruler is always flipped.

markers

Returns the receiver's NSRulerMarkers.

```
public NSArray markers()
```

Discussion

The markers aren't guaranteed to be sorted in any particular order.

See Also

setMarkers (page 1220)

addMarker (page 1215)

removeMarker (page 1218)

markerLocation (page 1205) (NSRulerMarker)

measurementUnits

Returns the full name of the measurement units in effect for the receiver.

```
public String measurementUnits()
```

See Also

setMeasurementUnits (page 1220)

registerUnit (page 1214)

moveRulerline

Draws temporary lines in the ruler area.

```
public void moveRulerline(float oldLoc, float newLoc)
```

Discussion

If *oldLoc* is 0 or greater, erases the ruler line at that location; if *newLoc* is 0 or greater, draws a new rulerline at that location. *oldLoc* and *newLoc* are expressed in the coordinate system of the NSRulerView, not the client or document view, and are x coordinates for horizontal rulers and y coordinates for vertical rulers. Use NSView's `convert...` methods to convert coordinates from the client or document view's coordinate system to that of the NSRulerView.

This method is useful for drawing highlight lines in the ruler to show the position or extent of an object while it's being dragged in the client view. The sender is responsible for keeping track of the number and positions of temporary lines—the NSRulerView only does the drawing.

orientation

Returns the orientation of the receiver.

```
public int orientation()
```

Discussion

Possible values are described in "Constants" (page 1222).

See Also

setOrientation (page 1220)

originOffset

Returns the distance from the receiver's zero hash mark to the bounds origin of the NSScrollView's document view (not the receiver's client view), in the document view's coordinate system.

```
public float originOffset()
```

See Also

setOriginOffset (page 1220)

removeMarker

Removes *aMarker* from the receiver, without consulting the client view for approval.

```
public void removeMarker(NSRulerMarker aMarker)
```

See Also

setMarkers (page 1220)

addMarker (page 1215)

requiredThickness

Returns the thickness needed for proper tiling of the receiver within an NSScrollView.

```
public float requiredThickness()
```

Discussion

This thickness is the height of a horizontal ruler and the width of a vertical ruler. The required thickness is the sum of the thicknesses of the ruler area, the marker area, and the accessory view.

See Also

ruleThickness (page 1219)

reservedThicknessForMarkers (page 1218)

reservedThicknessForAccessoryView (page 1218)

reservedThicknessForAccessoryView

Returns the thickness reserved to contain the receiver's accessory view, its height or width depending on the receiver's orientation.

```
public float reservedThicknessForAccessoryView()
```

Discussion

This thickness is automatically enlarged as necessary to the accessory view's thickness (but never automatically reduced). To prevent retiling of a ruler view's scroll view, you should set its maximal thickness upon creating using setReservedThicknessForAccessoryView (page 1221).

reservedThicknessForMarkers

Returns the thickness reserved to contain the images of the receiver's ruler markers, the height or width depending on the receiver's orientation.

```
public float reservedThicknessForMarkers()
```

Discussion

This thickness is automatically enlarged as necessary to accommodate the thickest ruler marker image (but never automatically reduced). To prevent retiling of a ruler view's scroll view, you should set its maximal thickness upon creating using setReservedThicknessForMarkers (page 1221).

See Also

thicknessRequiredInRuler (page 1207) (NSRulerMarker)

ruleThickness

Returns the thickness of the receiver's ruler area (the area where hash marks and labels are drawn), its height or width depending on the receiver's orientation.

```
public float ruleThickness()
```

See Also

`setRuleThickness` (page 1221)

scrollView

Returns the NSScrollView object that contains the receiver.

```
public NSScrollView scrollView()
```

See Also

`setScrollView` (page 1222)
`setHorizontalRulerView` (page 1281) (NSScrollView)
`setVerticalRulerView` (page 1284) (NSScrollView)

setAccessoryView

Sets the receiver's accessory view to *aView*.

```
public void setAccessoryView(NSView aView)
```

Discussion

Throws an `InternalInconsistencyException` if *aView* is not `null` and the receiver has no client view.

See Also

`accessoryView` (page 1214)
`reservedThicknessForAccessoryView` (page 1218)

setClientView

Sets the receiver's client view to *aView*, without retaining it, and removes its ruler markers, after informing the prior client of the change using `rulerViewWillSetClientView` (page 1225).

```
public void setClientView(NSView aView)
```

Discussion

aView must be either the document view of the NSScrollView that contains the receiver or a subview of the document view.

See Also

`clientView` (page 1215)

setMarkers

Sets the receiver's ruler markers to *markers*, removing any existing ruler markers and not consulting with the client view about the new markers.

```
public void setMarkers(NSArray markers)
```

Discussion

markers can be `null` or empty to remove all ruler markers. Throws an `InternalInconsistencyException` if *markers* is not `null` and the receiver has no client view.

See Also

`addMarker` (page 1215)

`removeMarker` (page 1218)

setMeasurementUnits

Sets the measurement units used by the ruler to *unitName*.

```
public void setMeasurementUnits(String unitName)
```

Discussion

unitName must have been registered with the NSRulerView class object prior to invoking this method. See the description of the class method `registerUnit` (page 1214) for a list of predefined units.

See Also

`measurementUnits` (page 1217)

setOrientation

Sets the orientation of the receiver to *orientation*.

```
public void setOrientation(int orientation)
```

Discussion

Possible values for *orientation* are described in "Constants" (page 1222). You should never need to invoke this method directly—it's automatically invoked by the containing NSScrollView.

See Also

`orientation` (page 1217)

setOriginOffset

Sets the distance to the zero hash mark from the bounds origin of the NSScrollView's document view (not of the receiver's client view), in the document view's coordinate system.

```
public void setOriginOffset(float offset)
```

Discussion

The default offset is 0.0, meaning that the ruler origin coincides with the bounds origin of the document view.

See Also
`originOffset` (page 1217)

setReservedThicknessForAccessoryView

Sets the room available for the receiver's accessory view to *thickness*.

```
public void setReservedThicknessForAccessoryView(float thickness)
```

Discussion
If the ruler is horizontal, *thickness* is the height of the accessory view; otherwise, it's the width. NSRulerViews by default reserve no space for an accessory view.

An NSRulerView automatically increases the reserved thickness as necessary to that of the accessory view. When the accessory view is thinner than the reserved space, it's centered in that space. If you plan to use several accessory views of different sizes, you should set the reserved thickness beforehand to that of the thickest accessory view, in order to avoid retiling of the NSScrollView.

See Also
`reservedThicknessForAccessoryView` (page 1218)
`setAccessoryView` (page 1219)
`setReservedThicknessForMarkers` (page 1221)

setReservedThicknessForMarkers

Sets the room available for ruler markers to *thickness*.

```
public void setReservedThicknessForMarkers(float thickness)
```

Discussion
The default thickness reserved for markers is 15.0 PostScript units for a horizontal ruler and 0.0 PostScript units for a vertical ruler (under the assumption that vertical rulers rarely contain markers). If you don't expect to have any markers on the ruler, you can set the reserved thickness to 0.0.

An NSRulerView automatically increases the reserved thickness as necessary to that of its thickest marker. If you plan to use markers of varying sizes, you should set the reserved thickness beforehand to that of the thickest one in order to avoid retiling of the NSScrollView.

See Also
`reservedThicknessForMarkers` (page 1218)
`setMarkers` (page 1220)
`setReservedThicknessForAccessoryView` (page 1221)
`thicknessRequiredInRuler` (page 1207) (NSRulerMarker)

setRuleThickness

Sets to *thickness* the thickness of the area where ruler hash marks and labels are drawn.

```
public void setRuleThickness(float thickness)
```

Discussion
This value is the height of the ruler area for a horizontal ruler or the width of the ruler area for a vertical ruler. Rulers are by default 16.0 PostScript units thick. You should rarely need to change this layout attribute, but subclasses might do so to accommodate custom drawing.

See Also
`ruleThickness` (page 1219)

setScrollView

Sets the NSScrollView that owns the receiver to *scrollView*, without retaining it.

```
public void setScrollView(NSScrollView scrollView)
```

Discussion
This method is generally invoked only by the ruler's scroll view; you should rarely need to invoke it directly.

See Also
`scrollView` (page 1219)
`setHorizontalRulerView` (page 1281) (NSScrollView)
`setVerticalRulerView` (page 1284) (NSScrollView)

trackMarker

Tracks the mouse to add *aMarker* based on the initial mouse-down or mouse-dragged event *theEvent*.

```
public boolean trackMarker(NSRulerMarker aMarker, NSEvent theEvent)
```

Discussion
Returns `true` if the receiver adds *aMarker*, `false` if it doesn't. This method works by sending `trackMouseToAddMarker` (page 1207) to *aMarker* with *theEvent* and `true` as arguments.

An application typically invokes this method in one of two cases. In the simpler case, the client view can implement `rulerViewHandleMouseDown` (page 1224) to invoke this method when the user presses the mouse button while the cursor is in the NSRulerView's ruler area. This technique is appropriate when it's clear what kind of marker will be added by clicking the ruler area. The second, more general, case involves the application providing a palette of different kinds of markers that can be dragged onto the ruler, from the ruler's accessory view or from some other place. With this technique the palette tracks the cursor until it enters the ruler view, at which time it hands over control to the ruler view by invoking `trackMarker` (page 1222).

See Also
`addMarker` (page 1215)
`setMarkers` (page 1220)

Constants

The following constants are defined to specify a ruler's orientation and are used by `orientation` (page 1217) and `setOrientation` (page 1220):

Constant	Description
`HorizontalRuler`	Ruler is oriented horizontally.
`VerticalRuler`	Ruler is oriented vertically.

Delegate Methods

rulerViewDidAddMarker

Informs the client that *aRulerView* allowed the user to add *aMarker*.

```
public abstract void rulerViewDidAddMarker(NSRulerView aRulerView, NSRulerMarker
    aMarker)
```

Discussion
The client can take whatever action it needs based on this message, such as adding a new tab stop to the selected paragraph or creating a layout guideline.

See Also
`representedObject` (page 1205) (NSRulerMarker)
`markerLocation` (page 1205) (NSRulerMarker)

rulerViewDidMoveMarker

Informs the client that *aRulerView* allowed the user to move *aMarker*.

```
public abstract void rulerViewDidMoveMarker(NSRulerView aRulerView, NSRulerMarker
     aMarker)
```

Discussion
The client can take whatever action it needs based on this message, such as updating the location of a tab stop in the selected paragraph, moving a layout guideline, or resizing a graphics element.

See Also
`representedObject` (page 1205) (NSRulerMarker)
`markerLocation` (page 1205) (NSRulerMarker)

rulerViewDidRemoveMarker

Informs the client that *aRulerView* allowed the user to remove *aMarker*.

```
public abstract void rulerViewDidRemoveMarker(NSRulerView aRulerView, NSRulerMarker
     aMarker)
```

Discussion
The client can take whatever action it needs based on this message, such as deleting a tab stop from the paragraph style or removing a layout guideline.

See Also

`representedObject` (page 1205) (NSRulerMarker)

rulerViewHandleMouseDown

Informs the client that the user has pressed the mouse button while the cursor is in the ruler area of *aRulerView*.

```
public abstract void rulerViewHandleMouseDown(NSRulerView aRulerView, NSEvent
    theEvent)
```

Discussion

theEvent is the mouse-down event that triggered the message. The client view can implement this method to perform an action such as adding a new marker using `trackMarker` (page 1222) or adding layout guidelines.

rulerViewShouldAddMarker

Requests permission for *aRulerView* to add *aMarker*, an NSRulerMarker being dragged onto the ruler by the user.

```
public abstract boolean rulerViewShouldAddMarker(NSRulerView aRulerView,
    NSRulerMarker aMarker)
```

Discussion

If the client returns `true` the ruler view accepts the new marker and begins tracking its movement; if the client returns `false` the ruler view refuses the new marker.

See Also

`rulerViewWillAddMarker` (page 1225)

rulerViewShouldMoveMarker

Requests permission for *aRulerView* to move *aMarker*.

```
public abstract boolean rulerViewShouldMoveMarker(NSRulerView aRulerView,
    NSRulerMarker aMarker)
```

Discussion

If the client returns `true` the ruler view allows the user to move the marker; if the client returns `false` the marker doesn't move.

The user's ability to move a marker is typically set on the marker itself, using NSRulerMarker's `setMovable` (page 1206) method. You should use this client view method only when the marker's movability can vary depending on a variable condition (for example, if graphic items can be locked down to prevent them from being inadvertently moved).

See Also

`rulerViewWillMoveMarker` (page 1225)

rulerViewShouldRemoveMarker

Requests permission for `aRulerView` to remove `aMarker`.

```
public abstract boolean rulerViewShouldRemoveMarker(NSRulerView aRulerView,
    NSRulerMarker aMarker)
```

Discussion

If the client returns `true` the ruler view allows the user to remove the marker; if the client returns `false` the marker is kept pinned to the ruler's baseline. This message is sent as many times as needed while the user drags the marker.

The user's ability to remove a marker is typically set on the marker itself, using NSRulerMarker's `setRemovable` (page 1207) method. You should use this client view method only when the marker's removability can vary while the user drags it (for example, if the user must press the Shift key to remove a marker).

rulerViewWillAddMarker

Informs the client that `aRulerView` will add the new NSRulerMarker, `aMarker`.

```
public abstract float rulerViewWillAddMarker(NSRulerView aRulerView, NSRulerMarker
     aMarker, float location)
```

Discussion

`location` is the marker's tentative new location, expressed in the client view's coordinate system. The value returned by the client view is actually used; the client can simply return `location` unchanged or adjust it as needed. For example, it may snap the location to a grid. This message is sent repeatedly to the client as the user drags the marker.

See Also

`rulerViewWillMoveMarker` (page 1225)

rulerViewWillMoveMarker

Informs the client that `aRulerView` will move `aMarker`, an NSRulerMarker already on the ruler view.

```
public abstract float rulerViewWillMoveMarker(NSRulerView aRulerView, NSRulerMarker
     aMarker, float location)
```

Discussion

`location` is the marker's tentative new location, expressed in the client view's coordinate system. The value returned by the client view is actually used; the client can simply return `location` unchanged or adjust it as needed. For example, it may snap the location to a grid. This message is sent repeatedly to the client as the user drags the marker.

See Also

`rulerViewWillAddMarker` (page 1225)

rulerViewWillSetClientView

Informs the client view that `aRulerView` is about to be appropriated by `newClient`.

```
public abstract void rulerViewWillSetClientView(NSRulerView aRulerView, NSView
    newClient)
```

Discussion

The client view can use this opportunity to clear any cached information related to the ruler.

NSSavePanel

Inherits from	NSPanel : NSWindow : NSResponder : NSObject
Implements	NSCoding (NSResponder)
Package:	com.apple.cocoa.application
Companion guides	Application File Management Sheet Programming Topics for Cocoa

Class at a Glance

An NSSavePanel object manages a panel that allows users to specify the directory and name under which a file is saved. It supports browsing of the file system, and it accommodates custom accessory views. An NSSavePanel is a recycled object: when you request a Save panel, NSSavePanel tries to reuse an existing Save panel rather than create a new one.

Principal Attributes

- Filename

`savePanel` (page 1232)
Returns a Save panel instance.

Commonly Used Methods

`runModal` (page 1237)
Displays the panel and begins the event loop.

`filename` (page 1235)
Returns the selected or entered filename.

`directory` (page 1235)
Returns the full path of the selected file.

`ok` (page 1236)
Invoked when users click OK.

Overview

NSSavePanel creates and manages a Save panel and allows you to run the panel in a modal loop. 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 an extension.

Tasks

Constructors

`NSSavePanel` (page 1231)
Creates an empty NSSavePanel.

Obtaining

`savePanel` (page 1232)
Creates and returns a new NSSavePanel object.

Customizing the NSSavePanel

`setAccessoryView` (page 1238)
Customizes the panel for the application by adding a custom NSView (*aView*) to the panel.

`accessoryView` (page 1232)
Returns the custom accessory view for the current application.

`setTitle` (page 1242)
Sets the title of the receiver to *title*.

`title` (page 1242)
Returns the title of the receiver.

`setPrompt` (page 1241)
Sets the prompt of the default button.

`prompt` (page 1237)
Returns the prompt of the default button.

`setNameFieldLabel` (page 1241)
Sets the text displayed in front of the text field to *label*.

`nameFieldLabel` (page 1236)
Returns the string displayed in front of the filename text field.

`setMessage` (page 1241)
Sets the message text displayed in the panel.

`message` (page 1236)
Returns the message displayed in the save panel.

Working with Extension Hiding

`setCanSelectHiddenExtension` (page 1240)
Sets whether the receiver allows the user to hide or show extensions to *flag*.

`canSelectHiddenExtension` (page 1235)
Returns whether the receiver allows the user to hide or show extensions.

`setExtensionHidden` (page 1240)
Sets the value of the extension-hiding checkbox to *flag*.

`isExtensionHidden` (page 1236)
Returns `true` if the extension-hiding checkbox is visible and checked.

Setting Directory and File Type

`setDirectory` (page 1240)
Sets the current pathname in the receiver's browser.

`setRequiredFileType` (page 1242)
Specifies the type, an extension to be appended to any selected files that don't already have that extension; "nib" and "rtf" are examples.

`requiredFileType` (page 1237)
Returns the required file type (if any).

`setAllowedFileTypes` (page 1239)
Specifies the allowed file types for the receiver.

`allowedFileTypes` (page 1233)
Returns an array of the allowed file types.

`setAllowsOtherFileTypes` (page 1239)
Sets whether the receiver allows the user to save files with an extension that's not in the list of allowed types.

`allowsOtherFileTypes` (page 1233)
Returns whether the receiver allows the user to save files with an extension that's not in the list of allowed types.

`treatsFilePackagesAsDirectories` (page 1243)
Returns whether the receiver displays file packages to the user as directories.

`setTreatsFilePackagesAsDirectories` (page 1242)
Sets the receiver's behavior for displaying file packages (for example, `MyApp.app`) to the user.

`validateVisibleColumns` (page 1243)
Validates and possibly reloads the browser columns visible in the receiver by invoking the delegate method `panelShouldShowFilename` (page 1244).

`setCanCreateDirectories` (page 1239)
Sets whether the receiver allows the user to create directories.

`canCreateDirectories` (page 1234)
Returns whether the receiver allows the user to create directories.

Running the NSSavePanel

beginSheetForDirectory (page 1234)
Presents a Save panel as a sheet with the directory specified by *path* and optionally, the file specified by *name* selected.

runModal (page 1237)
Displays the receiver and begins its event loop with the current working (or last selected) directory as the default starting point.

runModalInDirectory (page 1238)
Initializes the receiver to the directory specified by *path* and, optionally, the file specified by *filename*, then displays it and begins its modal event loop; *path* and *filename* can be empty strings.

Getting User Selections

directory (page 1235)
Returns the absolute pathname of the directory currently shown in the receiver.

filename (page 1235)
Returns the absolute pathname of the file currently shown in the receiver.

URL (page 1243)
Returns the absolute pathname of the file currently shown in the receiver as a URL.

isExpanded (page 1235)
Returns whether the receiver is expanded.

Action Methods

cancel (page 1234)
This action method is invoked when the user clicks the panel's Cancel button.

ok (page 1236)
This action method is invoked when the user clicks the panel's OK button.

Responding to User Input

selectText (page 1238)
This method has been deprecated.

Setting the Delegate

setDelegate (page 1240)
Makes *anObject* the receiver's delegate, after verifying which delegate methods are implemented.

delegate (page 1235)
Returns the receiver's delegate.

Working with filenames

`panelCompareFilenames` (page 1243) *delegate method*
Controls the ordering of files presented by the NSSavePanel *sender*.

`panelIsValidFilename` (page 1244) *delegate method*

`panelShouldShowFilename` (page 1244) *delegate method*
Gives the delegate the opportunity to filter out items that it doesn't want the user to see or choose.

`panelUserEnteredFilename` (page 1245) *delegate method*
Sent when the user confirms a filename choice by hitting OK or Return in the NSSavePanel *sender*.

Expanding the panel

`panelWillExpand` (page 1245) *delegate method*
Sent when the NSSavePanel *sender* is about to expand or collapse because the user clicked the disclosure triangle that displays or hides the file browser.

Managing panel changes

`panelDirectoryDidChange` (page 1244) *delegate method*
Sent when the user has changed the selected directory in the NSSavePanel *sender*.

`panelSelectionDidChange` (page 1244) *delegate method*
Sent when the user has changed the selection in the NSSavePanel *sender*.

Constructors

NSSavePanel

Creates an empty NSSavePanel.

```
public NSSavePanel()
```

Creates a new NSSavePanel.

```
public NSSavePanel(NSRect contentRect, int styleMask, int backingType, boolean
    flag)
```

Discussion
The *contentRect* argument specifies the location and size of the panel's content area in screen coordinates. Note that the window server limits window position coordinates to ±16,000 and sizes to 10,000.

The *styleMask* argument specifies the panel's style. Either it can be `NSWindow.BorderlessWindowMask`, or it can contain any of the options described in NSWindow's "Constants" (page 1875), combined using the C bitwise OR operator.

Borderless windows display none of the usual peripheral elements and are generally useful only for display or caching purposes; you should normally not need to create them. Also, note that an NSWindow's style mask should include `NSWindow.TitledWindowMask` if it includes any of the others.

The *backingType* argument specifies how the drawing done in the panel is buffered by the object's window device, and possible values are described in NSWindow's "Constants" (page 1875).

The *flag* argument determines whether the window server creates a window device for the new panel immediately. If *flag* is `true`, it defers creating the window until the panel is moved onscreen. All display messages sent are postponed until the panel is created, just before it's moved onscreen. Deferring the creation of the window improves launch time and minimizes the virtual memory load on the window server.

The new panel creates an instance of NSView to be its default content view. You can replace it with your own object by using the `setContentView` (page 1858) method.

Creates a new NSSavePanel.

```
public NSSavePanel(NSRect contentRect, int styleMask, int backingType, boolean
    flag, NSScreen aScreen)
```

Discussion
This constructor is equivalent to the preceding one except *contentRect* is specified relative to the lower-left corner of *aScreen*.

If *aScreen* is `null`, *contentRect* 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.

Static Methods

savePanel

Creates and returns a new NSSavePanel object.

```
public static NSSavePanel savePanel()
```

Discussion
The save panel has been initialized with default values.

Instance Methods

accessoryView

Returns the custom accessory view for the current application.

```
public NSView accessoryView()
```

Discussion

This view is set by setAccessoryView (page 1238).

In order to free up unused memory after closing the receiver, the accessory view is released after the panel is closed. If you rely on the receiver to hold onto the accessory view until the next time you use it, the accessory view may be deallocated unexpectedly. If you retain the accessory view in your own code, though, this deallocation should not be a problem.

See Also

setAccessoryView (page 1238)

allowedFileTypes

Returns an array of the allowed file types.

```
public NSArray allowedFileTypes()
```

Discussion

If the user specifies a file whose type is in the array of allowed types, they will not be presented with another dialog (see allowsOtherFileTypes (page 1233) for details about this dialog) when trying to save. Examples of common file types are “rtf”, “tiff”, and “ps”. File type strings encoding HFS file types are not valid values for this attribute. A null return value, which is the default, indicates that the user can save to any ASCII file.

Availability

Available in Mac OS X v10.3 and later.

See Also

setAllowedFileTypes (page 1239)
requiredFileType (page 1237)

allowsOtherFileTypes

Returns whether the receiver allows the user to save files with an extension that’s not in the list of allowed types.

```
public boolean allowsOtherFileTypes()
```

Discussion

If the user tries to save a filename with a recognized extension that's not in the list of allowed types they are presented with a dialog. If this method returns true, then the dialog presents the option of using the extension the user specified.

The default setting is false.

Availability

Available in Mac OS X v10.3 and later.

See Also

setAllowsOtherFileTypes (page 1239)
allowedFileTypes (page 1233)

beginSheetForDirectory

Presents a Save panel as a sheet with the directory specified by *path* and optionally, the file specified by *name* selected.

```
public void beginSheetForDirectory(String path, String name, NSWindow docWindow,
    Object modalDelegate, NSSelector didEndSelector, Object contextInfo)
```

Discussion
If *docWindow* is not `null`, the Save panel slides down as a sheet running as a document modal window. If *docWindow* is `null`, the behavior defaults to a standalone modal window.

The *didEndSelector* method is optional. If implemented by the *modalDelegate*, this method is invoked, passing *contextInfo* as an argument, after the modal session has ended, but before dismissing the Save panel. *didEndSelector* may dismiss the Save panel itself; otherwise it will be dismissed on return from the method. *modalDelegate* is not the same as a delegate assigned to the panel. Modal delegates in sheets are temporary and the relationship only lasts until the sheet is dismissed. The NSSavePanel object does not retain the modal delegate.

didEndSelector should have the following signature:

```
public void savePanelDidEndReturnCode (NSSavePanel sheet, int returnCode,  Object
  contextInfo)
```

The value passed as *returnCode* will be either `NSPanel.CancelButton` or `NSPanel.OKButton`.

Save as above, without the optional *contextInfo*.

```
public void beginSheetForDirectory(String path, String name, NSWindow docWindow,
    Object modalDelegate, NSSelector didEndSelector)
```

cancel

This action method is invoked when the user clicks the panel's Cancel button.

```
public void cancel(Object sender)
```

See Also
`ok` (page 1236)

canCreateDirectories

Returns whether the receiver allows the user to create directories.

```
public boolean canCreateDirectories()
```

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

See Also
`setCanCreateDirectories` (page 1239)

canSelectHiddenExtension

Returns whether the receiver allows the user to hide or show extensions.

```
public boolean canSelectHiddenExtension()
```

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

See Also
`setCanSelectHiddenExtension` (page 1240)

delegate

Returns the receiver's delegate.

```
public Object delegate()
```

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

See Also
`setDelegate` (page 1240)

directory

Returns the absolute pathname of the directory currently shown in the receiver.

```
public String directory()
```

See Also
`setDirectory` (page 1240)

filename

Returns the absolute pathname of the file currently shown in the receiver.

```
public String filename()
```

See Also
`URL` (page 1243)

isExpanded

Returns whether the receiver is expanded.

```
public boolean isExpanded()
```

isExtensionHidden

Returns true if the extension-hiding checkbox is visible and checked.

```
public boolean isExtensionHidden()
```

Discussion

Returns false otherwise.

See Also

setCanSelectHiddenExtension (page 1240)

setExtensionHidden (page 1240)

message

Returns the message displayed in the save panel.

```
public String message()
```

Discussion

This message appears on all NSSavePanels (or all NSOpenPanels if the receiver of this message is an NSOpenPanel) in your application. The default message text is an empty string.

Availability

Available in Mac OS X v10.3 and later.

See Also

setMessage (page 1241)

nameFieldLabel

Returns the string displayed in front of the filename text field.

```
public String nameFieldLabel()
```

Discussion

By default the label is “Save As:”.

Availability

Available in Mac OS X v10.3 and later.

See Also

setNameFieldLabel (page 1241)

ok

This action method is invoked when the user clicks the panel’s OK button.

```
public void ok(Object sender)
```

See Also

cancel (page 1234)

prompt

Returns the prompt of the default button.

```
public String prompt()
```

Discussion

This prompt appears on all NSSavePanels (or all NSOpenPanels if the receiver of this message is an NSOpenPanel) in your application. By default the text in the default button is “Open” for an open panel and “Save” for a Save panel.

See Also

`setPrompt` (page 1241)

requiredFileType

Returns the required file type (if any).

```
public String requiredFileType()
```

Discussion

A file specified in the Save panel is saved with the designated filename and this file type as an extension. Examples of common file types are “rtf”, “tiff”, and “ps”. File type strings encoding HFS file types are not valid values for this attribute. An `null` return value indicates that the user can save to any ASCII file.

This method is equivalent to calling `allowedFileTypes` (page 1233) and returning the first element of the list of allowed types, or `null` if there are none.

See Also

`setRequiredFileType` (page 1242)

runModal

Displays the receiver and begins its event loop with the current working (or last selected) directory as the default starting point.

```
public int runModal()
```

Discussion

Invokes `runModalInDirectory` (page 1238) (*`file`* argument is an empty String), which in turn performs NSApplication’s `runModalForWindow` (page 119) method with `this` as the argument. Returns `NSPanel.OKButton` (if the user clicks the OK button) or `NSPanel.CancelButton` (if the user clicks the Cancel button). Do not invoke `filename` (page 1235) or `directory` (page 1235) within a modal loop because the information these methods fetch is updated only upon return.

See Also

`runModalInDirectory` (page 1238)

`runModalForWindow` (page 119) (NSApplication)

runModalInDirectory

Initializes the receiver to the directory specified by *path* and, optionally, the file specified by *filename*, then displays it and begins its modal event loop; *path* and *filename* can be empty strings.

```
public int runModalInDirectory(String path, String filename)
```

Discussion

If *path* is null, the previous directory the Save panel was in is used. This method invokes NSApplication's runModalForWindow (page 119) method with this as the argument. Returns NSPanel.OKButton (if the user clicks the OK button) or NSPanel.CancelButton (if the user clicks the Cancel button). Do not invoke filename (page 1235) or directory (page 1235) within a modal loop because the information these methods fetch is updated only upon return.

If *window* is not null, the Save panel slides down as a sheet running as a document modal window. If *window* is null, the behavior defaults to a standalone modal panel.

```
public int runModalInDirectory(String path, String filename, NSWindow window)
```

Availability

Available in Mac OS X v10.3 and later.

See Also

runModal (page 1237)

runModalForWindow (page 119) (NSApplication)

selectText

This method has been deprecated.

```
public void selectText(Object sender)
```

Availability

Deprecated in Mac OS X v10.3.

setAccessoryView

Customizes the panel for the application by adding a custom NSView (*aView*) to the panel.

```
public void setAccessoryView(NSView aView)
```

Discussion

The custom NSView that's added appears just above the OK and Cancel buttons at the bottom of the panel. The NSSavePanel automatically resizes itself to accommodate *aView*. You can invoke this method repeatedly to change the accessory view as needed. If *aView* is null, the NSSavePanel removes the current accessory view.

See Also

accessoryView (page 1232)

setAllowedFileTypes

Specifies the allowed file types for the receiver.

```
public void setAllowedFileTypes(NSArray types)
```

Discussion

`types` may not be empty. A file type is an extension to be appended to any selected files that don’t already have that extension; “nib” and “rtf” are examples. The items in `types` should not include the period that begins the extension. File type strings encoding HFS file types are not valid values. Pass `null`, to allow any file type, which is the default.

Availability

Available in Mac OS X v10.3 and later.

See Also

`allowedFileTypes` (page 1233)

`setRequiredFileType` (page 1242)

setAllowsOtherFileTypes

Sets whether the receiver allows the user to save files with an extension that’s not in the list of allowed types.

```
public void setAllowsOtherFileTypes(boolean flag)
```

Discussion

If the user tries to save a filename with a recognized extension that's not in the list of allowed types they are presented with a dialog. If `allowsOtherFileTypes` (page 1233) is `true`, then the dialog presents the option of using the extension the user specified.

The default setting is `false`.

Availability

Available in Mac OS X v10.3 and later.

See Also

`allowsOtherFileTypes` (page 1233)

`allowedFileTypes` (page 1233)

setCanCreateDirectories

Sets whether the receiver allows the user to create directories.

```
public void setCanCreateDirectories(boolean)
```

Availability

Available in Mac OS X v10.3 and later.

See Also

`canCreateDirectories` (page 1234)

setCanSelectHiddenExtension

Sets whether the receiver allows the user to hide or show extensions to *flag*.

```
public void setCanSelectHiddenExtension(boolean flag)
```

Discussion
This method must be called before the panel is displayed.

See Also
canSelectHiddenExtension (page 1235)
isExtensionHidden (page 1236)
setExtensionHidden (page 1240)

setDelegate

Makes *anObject* the receiver's delegate, after verifying which delegate methods are implemented.

```
public void setDelegate(Object anObject)
```

Discussion
Use NSWindow's delegate (page 1826) method to retrieve the NSSavePanel's delegate.

setDirectory

Sets the current pathname in the receiver's browser.

```
public void setDirectory(String path)
```

Discussion
The *path* argument must be an absolute pathname.

See Also
directory (page 1235)

setExtensionHidden

Sets the value of the extension-hiding checkbox to *flag*.

```
public void setExtensionHidden(boolean flag)
```

Discussion
This method should rarely be used since the state is saved on a per application basis. Use this method to set whether a file's extension should be indicated as being shown.

See Also
setCanSelectHiddenExtension (page 1240)
isExtensionHidden (page 1236)

setMessage

Sets the message text displayed in the panel.

```
public void setMessage(String message)
```

Discussion
This message appears on all NSSavePanels (or all NSOpenPanels if the receiver of this message is an NSOpenPanel) in your application. The default message text is an empty string.

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

See Also
`message` (page 1236)

setNameFieldLabel

Sets the text displayed in front of the text field to *`label`*.

```
public void setNameFieldLabel(String label)
```

Discussion
By default the label is "Save As:".

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

See Also
`nameFieldLabel` (page 1236)

setPrompt

Sets the prompt of the default button.

```
public void setPrompt(String prompt)
```

Discussion
This prompt appears on all NSSavePanels (or all NSOpenPanels if the receiver of this message is an NSOpenPanel) in your application. By default the text in the default button is "Open" for an Open panel and "Save" for a Save panel.

It is intended that short words or phrases, such as "Open," "Save," "Set," or "Choose," be used on the button. The button is not resized to accommodate long prompts.

Since this method previously affected a title field, any colon at the end of *`prompt`* is removed.

See Also
`prompt` (page 1237)

setRequiredFileType

Specifies the type, an extension to be appended to any selected files that don't already have that extension; "nib" and "rtf" are examples.

```
public void setRequiredFileType(String type)
```

Discussion

The argument *type* should not include the period that begins the extension. Pass `null` to indicate any type. File type strings encoding HFS file types are not valid values for this attribute. You need to invoke this method each time the Save panel is used for another file type within the application.

This method is equivalent to calling `setAllowedFileTypes` (page 1239) with an array containing only *type* (unless *type* is null, and then it's equivalent to calling `setAllowedFileTypes` with `null`).

See Also

`requiredFileType` (page 1237)

setTitle

Sets the title of the receiver to *title*.

```
public void setTitle(String title)
```

Discussion

By default, "Save" is the title string. If you adapt the NSSavePanel for other uses, its title should reflect the user action that brings it to the screen.

See Also

`title` (page 1242)

setTreatsFilePackagesAsDirectories

Sets the receiver's behavior for displaying file packages (for example, `MyApp.app`) to the user.

```
public void setTreatsFilePackagesAsDirectories(boolean flag)
```

Discussion

If *flag* is `true`, the user is shown files and subdirectories within a file package. If `false`, the NSSavePanel shows each file package as a file, thereby giving no indication that it is a directory.

See Also

`treatsFilePackagesAsDirectories` (page 1243)

title

Returns the title of the receiver.

```
public String title()
```

See Also

`setTitle` (page 1242)

treatsFilePackagesAsDirectories

Returns whether the receiver displays file packages to the user as directories.

```
public boolean treatsFilePackagesAsDirectories()
```

See Also
`setTreatsFilePackagesAsDirectories` (page 1242)

URL

Returns the absolute pathname of the file currently shown in the receiver as a URL.

```
public java.net.URL URL()
```

See Also
`filename` (page 1235)

validateVisibleColumns

Validates and possibly reloads the browser columns visible in the receiver by invoking the delegate method `panelShouldShowFilename` (page 1244).

```
public void validateVisibleColumns()
```

Discussion
You might use this method if you want the browser to show only files with certain extensions based on the selection made in an accessory-view pop-up list. When the user changes the selection, you would invoke this method to revalidate the visible columns.

Delegate Methods

panelCompareFilenames

Controls the ordering of files presented by the NSSavePanel *sender*.

```
public abstract int panelCompareFilenames(Object sender, String fileName1, String
     fileName2, boolean flag)
```

Discussion
This method should return:

- `OrderedAscending` if *fileName1* should precede *fileName2*
- `OrderedSame` if the two names are equivalent
- `OrderedDescending` if *fileName2* should precede *fileName1*

The *flag* argument, if `true`, indicates that the ordering is to be case-sensitive.

Don't reorder filenames in the Save panel without good reason, because 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 Finder. The default behavior of Save and Open panels is to order files as they appear in the Finder. Note also that by implementing this method you will reduce the operating performance of the panel.

panelDirectoryDidChange

Sent when the user has changed the selected directory in the NSSavePanel *sender*.

```
public abstract void panelDirectoryDidChange(Object sender, String path)
```

Discussion
path contains the newly selected directory.

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

panelIsValidFilename

```
public abstract boolean panelIsValidFilename(Object sender, String filename)
```

Discussion
The NSSavePanel *sender* sends this message just before the end of a modal session for each filename displayed or selected (including filenames in multiple selections). The delegate determines whether it wants the file identified by *filename*; it returns `true` if the filename is valid, or `false` if the NSSavePanel should stay in its modal loop and wait for the user to type in or select a different filename or names. If the delegate refuses a filename in a multiple selection, none of the filenames in the selection is accepted.

panelSelectionDidChange

Sent when the user has changed the selection in the NSSavePanel *sender*.

```
public abstract void panelSelectionDidChange(Object sender)
```

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

panelShouldShowFilename

Gives the delegate the opportunity to filter out items that it doesn't want the user to see or choose.

```
public abstract boolean panelShouldShowFilename(Object sender, String filename)
```

Discussion
The NSSavePanel *sender* sends this message to the panel's delegate for each file or directory (filename) it is about to load in the browser. The delegate returns `true` if *filename* should be displayed, and `false` if the NSSavePanel should ignore the file or directory.

panelUserEnteredFilename

Sent when the user confirms a filename choice by hitting OK or Return in the NSSavePanel *sender*.

```
public abstract String panelUserEnteredFilename(Object sender, String filename,
    boolean okFlag)
```

Discussion

You can either leave the filename alone, return a new filename, or return `null` to cancel the save (and leave the Save panel as is). This method is sent before any required extension is appended to the filename and before the Save panel asks the user whether to replace an existing file.

Note that in the future, this method may be called multiple times in the sessions as the user types. In those cases, *okFlag* will be `false` until the user confirms the choice, in which case *okFlag* will become `true`. If the delegate does extensive validation or puts up alerts, it should do so only when *okFlag* is `true`.

See Also

`panelIsValidFilename` (page 1244)

panelWillExpand

Sent when the NSSavePanel *sender* is about to expand or collapse because the user clicked the disclosure triangle that displays or hides the file browser.

```
public abstract void panelWillExpand(Object sender, boolean expanding)
```

NSScreen

Inherits from	NSObject
Package:	com.apple.cocoa.application
Companion guide	The Drawing Environment

Overview

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` (page 1248) 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 application object should be created before you use the methods in this class, so that the application object can make the necessary connection to the window system. You can make sure the application object exists by invoking NSApplication's `sharedApplication` (page 105) method, which creates it if necessary. If you created your application with Xcode, the application object is automatically created for you during initialization.

Note: NSScreen is for getting information about the available displays only. If you need additional information or want to change the attributes relating to a display, you must use Quartz Services. For more information, see *Quartz Display Services Reference*.

Tasks

Constructors

`NSScreen` (page 1248)
Throws an exception.

Getting NSScreens

`mainScreen` (page 1248)
Returns an NSScreen object representing the main screen.

`deepestScreen` (page 1248)
Returns an NSScreen object representing the screen that can best represent color.

screens (page 1249)
Returns an array of NSScreen objects representing all of the screens available on the system.

Reading Screen Information

depth (page 1249)
Returns the receiver's current depth, including whether it can display color.

frame (page 1250)
Returns the dimensions and location of the receiver in an NSRect.

supportedWindowDepths (page 1250)
Returns a zero-terminated array of the window depths supported by the receiver.

deviceDescription (page 1249)
Returns the device dictionary for the screen.

userSpaceScaleFactor (page 1250)
Returns the scaling factor from user space to device space on the screen represented by the receiver.

visibleFrame (page 1250)
Returns the current location and dimensions of the visible screen.

Constructors

NSScreen

Throws an exception.

```
public NSScreen()
```

Discussion
Use mainScreen (page 1248) instead.

Static Methods

deepestScreen

Returns an NSScreen object representing the screen that can best represent color.

```
public static NSScreen deepestScreen()
```

Discussion
This method always returns an object, even if there is only one screen and it is not a color screen.

mainScreen

Returns an NSScreen object representing the main screen.

```
public static NSScreen mainScreen()
```

Discussion

The main screen is the screen with the key window. To obtain the menu bar screen use `NSScreen.screens().objectAtIndex(0)` (after checking that the `screens` array is not empty).

screens

Returns an array of NSScreen objects representing all of the screens available on the system.

```
public static NSArray screens()
```

Discussion

Throws a `WindowServerCommunicationException` if the screen's information can't be obtained from the window system. When the display configuration is changed, `ApplicationDidChangeScreenParametersNotification` (page 140) is sent by the default notification center.

The first screen in the `screens` array is always the "zero" screen. To obtain the menu bar screen use `NSScreen.screens().objectAtIndex(0)` (after checking that the `screens` array is not empty).

The array should not be cached, as screens can be dynamically reconfigured, and a display can be added or removed.

Instance Methods

depth

Returns the receiver's current depth, including whether it can display color.

```
public int depth()
```

Discussion

The return value is not directly usable; you must pass it to one of the static methods of the NSGraphics class to obtain concrete values.

See Also

`bitsPerPixelFromDepth` (page 717) (NSGraphics)

`bitsPerSampleFromDepth` (page 717) (NSGraphics)

`colorSpaceFromDepth` (page 718) (NSGraphics)

deviceDescription

Returns the device dictionary for the screen.

```
public NSDictionary deviceDescription()
```

Discussion

See "Constants" (page 1251) for the list of keys you can use to retrieve values from the returned dictionary.

frame

Returns the dimensions and location of the receiver in an NSRect.

```
public NSRect frame()
```

Discussion

This method returns the full screen rectangle at the current resolution and includes any space currently occupied by the menu bar and dock.

See Also

`visibleFrame` (page 1250)

supportedWindowDepths

Returns a zero-terminated array of the window depths supported by the receiver.

```
public int[] supportedWindowDepths()
```

userSpaceScaleFactor

Returns the scaling factor from user space to device space on the screen represented by the receiver.

```
public float userSpaceScaleFactor()
```

Discussion

This factor is in pixels per point, where a point is always equal to 1/72 of an inch. For example, a scaling factor of 2.0 indicates the display has a resolution of 144 pixels-per-inch.

Availability

Available in Mac OS X v10.4 and later.

visibleFrame

Returns the current location and dimensions of the visible screen.

```
public NSRect visibleFrame()
```

Discussion

The returned rectangle represents the portion of the screen in which it is currently safe to draw your application content. This rectangle is always based on the current user-interface settings and does not include the area currently occupied by the dock and menu bar. Because it is based on the current user -interface settings, the returned rectangle can change between calls and should not be cached.

> **Note:** Even when dock hiding is enabled, the rectangle returned by this method may be smaller than the full screen. The system uses a small boundary area to determine when it should display the dock.

See Also

`frame` (page 1250)

Constants

The following constants are used as keys to retrieve attributes from the display device dictionary.

Constant	Value
NSDeviceBitsPerSample	An NSNumber containing an integer that indicates the current bit depth of each color sample in the device (2-bit, 8-bit, and so on).
NSDeviceColorSpaceName	The device's current color space name. See the NSGraphics Java class specification or the NSGraphics.h header file for a list of possible values.
NSDeviceIsPrinter	"YES" (a string), if present, indicating the device is a printer.
NSDeviceIsScreen	"YES" (a string), if present, indicating the device is a screen.
NSDeviceResolution	An NSValue that contains an NSSize which indicates the device's current resolution in dots per inch (dpi).
NSDeviceSize	An NSValue that contains an NSSize which indicates the device's current size in points.
@"NSScreenNumber"	An NSNumber that contains the CGDirectDisplayID for the screen device. This key is only valid for the device description dictionary for an NSScreen.

NSScroller

Inherits from	NSControl : NSView : NSResponder : NSObject
Implements	NSCoding (NSResponder)
Package:	com.apple.cocoa.application
Companion guide	Drawing and Views

Class at a Glance

An NSScroller object is a user control for scrolling a document view within a container view. You normally don't need to program with NSScrollers, as Interface Builder allows you to fully configure them with an NSScrollView.

Principal Attributes

- Scrolling by small and large increments
- Proportional knob showing visible amount of document

Commonly Used Methods

`hitPart` (page 1258)
Indicates where the user clicked the NSScroller.

`floatValue` (page 451) (NSControl)
Returns the position of the NSScroller's knob.

`setFloatValueAndKnobProportion` (page 1260)
Sets the position and size of the NSScroller's knob.

Overview

An NSScroller controls scrolling of a document view within an NSScrollView's clip view (or potentially another kind of container view). It typically displays a pair of buttons that the user can click to scroll by a small amount (called a line increment or decrement) and Alt-click to scroll by a large amount (called a page increment or decrement), plus a slot containing a knob that the user can drag directly to the desired location. The knob indicates both the position within the document view and, by varying in size within the slot, the amount visible relative to the size of the document view. You can configure whether an NSScroller uses scroll buttons, but it always draws the knob when there's room for it.

Don't use an NSScroller when an NSSlider would be better. A slider represents a range of values for something in the application and lets the user choose a setting. A scroller represents the relative position of the visible portion of a view and lets the user choose which portion to view.

Tasks

Constructors

`NSScroller` (page 1256)
  Creates an NSScroller with a zero-size frame rectangle.

Determining NSScroller Size

`scrollerWidth` (page 1256)
  Returns the width of "normal-sized" instances.

`scrollerWidthForControlSize` (page 1256)
  Returns the width of the scroller based on *controlSize*.

`setControlSize` (page 1259)
  Sets the size of the receiver.

`controlSize` (page 1257)
  Returns the size of the receiver.

Laying out an NSScroller

`setArrowsPosition` (page 1259)
  Sets the location of the scroll buttons within the receiver to *location*, or inhibits their display.

`arrowsPosition` (page 1256)
  Returns the location of the scroll buttons within the receiver, as described under "Constants" (page 1261).

Setting the Knob Position

`setFloatValueAndKnobProportion` (page 1260)
  Sets the position of the knob to *aFloat*, which is a value from 0.0 (indicating the top or left end) to 1.0 (the bottom or right end).

`knobProportion` (page 1259)
  Returns the portion of the knob slot the knob should fill, as a floating-point value from 0.0 (minimal size) to 1.0 (fills the slot).

Calculating Layout

`rectForPart` (page 1259)
Returns the rectangle occupied by *aPart*, which for this method is interpreted literally rather than as an indicator of scrolling direction.

`testPart` (page 1260)
Returns the part that would be hit by a mouse-down event at *aPoint* (expressed in the window's coordinate system).

`checkSpaceForParts` (page 1257)
Checks to see if there is enough room in the receiver to display the knob and buttons.

`usableParts` (page 1261)
Returns a value indicating which parts of the receiver are displayed and usable.

Drawing the Parts

`drawArrow` (page 1257)
Draws the scroll button indicated by *arrow*, which is either `IncrementArrow` (the down or right scroll button) or `DecrementArrow` (up or left).

`drawKnob` (page 1258)
Draws the knob

`drawParts` (page 1258)
Caches images for the scroll buttons and knob.

`highlight` (page 1258)
Highlights or unhighlights the scroll button the user clicked.

Event Handling

`hitPart` (page 1258)
Returns a part code indicating the manner in which the scrolling should be performed.

`trackKnob` (page 1260)
Tracks the knob and sends action messages to the receiver's target.

`trackScrollButtons` (page 1261)
Tracks the scroll buttons and sends action messages to the receiver's target.

Setting Control Tint

`controlTint` (page 1257)
Returns the receiver's control tint.

`setControlTint` (page 1260)
Sets the receiver's control tint.

Constructors

NSScroller

Creates an NSScroller with a zero-size frame rectangle.

```
public NSScroller()
```

Creates an NSScroller with *frameRect* as its frame rectangle.

```
public NSScroller(NSRect frameRect)
```

Static Methods

scrollerWidth

Returns the width of "normal-sized" instances.

```
public static float scrollerWidth()
```

Discussion
NSScrollView uses this value to lay out its components. Subclasses that use a different width should override this method.

scrollerWidthForControlSize

Returns the width of the scroller based on *controlSize*.

```
public static float scrollerWidthForControlSize(int controlSize)
```

Discussion
Valid values for *controlSize* are described in NSCell's "Constants" (page 337).

Instance Methods

arrowsPosition

Returns the location of the scroll buttons within the receiver, as described under "Constants" (page 1261).

```
public int arrowsPosition()
```

See Also
`setArrowsPosition` (page 1259)

checkSpaceForParts

Checks to see if there is enough room in the receiver to display the knob and buttons.

```
public void checkSpaceForParts()
```

Discussion

`usableParts` (page 1261) returns the state calculated by this method. You should never need to invoke this method; it's invoked automatically whenever the NSScroller's size changes.

controlSize

Returns the size of the receiver.

```
public int controlSize()
```

Discussion

Valid return values are described in "Constants" (page 1261).

See Also

`setControlSize` (page 1259)

controlTint

Returns the receiver's control tint.

```
public int controlTint()
```

Discussion

Valid return values are described in "Constants" (page 1261).

See Also

`setControlTint` (page 1260)

drawArrow

Draws the scroll button indicated by *arrow*, which is either `IncrementArrow` (the down or right scroll button) or `DecrementArrow` (up or left).

```
public void drawArrow(int arrow, boolean flag)
```

Discussion

If *flag* is `true`, the button is drawn highlighted; otherwise it's drawn normally. You should never need to invoke this method directly, but may wish to override it to customize the appearance of scroll buttons.

See Also

`drawKnob` (page 1258)

`rectForPart` (page 1259)

drawKnob

Draws the knob

```
public void drawKnob()
```

Discussion
. You should never need to invoke this method directly, but may wish to override it to customize the appearance of the knob.

See Also
`drawArrow` (page 1257)
`rectForPart` (page 1259)

drawParts

Caches images for the scroll buttons and knob.

```
public void drawParts()
```

Discussion
It's invoked only once when the NSScroller is created. You may want to override this method if you alter the look of the NSScroller, but you should never invoke it directly.

highlight

Highlights or unhighlights the scroll button the user clicked.

```
public void highlight(boolean flag)
```

Discussion
The receiver invokes this method while tracking the mouse; you should not invoke it directly. If *`flag`* is `true`, the appropriate part is drawn highlighted; otherwise it's drawn normally.

See Also
`drawArrow` (page 1257)
`rectForPart` (page 1259)

hitPart

Returns a part code indicating the manner in which the scrolling should be performed.

```
public int hitPart()
```

Discussion
See "Constants" (page 1261) for a list of part codes.

This method is typically invoked by an NSScrollView to determine how to scroll its document view when it receives an action message from the NSScroller.

knobProportion

Returns the portion of the knob slot the knob should fill, as a floating-point value from 0.0 (minimal size) to 1.0 (fills the slot).

```
public float knobProportion()
```

rectForPart

Returns the rectangle occupied by *aPart*, which for this method is interpreted literally rather than as an indicator of scrolling direction.

```
public NSRect rectForPart(int aPart)
```

Discussion

See "Constants" (page 1261) for a list of possible values for *aPart*.

Note the interpretations of `DecrementPage` and `IncrementPage`. The actual part of an NSScroller that causes page-by-page scrolling varies, so as a convenience these part codes refer to useful parts different from the scroll buttons.

Returns `NSRect.ZeroRect` if the part requested isn't present on the receiver.

See Also

`hitPart` (page 1258)
`testPart` (page 1260)
`usableParts` (page 1261)

setArrowsPosition

Sets the location of the scroll buttons within the receiver to *location*, or inhibits their display.

```
public void setArrowsPosition(int location)
```

Discussion

See "Constants" (page 1261) for a list of possible values for *location*.

See Also

`arrowsPosition` (page 1256)

setControlSize

Sets the size of the receiver.

```
public void setControlSize(int controlSize)
```

Discussion

Valid values for *controlSize* are described in "Constants" (page 1261).

See Also

`controlSize` (page 1257)

setControlTint

Sets the receiver's control tint.

```
public void setControlTint(int controlTint)
```

Discussion

Valid values for `controlTint` are described in "Constants" (page 1261).

See Also

`controlTint` (page 1257)

setFloatValueAndKnobProportion

Sets the position of the knob to `aFloat`, which is a value from 0.0 (indicating the top or left end) to 1.0 (the bottom or right end).

```
public void setFloatValueAndKnobProportion(float aFloat, float knobProp)
```

Discussion

Also sets the proportion of the knob slot filled by the knob to `knobProp`, also a value from 0.0 (minimal size) to 1.0 (fills the slot).

See Also

`floatValue` (page 451) (NSControl)

`knobProportion` (page 1259)

testPart

Returns the part that would be hit by a mouse-down event at `aPoint` (expressed in the window's coordinate system).

```
public int testPart(NSPoint aPoint)
```

Discussion

See "Constants" (page 1261) for a list of possible return values.

Note the interpretations of `DecrementPage` and `IncrementPage`. The actual part of an NSScroller that causes page-by-page scrolling varies, so as a convenience these part codes refer to useful parts different from the scroll buttons.

See Also

`hitPart` (page 1258)

`rectForPart` (page 1259)

trackKnob

Tracks the knob and sends action messages to the receiver's target.

```
public void trackKnob(NSEvent theEvent)
```

Discussion
This method is invoked automatically when the receiver receives *theEvent* mouse-down event in the knob; you should not invoke it directly.

trackScrollButtons

Tracks the scroll buttons and sends action messages to the receiver's target.

```
public void trackScrollButtons(NSEvent theEvent)
```

Discussion
This method is invoked automatically when the receiver receives *theEvent* mouse-down event in a scroll button; you should not invoke this method directly.

usableParts

Returns a value indicating which parts of the receiver are displayed and usable.

```
public int usableParts()
```

Discussion
See "Constants" (page 1261) for a list of possible values.

See Also
`checkSpaceForParts` (page 1257)
`arrowsPosition` (page 1256)

Constants

These constants specify the different parts of the scroller:

Constant	Description
`Knob`	Directly to the NSScroller's value, as given by `floatValue` (page 451).
`KnobSlot`	Directly to the NSScroller's value, as given by `floatValue` (page 451).
`DecrementLine`	Up or left by a small amount.
`DecrementPage`	Up or left by a large amount.
`IncrementLine`	Down or right by a small amount.
`IncrementPage`	Down or right by a large amount.
`NoPart`	Don't scroll at all.

These constants describe the two scroller buttons and are used by `drawArrow` (page 1257):

Constant	Description
IncrementArrow	The down or right scroll button.
DecrementArrow	The up or left scroll button.

These constants specify where the scroller's buttons appear and are used by arrowsPosition (page 1256) and setArrowsPosition (page 1259):

Constant	Description
ArrowsMaxEnd	Buttons at bottom or right. This constant has been deprecated.
ArrowsMinEnd	Buttons at top or left. This has been deprecated.
ArrowsDefaultSetting	Contains the information from the AppleScrollBarVariant default value.
ArrowsNone	No buttons.

These constants specify which parts of the scroller are visible:

Constant	Description
NoParts	Scroller has neither a knob nor scroll buttons, only the knob slot.
OnlyArrows	Scroller has only scroll buttons, no knob.
AllParts	Scroller has at least a knob, possibly also scroll buttons.

These constants specify control tints:

Constant	Description
DefaultControlTint	The current default tint setting.
ClearControlTint	Clear control tint.
BlueControlTint	Aqua control tint
GraphiteControlTint	Graphite control tint

These constants specify control size:

Constant	Description
RegularControlSize	The control is sized as regular.
SmallControlSize	The control has a smaller size. This constant is for controls that cannot be resized in one direction, such as push buttons, radio buttons, checkboxes, sliders, scroll bars, pop-up buttons, tabs, and progress indicators. You should use a small system font when using with a small control.

Constant	Description
`MiniControlSize`	The control has a smaller size than `SmallControlSize`.

NSScrollView

Inherits from	NSView : NSResponder : NSObject
Implements	NSCoding (NSResponder)
Package:	com.apple.cocoa.application
Companion guide	Drawing and Views

Class at a Glance

An `NSScrollView` allows the user to scroll a document view that's too large to display in its entirety. In addition to the document view, it displays horizontal and vertical scrollers and rulers (depending on which it's configured to have).

Principal Attributes

- Configurable scrollers
- Configurable rulers
- Small and large increment scrolling
- Dynamic (continuous) scrolling
- Display of a special cursor over its document view

Interface Builder
Drag a scroll view to a window.

Commonly Used Methods

`setDocumentView` (page 1279)
Sets the scroll view's document view.

`setLineScroll` (page 1282)
Sets the amount by which the document view moves during scrolling.

`setRulersVisible` (page 1283)
Displays or hides rulers.

Overview

The NSScrollView class is the central coordinator for the Application Kit's scrolling machinery, composed of this class, NSClipView, and NSScroller. An NSScrollView displays a portion of a document view that's too large to be displayed whole and provides NSScroller scroll bars that allow the user to move the document view within the NSScrollView. Note that, when using an NSClipView within an NSScrollView (the usual configuration), you should issue messages that control background drawing state to the NSScrollView, rather than messaging the NSClipView directly.

Tasks

Constructors

`NSScrollView` (page 1270)
    Creates an NSScrollView with a zero-sized frame rectangle.

Calculating Layout

`contentSizeForFrameSize` (page 1270)
    Returns the size of a content view for an NSScrollView whose frame size is *`frameSize`*.

`frameSizeForContentSize` (page 1270)
    Returns the frame size of an NSScrollView that contains a content view whose size is *`contentSize`*.

Determining Component Sizes

`contentSize` (page 1272)
    Returns the size of the receiver's content view.

`documentVisibleRect` (page 1273)
    Returns the portion of the document view, in its own coordinate system, visible through the receiver's content view.

Managing Graphics Attributes

`setBackgroundColor` (page 1278)
    Sets the color of the content view's background to *`aColor`*.

`backgroundColor` (page 1272)
    Returns the content view's background color.

`drawsBackground` (page 1273)
    Returns `true` if the receiver cell fills the background with its background color; otherwise, `false`.

`setDrawsBackground` (page 1279)
    Sets whether the receiver draws its background.

setBorderType (page 1278)
Sets the border type of the receiver to *borderType*.

borderType (page 1272)
Returns a value that represents the type of border surrounding the receiver; see the description of setBorderType (page 1278) for a list of possible values.

setDocumentCursor (page 1279)
Sets the cursor used when the cursor is over the content view to *aCursor*, by sending setDocumentCursor (page 1279) to the content view.

documentCursor (page 1272)
Returns the content view's document cursor.

Managing the Scrolled Views

setContentView (page 1278)
Sets the receiver's content view, the view that clips the document view, to *aView*.

contentView (page 1272)
Returns the receiver's content view, the view that clips the document view.

setDocumentView (page 1279)
Sets the receiver's document view to *aView*.

documentView (page 1273)
Returns the view the receiver scrolls within its content view.

Managing Scrollers

setHorizontalScroller (page 1282)
Sets the receiver's horizontal scroller to *aScroller*, establishing the appropriate target-action relationships between them.

horizontalScroller (page 1275)
Returns the receiver's horizontal scroller, regardless of whether the receiver is currently displaying it, or null if the receiver has none.

setHasHorizontalScroller (page 1280)
Determines whether the receiver keeps a horizontal scroller.

hasHorizontalScroller (page 1273)
Returns true if the receiver displays a horizontal scroller, false if it doesn't.

setVerticalScroller (page 1285)
Sets the receiver's vertical scroller to *aScroller*, establishing the appropriate target-action relationships between them.

verticalScroller (page 1286)
Returns the receiver's vertical scroller, regardless of whether the receiver is currently displaying it, or null if the receiver has none.

setHasVerticalScroller (page 1280)
Determines whether the receiver keeps a vertical scroller.

hasVerticalScroller (page 1274)
Returns true if the receiver displays a vertical scroller, false if it doesn't.

`setAutohidesScrollers` (page 1277)
Determines whether the receiver automatically hides its scroll bars when they are not needed.

`autohidesScrollers` (page 1271)
Returns `true` when autohiding is set for scroll bars in the receiver.

Managing Rulers

`setRulerViewClass` (page 1271)
Sets the default class to be used for ruler objects in NSScrollViews to *`aClass`*.

`rulerViewClass` (page 1271)
Returns the default class to be used for ruler objects in NSScrollViews.

`setHasHorizontalRuler` (page 1279)
Determines whether the receiver keeps a horizontal ruler object.

`hasHorizontalRuler` (page 1273)
Returns `true` if the receiver maintains a horizontal ruler view, `false` if it doesn't.

`setHorizontalRulerView` (page 1281)
Sets the receiver's horizontal ruler view to *`aRulerView`*.

`horizontalRulerView` (page 1275)
Returns the receiver's horizontal ruler view, regardless of whether the receiver is currently displaying it, or `null` if the receiver has none.

`setHasVerticalRuler` (page 1280)
Determines whether the receiver keeps a vertical ruler object.

`hasVerticalRuler` (page 1274)
Returns `true` if the receiver maintains a vertical ruler view, `false` if it doesn't.

`setVerticalRulerView` (page 1284)
Sets the receiver's vertical ruler view to *`aRulerView`*.

`verticalRulerView` (page 1286)
Returns the receiver's vertical ruler view, regardless of whether the receiver is currently displaying it, or `null` if the receiver has none.

`setRulersVisible` (page 1283)
Determines whether the receiver displays its rulers.

`rulersVisible` (page 1277)
Returns `true` if the receiver was set to show rulers using `setRulersVisible` (page 1283) (whether or not it has rulers at all), `false` if it was set to hide them.

Setting Scrolling Behavior

`setLineScroll` (page 1282)
Sets the horizontal and vertical line scroll amounts to *`aFloat`*.

`lineScroll` (page 1275)
Returns the vertical line scroll amount: the amount by which the receiver scrolls itself vertically when scrolling line by line, expressed in the content view's coordinate system.

`setHorizontalLineScroll` (page 1281)
: Sets the amount by which the receiver scrolls itself horizontally when scrolling line by line to *aFloat*, expressed in the content view's coordinate system.

`horizontalLineScroll` (page 1274)
: Returns the amount by which the receiver scrolls itself horizontally when scrolling line by line, expressed in the content view's coordinate system.

`setVerticalLineScroll` (page 1283)
: Sets the amount by which the receiver scrolls itself vertically when scrolling line by line to *aFloat*, expressed in the content view's coordinate system.

`verticalLineScroll` (page 1285)
: Returns the amount by which the receiver scrolls itself vertically when scrolling line by line, expressed in the content view's coordinate system.

`setPageScroll` (page 1282)
: Sets the horizontal and vertical page scroll amounts to *aFloat*.

`pageScroll` (page 1276)
: Returns the vertical page scroll amount: the amount of the document view kept visible when scrolling vertically page by page, expressed in the content view's coordinate system.

`setHorizontalPageScroll` (page 1281)
: Sets the amount of the document view kept visible when scrolling horizontally page by page to *aFloat*, expressed in the content view's coordinate system.

`horizontalPageScroll` (page 1275)
: Returns the amount of the document view kept visible when scrolling horizontally page by page, expressed in the content view's coordinate system.

`setVerticalPageScroll` (page 1284)
: Sets the amount of the document view kept visible when scrolling vertically page by page to *aFloat*, expressed in the content view's coordinate system.

`verticalPageScroll` (page 1285)
: Returns the amount of the document view kept visible when scrolling vertically page by page, expressed in the content view's coordinate system.

`setScrollsDynamically` (page 1283)
: Determines whether the receiver redraws its document view while scrolling continuously.

`scrollsDynamically` (page 1277)
: Returns `true` if the receiver redraws its document view while tracking the knob, `false` if it redraws only when the scroller knob is released.

`scrollWheel` (page 1277)
: Scrolls the receiver up or down, in response to the user moving the mouse's scroll wheel specified by *theEvent*.

Updating Display After Scrolling

`reflectScrolledClipView` (page 1276)

Arranging Components

`tile` (page 1285)
Lays out the components of the receiver: the content view, the scrollers, and the ruler views.

Constructors

NSScrollView

Creates an NSScrollView with a zero-sized frame rectangle.

```
public NSScrollView()
```

Creates an NSScrollView with *frameRect* as its frame rectangle.

```
public NSScrollView(NSRect frameRect)
```

Static Methods

contentSizeForFrameSize

Returns the size of a content view for an NSScrollView whose frame size is *frameSize*.

```
public static NSSize contentSizeForFrameSize(NSSize frameSize, boolean hFlag,
    boolean vFlag, int borderType)
```

Discussion

hFlag and *vFlag* indicate whether a horizontal or vertical scroller, respectively, is present. If either flag is `true` then the content size is reduced in the appropriate dimension by the width of an NSScroller. The *borderType* argument indicates the appearance of the NSScrollView's edge, which also affects the content size; see the description of `setBorderType` (page 1278) for a list of possible values.

For an existing NSScrollView, you can simply use the `contentSize` (page 1272) method.

See Also

`frameSizeForContentSize` (page 1270)
`scrollerWidth` (page 1256) (NSScroller)

frameSizeForContentSize

Returns the frame size of an NSScrollView that contains a content view whose size is *contentSize*.

```
public static NSSize frameSizeForContentSize(NSSize contentSize, boolean hFlag,
    boolean vFlag, int borderType)
```

Discussion
The `hFlag` and `vFlag` arguments indicate whether a horizontal or vertical scroller, respectively, is present. If either flag is `true` then the frame size is increased in the appropriate dimension by the width of an NSScroller. `borderType` indicates the appearance of the NSScrollView's edge, which also affects the frame size; see the description of `setBorderType` (page 1278) for a list of possible values.

For an existing NSScrollView, you can simply use the `frame` method and extract its size.

See Also
`contentSizeForFrameSize` (page 1270)
`scrollerWidth` (page 1256) (NSScroller)

rulerViewClass

Returns the default class to be used for ruler objects in NSScrollViews.

```
public static Class rulerViewClass()
```

Discussion
This class is normally NSRulerView.

See Also
`setRulerViewClass` (page 1271)

setRulerViewClass

Sets the default class to be used for ruler objects in NSScrollViews to *aClass*.

```
public static void setRulerViewClass(Class aClass)
```

Discussion
This class is normally NSRulerView, but you can use this method to set it to a custom subclass of NSRulerView.

This method simply sets a global variable private to NSScrollView. Subclasses of NSScrollView should override both this method and `rulerViewClass` (page 1271) to store their ruler view classes in private variables.

See Also
`rulerViewClass` (page 1271)

Instance Methods

autohidesScrollers

Returns `true` when autohiding is set for scroll bars in the receiver.

```
public boolean autohidesScrollers()
```

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

See Also
`setAutohidesScrollers` (page 1277)

backgroundColor

Returns the content view's background color.

```
public NSColor backgroundColor()
```

See Also
`setBackgroundColor` (page 1278)
`backgroundColor` (page 345) (NSClipView)

borderType

Returns a value that represents the type of border surrounding the receiver; see the description of `setBorderType` (page 1278) for a list of possible values.

```
public int borderType()
```

contentSize

Returns the size of the receiver's content view.

```
public NSSize contentSize()
```

See Also
`contentSizeForFrameSize` (page 1270)

contentView

Returns the receiver's content view, the view that clips the document view.

```
public NSClipView contentView()
```

See Also
`setContentView` (page 1278)
`documentView` (page 1273)

documentCursor

Returns the content view's document cursor.

```
public NSCursor documentCursor()
```

See Also
`setDocumentCursor` (page 1279)
`documentCursor` (page 345) (NSClipView)

documentView

Returns the view the receiver scrolls within its content view.

```
public NSView documentView()
```

See Also
`setDocumentView` (page 1279)
`documentView` (page 346) (NSClipView)

documentVisibleRect

Returns the portion of the document view, in its own coordinate system, visible through the receiver's content view.

```
public NSRect documentVisibleRect()
```

See Also
`documentVisibleRect` (page 346) (NSClipView)
`visibleRect` (page 1786) (NSView)

drawsBackground

Returns `true` if the receiver cell fills the background with its background color; otherwise, `false`.

```
public boolean drawsBackground()
```

hasHorizontalRuler

Returns `true` if the receiver maintains a horizontal ruler view, `false` if it doesn't.

```
public boolean hasHorizontalRuler()
```

Discussion
Display of rulers is controlled using the `setRulersVisible` (page 1283) method.

See Also
`horizontalRulerView` (page 1275)
`setHasHorizontalRuler` (page 1279)
`hasVerticalRuler` (page 1274)
`rulerViewClass` (page 1271)

hasHorizontalScroller

Returns `true` if the receiver displays a horizontal scroller, `false` if it doesn't.

```
public boolean hasHorizontalScroller()
```

See Also
`horizontalScroller` (page 1275)

setHasHorizontalScroller (page 1280)

hasVerticalScroller (page 1274)

hasVerticalRuler

Returns true if the receiver maintains a vertical ruler view, false if it doesn't.

```
public boolean hasVerticalRuler()
```

Discussion

Display of rulers is controlled using the setRulersVisible (page 1283) method.

See Also

verticalRulerView (page 1286)

setHasVerticalRuler (page 1280)

hasHorizontalRuler (page 1273)

rulerViewClass (page 1271)

hasVerticalScroller

Returns true if the receiver displays a vertical scroller, false if it doesn't.

```
public boolean hasVerticalScroller()
```

See Also

verticalScroller (page 1286)

setHasVerticalScroller (page 1280)

hasHorizontalScroller (page 1273)

horizontalLineScroll

Returns the amount by which the receiver scrolls itself horizontally when scrolling line by line, expressed in the content view's coordinate system.

```
public float horizontalLineScroll()
```

Discussion

This amount is used when the user clicks the scroll arrows on the horizontal scroll bar without holding down a modifier key.

See Also

setHorizontalLineScroll (page 1281)

verticalLineScroll (page 1285)

setLineScroll (page 1282)

horizontalPageScroll (page 1275)

horizontalPageScroll

Returns the amount of the document view kept visible when scrolling horizontally page by page, expressed in the content view's coordinate system.

```
public float horizontalPageScroll()
```

Discussion

This amount is used when the user clicks the scroll arrows on the horizontal scroll bar while holding down the Option key.

This amount expresses the context that remains when the receiver scrolls by one page, allowing the user to orient to the new display. It differs from the line scroll amount, which indicates how far the document view moves. The page scroll amount is the amount common to the content view before and after the document view is scrolled by one page.

See Also

`setHorizontalPageScroll` (page 1281)

`verticalPageScroll` (page 1285)

`setPageScroll` (page 1282)

`horizontalLineScroll` (page 1274)

horizontalRulerView

Returns the receiver's horizontal ruler view, regardless of whether the receiver is currently displaying it, or `null` if the receiver has none.

```
public NSRulerView horizontalRulerView()
```

Discussion

If the receiver is set to display a horizontal ruler view and doesn't yet have one, this method creates an instance of the ruler view class set using the class method `setRulerViewClass` (page 1271). Display of rulers is controlled using the `setRulersVisible` (page 1283) method.

See Also

`hasHorizontalRuler` (page 1273)

`verticalRulerView` (page 1286)

horizontalScroller

Returns the receiver's horizontal scroller, regardless of whether the receiver is currently displaying it, or `null` if the receiver has none.

```
public NSScroller horizontalScroller()
```

lineScroll

Returns the vertical line scroll amount: the amount by which the receiver scrolls itself vertically when scrolling line by line, expressed in the content view's coordinate system.

```
public float lineScroll()
```

Discussion

This amount is used when the user clicks the scroll arrows on the vertical scroll bar without holding down a modifier key. As part of its implementation, this method calls `verticalLineScroll` (page 1285).

Note that a scroll view can have two different line scroll amounts: `verticalLineScroll` (page 1285) and `horizontalLineScroll` (page 1274). Use this method only if you can be sure they're both the same; for example, you always use `setLineScroll` (page 1282), which sets both amounts to the same value.

See Also

`setLineScroll` (page 1282)
`verticalPageScroll` (page 1285)
`horizontalPageScroll` (page 1275)

pageScroll

Returns the vertical page scroll amount: the amount of the document view kept visible when scrolling vertically page by page, expressed in the content view's coordinate system.

```
public float pageScroll()
```

Discussion

This amount is used when the user clicks the scroll arrows on the vertical scroll bar while holding down the Option key. As part of its implementation, this method calls `verticalPageScroll` (page 1285).

This amount expresses the context that remains when the receiver scrolls by one page, allowing the user to orient to the new display. It differs from the line scroll amount, which indicates how far the document view moves. The page scroll amount is the amount common to the content view before and after the document view is scrolled by one page.

Note that a scroll view can have two different page scroll amounts: `verticalPageScroll` (page 1285) and `horizontalPageScroll` (page 1275). Use this method only if you can be sure they're both the same; for example, you always use `setPageScroll` (page 1282), which sets both amounts to the same value.

See Also

`setPageScroll` (page 1282)
`verticalLineScroll` (page 1285)
`horizontalLineScroll` (page 1274)

reflectScrolledClipView

```
public void reflectScrolledClipView(NSClipView aClipView)
```

Discussion

If *aClipView* is the receiver's content view, adjusts the receiver's scrollers to reflect the size and positioning of its document view. Does nothing if *aClipView* is any other view object (in particular, if it's an NSClipView that isn't the content view).

This method is invoked automatically during scrolling and when an NSClipView's relationship to its document view changes; you should rarely need to invoke it yourself, but may wish to override it for custom updating or other behavior. If you override this method, be sure to call the superclass implementation. If you do not, other controls (such as the current scrollers) may not be updated properly.

See Also

`contentView` (page 1272)

`documentView` (page 1273)

rulersVisible

Returns `true` if the receiver was set to show rulers using `setRulersVisible` (page 1283) (whether or not it has rulers at all), `false` if it was set to hide them.

```
public boolean rulersVisible()
```

See Also

`hasHorizontalRuler` (page 1273)

`hasVerticalRuler` (page 1274)

scrollsDynamically

Returns `true` if the receiver redraws its document view while tracking the knob, `false` if it redraws only when the scroller knob is released.

```
public boolean scrollsDynamically()
```

Discussion

NSScrollView scrolls dynamically by default.

See Also

`setScrollsDynamically` (page 1283)

scrollWheel

Scrolls the receiver up or down, in response to the user moving the mouse's scroll wheel specified by *theEvent*.

```
public void scrollWheel(NSEvent theEvent)
```

setAutohidesScrollers

Determines whether the receiver automatically hides its scroll bars when they are not needed.

```
public void setAutohidesScrollers(boolean flag)
```

Discussion

The horizontal and vertical scroll bars are hidden independently of each other. When autohiding is on and the content of the receiver doesn't extend beyond the size of the clip view on a given axis, the scroller on that axis is removed to leave more room for the content.

Availability

Available in Mac OS X v10.3 and later.

See Also

`autohidesScrollers` (page 1271)

setBackgroundColor

Sets the color of the content view's background to *aColor*.

```
public void setBackgroundColor(NSColor aColor)
```

Discussion

This color is used to paint areas inside the content view that aren't covered by the document view.

See Also

`backgroundColor` (page 1272)

`setBackgroundColor` (page 347) (NSClipView)

setBorderType

Sets the border type of the receiver to *borderType*.

```
public void setBorderType(int borderType)
```

Discussion

borderType may be one of:

```
    NSView.NoBorder
    NSView.LineBorder
    NSView.BezelBorder
    NSView.GrooveBorder
```

See Also

`borderType` (page 1272)

setContentView

Sets the receiver's content view, the view that clips the document view, to *aView*.

```
public void setContentView(NSClipView aView)
```

Discussion

If *aView* has a document view, this method also sets the receiver's document view to be the document view of *aView*. The original content view retains its document view.

See Also

`contentView` (page 1272)

`setDocumentView` (page 1279)

setDocumentCursor

Sets the cursor used when the cursor is over the content view to *aCursor*, by sending `setDocumentCursor` (page 1279) to the content view.

```
public void setDocumentCursor(NSCursor aCursor)
```

See Also

`documentCursor` (page 1272)

setDocumentView

Sets the receiver's document view to *aView*.

```
public void setDocumentView(NSView aView)
```

See Also

`documentView` (page 1273)

`setDocumentView` (page 347) (NSClipView)

setDrawsBackground

Sets whether the receiver draws its background.

```
public void setDrawsBackground(boolean flag)
```

Discussion

If *flag* is false, copy-on-scroll is automatically disabled.

If your NSScrollView encloses an NSClipView sending a `setDrawsBackground:` message with a parameter of false to the NSScrollView has the added effect of sending the NSClipView a `setCopiesOnScroll:` message with a parameter of false. The side effect of sending the `setDrawsBackground:` message directly to the NSClipView instead would be the appearance of "trails" (vestiges of previous drawing) in the document view as it is scrolled.

See Also

`drawsBackground` (page 1273)

`copiesOnScroll` (page 345) (NSClipView)

`setDrawsBackground` (page 348) (NSClipView)

setHasHorizontalRuler

Determines whether the receiver keeps a horizontal ruler object.

```
public void setHasHorizontalRuler(boolean flag)
```

Discussion

If *flag* is `true`, the receiver allocates a horizontal ruler the first time it's needed. Display of rulers is handled independently with the `setRulersVisible` (page 1283) method.

See Also

`hasHorizontalRuler` (page 1273)

horizontalRulerView (page 1275)

setHasVerticalRuler (page 1280)

setHasHorizontalScroller

Determines whether the receiver keeps a horizontal scroller.

```
public void setHasHorizontalScroller(boolean flag)
```

Discussion

If *flag* is `true`, the receiver allocates and displays a horizontal scroller as needed. An NSScrollView by default has neither a horizontal nor a vertical scroller.

See Also

hasHorizontalScroller (page 1273)

horizontalScroller (page 1275)

setHasVerticalScroller (page 1280)

setHasVerticalRuler

Determines whether the receiver keeps a vertical ruler object.

```
public void setHasVerticalRuler(boolean flag)
```

Discussion

If *flag* is `true`, the receiver allocates a vertical ruler the first time it's needed. Display of rulers is handled independently with the `setRulersVisible` (page 1283) method.

See Also

hasVerticalRuler (page 1274)

verticalRulerView (page 1286)

setHasHorizontalRuler (page 1279)

setRulersVisible (page 1283)

setHasVerticalScroller

Determines whether the receiver keeps a vertical scroller.

```
public void setHasVerticalScroller(boolean flag)
```

Discussion

If *flag* is `true`, the receiver allocates and displays a vertical scroller as needed. An NSScrollView by default has neither a vertical nor a horizontal scroller.

See Also

hasVerticalScroller (page 1274)

verticalScroller (page 1286)

setHasHorizontalScroller (page 1280)

setHorizontalLineScroll

Sets the amount by which the receiver scrolls itself horizontally when scrolling line by line to *aFloat*, expressed in the content view's coordinate system.

```
public void setHorizontalLineScroll(float aFloat)
```

Discussion
This amount is the amount used when the user clicks the scroll arrows on the horizontal scroll bar without holding down a modifier key. When displaying text in an NSScrollView, for example, you might set this amount to the height of a single line of text in the default font.

See Also
`lineScroll` (page 1275)
`setPageScroll` (page 1282)

setHorizontalPageScroll

Sets the amount of the document view kept visible when scrolling horizontally page by page to *aFloat*, expressed in the content view's coordinate system.

```
public void setHorizontalPageScroll(float aFloat)
```

Discussion
This amount is used when the user clicks the scroll arrows on the horizontal scroll bar while holding down the Option key.

This amount expresses the context that remains when the receiver scrolls by one page, allowing the user to orient to the new display. It differs from the line scroll amount, which indicates how far the document view moves. The page scroll amount is the amount common to the content view before and after the document view is scrolled by one page. Thus, setting the page scroll amount to 0.0 implies that the entire visible portion of the document view is replaced when a page scroll occurs.

See Also
`pageScroll` (page 1276)
`setLineScroll` (page 1282)

setHorizontalRulerView

Sets the receiver's horizontal ruler view to *aRulerView*.

```
public void setHorizontalRulerView(NSRulerView aRulerView)
```

Discussion
You can use this method to override the default ruler class set using the class method `setRulerViewClass` (page 1271). Display of rulers is controlled using the `setRulersVisible` (page 1283) method.

See Also
`horizontalRulerView` (page 1275)
`setHasHorizontalRuler` (page 1279)
`setVerticalRulerView` (page 1284)

`setRulersVisible` (page 1283)

setHorizontalScroller

Sets the receiver's horizontal scroller to *aScroller*, establishing the appropriate target-action relationships between them.

```
public void setHorizontalScroller(NSScroller aScroller)
```

Discussion

To make sure the scroller is visible, invoke the `setHasHorizontalScroller` (page 1280) method with an argument of `true`.

See Also

`horizontalScroller` (page 1275)

`setVerticalScroller` (page 1285)

setLineScroll

Sets the horizontal and vertical line scroll amounts to *aFloat*.

```
public void setLineScroll(float aFloat)
```

Discussion

The line scroll is the amount by which the receiver scrolls itself when scrolling line by line, expressed in the content view's coordinate system. It's used when the user clicks the scroll arrows without holding down a modifier key. When displaying text in an NSScrollView, for example, you might set this value to the height of a single line of text in the default font.

As part of its implementation, this method calls `setVerticalLineScroll` (page 1283) and `setHorizontalLineScroll` (page 1281).

See Also

`verticalLineScroll` (page 1285)

`horizontalLineScroll` (page 1274)

setPageScroll

Sets the horizontal and vertical page scroll amounts to *aFloat*.

```
public void setPageScroll(float aFloat)
```

Discussion

The page scroll is the amount of the document view kept visible when scrolling page by page to *aFloat*, expressed in the content view's coordinate system. It's used when the user clicks the scroll arrows while holding down the Option key.

This amount expresses the context that remains when the receiver scrolls by one page, allowing the user to orient to the new display. It differs from the line scroll amount, which indicates how far the document view moves. The page scroll amount is the amount common to the content view before and after the document view is scrolled by one page. Thus, setting the page scroll amount to 0.0 implies that the entire visible portion of the document view is replaced when a page scroll occurs.

As part of its implementation, this method calls `setVerticalPageScroll` (page 1284) and `setHorizontalPageScroll` (page 1281).

See Also

`verticalPageScroll` (page 1285)
`verticalLineScroll` (page 1285)

setRulersVisible

Determines whether the receiver displays its rulers.

```
public void setRulersVisible(boolean flag)
```

Discussion

If *flag* is `true`, the receiver displays its rulers (creating them if needed). If *flag* is `false`, the receiver doesn't display its rulers.

See Also

`rulersVisible` (page 1277)
`hasHorizontalRuler` (page 1273)
`hasVerticalRuler` (page 1274)

setScrollsDynamically

Determines whether the receiver redraws its document view while scrolling continuously.

```
public void setScrollsDynamically(boolean flag)
```

Discussion

If *flag* is `true` it does; if *flag* is `false` it redraws only when the scroller knob is released. NSScrollView scrolls dynamically by default.

See Also

`scrollsDynamically` (page 1277)

setVerticalLineScroll

Sets the amount by which the receiver scrolls itself vertically when scrolling line by line to *aFloat*, expressed in the content view's coordinate system.

```
public void setVerticalLineScroll(float aFloat)
```

Discussion

This value is the amount used when the user clicks the scroll arrows on the vertical scroll bar without holding down a modifier key. When displaying text in an NSScrollView, for example, you might set this value to the height of a single line of text in the default font.

See Also

`verticalLineScroll` (page 1285)
`setHorizontalLineScroll` (page 1281)
`lineScroll` (page 1275)
`setVerticalPageScroll` (page 1284)

setVerticalPageScroll

Sets the amount of the document view kept visible when scrolling vertically page by page to *aFloat*, expressed in the content view's coordinate system.

```
public void setVerticalPageScroll(float aFloat)
```

Discussion

This amount is used when the user clicks the scroll arrows on the vertical scroll bar while holding down the Option key.

This amount expresses the context that remains when the receiver scrolls by one page, allowing the user to orient to the new display. It differs from the line scroll amount, which indicates how far the document view moves. The page scroll amount is the amount common to the content view before and after the document view is scrolled by one page. Thus, setting the page scroll amount to 0.0 implies that the entire visible portion of the document view is replaced when a page scroll occurs.

See Also

`verticalPageScroll` (page 1285)
`setHorizontalPageScroll` (page 1281)
`pageScroll` (page 1276)
`setVerticalLineScroll` (page 1283)

setVerticalRulerView

Sets the receiver's vertical ruler view to *aRulerView*.

```
public void setVerticalRulerView(NSRulerView aRulerView)
```

Discussion

You can use this method to override the default ruler class set using the class method `setRulerViewClass` (page 1271). Display of rulers is controlled using the `setRulersVisible` (page 1283) method.

See Also

`verticalRulerView` (page 1286)
`setHasVerticalRuler` (page 1280)
`setHorizontalRulerView` (page 1281)
`setRulersVisible` (page 1283)

setVerticalScroller

Sets the receiver's vertical scroller to *aScroller*, establishing the appropriate target-action relationships between them.

```
public void setVerticalScroller(NSScroller aScroller)
```

Discussion
To make sure the scroller is visible, invoke the `setHasVerticalScroller` (page 1280) method with an argument of `true`.

See Also
`verticalScroller` (page 1286)
`setHorizontalScroller` (page 1282)

tile

Lays out the components of the receiver: the content view, the scrollers, and the ruler views.

```
public void tile()
```

Discussion
You rarely need to invoke this method, but subclasses may override it to manage additional components.

verticalLineScroll

Returns the amount by which the receiver scrolls itself vertically when scrolling line by line, expressed in the content view's coordinate system.

```
public float verticalLineScroll()
```

Discussion
This amount is used when the user clicks the scroll arrows on the vertical scroll bar without holding down a modifier key.

See Also
`setVerticalLineScroll` (page 1283)
`horizontalLineScroll` (page 1274)
`setLineScroll` (page 1282)
`verticalPageScroll` (page 1285)

verticalPageScroll

Returns the amount of the document view kept visible when scrolling vertically page by page, expressed in the content view's coordinate system.

```
public float verticalPageScroll()
```

Discussion
This amount is used when the user clicks the scroll arrows on the vertical scroll bar while holding down the Option key.

This amount expresses the context that remains when the receiver scrolls by one page, allowing the user to orient to the new display. It differs from the line scroll amount, which indicates how far the document view moves. The page scroll amount is the amount common to the content view before and after the document view is scrolled by one page.

See Also

`setVerticalPageScroll` (page 1284)

`horizontalPageScroll` (page 1275)

`setPageScroll` (page 1282)

`verticalLineScroll` (page 1285)

verticalRulerView

Returns the receiver's vertical ruler view, regardless of whether the receiver is currently displaying it, or `null` if the receiver has none.

```
public NSRulerView verticalRulerView()
```

Discussion

If the receiver is set to display a vertical ruler view and doesn't yet have one, this method creates an instance of the ruler view class set using the class method `setRulerViewClass` (page 1271). Display of rulers is controlled using the `setRulersVisible` (page 1283) method.

See Also

`hasVerticalRuler` (page 1274)

`horizontalRulerView` (page 1275)

verticalScroller

Returns the receiver's vertical scroller, regardless of whether the receiver is currently displaying it, or `null` if the receiver has none.

```
public NSScroller verticalScroller()
```

See Also

`hasVerticalScroller` (page 1274)

`setVerticalScroller` (page 1285)

`horizontalScroller` (page 1275)

NSSearchField

Inherits from	NSTextField : NSControl : NSView : NSResponder : NSObject
Implements	NSCoding (NSResponder)
Package:	com.apple.cocoa.application
Availability	Available in Mac OS X v10.3 and later.
Companion guide	Search Fields

Overview

Instances of the NSSearchField class are control objects that specifically designed to "wrap" NSSearchFieldCell instances. NSSearchField exposes only a small part of the programmatic interface presented by NSSearchFieldCell.

For a fuller description of search fields and the API for accessing and managing them, see the class specification of NSSearchFieldCell (page 1291).

Tasks

Constructors

`NSSearchField` (page 1288)
Creates an NSSearchField with a zero-sized frame rectangle.

Managing Recent Searches

`setRecentSearches` (page 1289)
Sets the list of recent search strings to list in the pop-up icon menu of the receiver.

`recentSearches` (page 1288)
Returns the list of recent search strings, as displayed in the search menu, or a list of recent search strings archived under an autosave name.

Managing Autosave Name

`setRecentsAutosaveName` (page 1289)
Sets an autosave name under which the receiver automatically archives the list of recent search strings.

`recentsAutosaveName` (page 1288)
Returns the key under which the prior list of recent search strings has been archived.

Constructors

NSSearchField

Creates an NSSearchField with a zero-sized frame rectangle.

```
public NSSearchField()
```

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

Creates an NSSearchField with *`frameRect`* as its frame rectangle.

```
public NSSearchField(NSRect frameRect)
```

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

Instance Methods

recentsAutosaveName

Returns the key under which the prior list of recent search strings has been archived.

```
public String recentsAutosaveName()
```

Discussion
NSSearchField implements this method by invoking the same method in its NSSearchFieldCell object.

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

See Also
`setRecentsAutosaveName` (page 1289)

recentSearches

Returns the list of recent search strings, as displayed in the search menu, or a list of recent search strings archived under an autosave name.

```
public NSArray recentSearches()
```

Discussion

Returns an empty array if there have been no recent searches and no prior searches have been saved under an autosave name. NSSearchField implements this method by invoking the same method in its NSSearchFieldCell object.

Availability

Available in Mac OS X v10.3 and later.

See Also

`setRecentSearches` (page 1289)

setRecentsAutosaveName

Sets an autosave name under which the receiver automatically archives the list of recent search strings.

```
public void setRecentsAutosaveName(String name)
```

Discussion

If *name* is `null` (the default) or is an empty string, no autosave name is set. NSSearchField implements this method by invoking the same method in its NSSearchFieldCell object.

Availability

Available in Mac OS X v10.3 and later.

See Also

`recentsAutosaveName` (page 1288)

setRecentSearches

Sets the list of recent search strings to list in the pop-up icon menu of the receiver.

```
public void setRecentSearches(NSArray searches)
```

Discussion

You might use this method to set the recent list from an archived copy. NSSearchField implements this method by invoking the same method in its NSSearchFieldCell object.

Availability

Available in Mac OS X v10.3 and later.

See Also

`recentSearches` (page 1288)

NSSearchFieldCell

Inherits from	NSTextFieldCell : NSActionCell : NSCell : NSObject
Implements	NSCoding (NSCell)
Package:	com.apple.cocoa.application
Availability	Available in Mac OS X v10.3 and later.
Companion guide	Search Fields

Overview

The NSSearchFieldCell class defines the programmatic interface for text fields that are optimized for text-based searches. An NSSearchFieldCell object is "wrapped" by an NSSearchField (page 1287) control object, which directly inherits from the NSTextField class. The search field implemented by these classes presents a standard user interface for searches, including a search button, a cancel button, and a pop-up icon menu for listing recent search strings and custom search categories.

When the user types and then pauses, the cell's action message is sent to its target. You can query the cell's string value for the current text to search for. Do not rely on the sender of the action to be an NSMenu object because the menu may change. If you need to change the menu, modify the search menu template and call the `setSearchMenuTemplate:` method to update.

Tasks

Constructors

`NSSearchFieldCell` (page 1293)
    Creates an empty NSSearchFieldCell.

Managing Buttons

`setSearchButtonCell` (page 1299)
    Sets the button cell used to display the search-button image to *`cell`*.

`searchButtonCell` (page 1296)
    Returns the button cell used for the receiver's search button.

`resetSearchButtonCell` (page 1295)
: Resets the search button cell to its default attributes, including target, action, and regular and pressed images.

`setCancelButtonCell` (page 1298)
: Sets the button cell used to display the cancel-button image to *cell*.

`cancelButtonCell` (page 1294)
: Returns the cell object currently used as the receiver's cancel button.

`resetCancelButtonCell` (page 1295)
: Resets the cancel button cell to its default attributes, including target, action, and regular and pressed images.

Custom Layout

`searchTextRectForBounds` (page 1297)
: Passes in the current bounding rectangle (*rect*) for the search text field and allows subclasses to return a new bounding rectangle for the text-field cell object.

`searchButtonRectForBounds` (page 1296)
: Passes in the current bounding rectangle (*rect*) for the search button and allows subclasses to return a new bounding rectangle for the button cell.

`cancelButtonRectForBounds` (page 1294)
: Passes in the current bounding rectangle (*rect*) for the cancel button and allows subclasses to return a new bounding rectangle for the button cell.

Managing Menu Template

`setSearchMenuTemplate` (page 1299)
: Sets to *menu* the menu template the receiver uses to dynamically construct its pop-up icon menu.

`searchMenuTemplate` (page 1296)
: Returns the menu object used by NSSearchFieldCell to construct the search pop-up icon menu dynamically.

Managing Search Mode

`setSendsWholeSearchString` (page 1300)
: Sets whether the receiver sends the search action message when the user clicks the search button or presses return (`true`) or sends the action message at each keystroke (`false`).

`sendsWholeSearchString` (page 1297)
: Returns whether the receiver sends the search action message when the user clicks the search button or clicks return (`true`) or sends the action message at each keystroke (`false`).

`sendsSearchStringImmediately` (page 1297)
: Returns `true` if the cell sends its action message to the target immediately upon notification of any changes to the search field text; otherwise, returns `false`.

`setSendsSearchStringImmediately` (page 1299)
: If *flag* is `true`, the cell sends its action message to the target immediately upon notification of any changes to the search field text.

Managing Recent Search Strings

`setMaximumRecents` (page 1298)
Sets the maximum number of search strings that can appear in the search menu to *maxRecents*.

`maximumRecents` (page 1294)
Returns the maximum number of recent search strings to display in the custom search menu.

`setRecentSearches` (page 1298)
Sets the list of recent search strings to list in the pop-up icon menu of the receiver.

`recentSearches` (page 1295)
Returns the list of recent search strings, as displayed in the search menu, or a list of recent search strings archived under an autosave name.

`setRecentsAutosaveName` (page 1298)
Sets an autosave name under which the receiver automatically archives the list of recent search strings.

`recentsAutosaveName` (page 1294)
Returns the key under which the prior list of recent search strings has been archived.

Constructors

NSSearchFieldCell

Creates an empty NSSearchFieldCell.

```
public NSSearchFieldCell()
```

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

Creates an NSSearchFieldCell initialized with *aString* and set to have the cell's default menu.

```
public NSSearchFieldCell(String aString)
```

Discussion
If no field editor has been created, one is created.

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

Creates an NSSearchFieldCell initialized with *anImage* and set to have the cell's default menu.

```
public NSSearchFieldCell(NSImage anImage)
```

Discussion
If *anImage* is `null`, no image is set.

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

Instance Methods

cancelButtonCell

Returns the cell object currently used as the receiver's cancel button.

```
public NSButtonCell cancelButtonCell()
```

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

See Also
`setCancelButtonCell` (page 1298)
`resetCancelButtonCell` (page 1295)

cancelButtonRectForBounds

Passes in the current bounding rectangle (*rect*) for the cancel button and allows subclasses to return a new bounding rectangle for the button cell.

```
public NSRect cancelButtonRectForBounds(NSRect rect)
```

Discussion
Subclasses of NSSearchFieldCell can override this method to get the current size of the cancel button and to specify a custom layout for it.

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

See Also
`searchButtonRectForBounds` (page 1296)
`searchTextRectForBounds` (page 1297)

maximumRecents

Returns the maximum number of recent search strings to display in the custom search menu.

```
public int maximumRecents()
```

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

See Also
`setMaximumRecents` (page 1298)

recentsAutosaveName

Returns the key under which the prior list of recent search strings has been archived.

```
public String recentsAutosaveName()
```

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

See Also
`setRecentsAutosaveName` (page 1298)

recentSearches

Returns the list of recent search strings, as displayed in the search menu, or a list of recent search strings archived under an autosave name.

```
public NSArray recentSearches()
```

Discussion
Returns an empty array if there have been no recent searches and no prior searches have been saved under an autosave name.

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

See Also
`setRecentsAutosaveName` (page 1298)
`setRecentSearches` (page 1298)

resetCancelButtonCell

Resets the cancel button cell to its default attributes, including target, action, and regular and pressed images.

```
public native void resetCancelButtonCell()
```

Discussion
By default, when users click the cancel button, the `delete` action message is sent up the responder chain to the first NSText object that can handle it. This method gives you a way to customize the cancel button for specific situations and then reset the button defaults without having to know what they are.

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

See Also
`setCancelButtonCell` (page 1298)
`cancelButtonCell` (page 1294)

resetSearchButtonCell

Resets the search button cell to its default attributes, including target, action, and regular and pressed images.

```
public void resetSearchButtonCell()
```

Discussion

By default, when users click the search button or press the Return key, the action defined for the receiver is sent to its designated target. This method gives you a way to customize the search button for specific situations and then reset the button defaults without having to know what they are.

Availability

Available in Mac OS X v10.3 and later.

See Also

`setSearchButtonCell` (page 1299)

`searchButtonCell` (page 1296)

searchButtonCell

Returns the button cell used for the receiver's search button.

```
public NSButtonCell searchButtonCell()
```

Availability

Available in Mac OS X v10.3 and later.

See Also

`setSearchButtonCell` (page 1299)

`resetSearchButtonCell` (page 1295)

searchButtonRectForBounds

Passes in the current bounding rectangle (*rect*) for the search button and allows subclasses to return a new bounding rectangle for the button cell.

```
public NSRect searchButtonRectForBounds(NSRect rect)
```

Discussion

Subclasses of NSSearchFieldCell can override this method to get the current size of the search button and to specify a custom layout for it.

Availability

Available in Mac OS X v10.3 and later.

See Also

`cancelButtonRectForBounds` (page 1294)

`searchTextRectForBounds` (page 1297)

searchMenuTemplate

Returns the menu object used by NSSearchFieldCell to construct the search pop-up icon menu dynamically.

```
public NSMenu searchMenuTemplate()
```

Availability

Available in Mac OS X v10.3 and later.

See Also

`setSearchMenuTemplate` (page 1299)

searchTextRectForBounds

Passes in the current bounding rectangle (*`rect`*) for the search text field and allows subclasses to return a new bounding rectangle for the text-field cell object.

```
public NSRect searchTextRectForBounds(NSRect rect)
```

Discussion

Subclasses of NSSearchFieldCell can override this method to get the current size of the search text field and to specify a custom layout for it.

Availability

Available in Mac OS X v10.3 and later.

See Also

`cancelButtonRectForBounds` (page 1294)

`searchButtonRectForBounds` (page 1296)

sendsSearchStringImmediately

Returns `true` if the cell sends its action message to the target immediately upon notification of any changes to the search field text; otherwise, returns `false`.

```
public boolean sendsSearchStringImmediately()
```

Discussion

Availability

Available in Mac OS X v10.4 and later.

See Also

`setSendsSearchStringImmediately` (page 1299)

sendsWholeSearchString

Returns whether the receiver sends the search action message when the user clicks the search button or clicks return (`true`) or sends the action message at each keystroke (`false`).

```
public boolean sendsWholeSearchString()
```

Discussion

The default is `false`.

Availability

Available in Mac OS X v10.3 and later.

See Also

`setSendsWholeSearchString` (page 1300)

setCancelButtonCell

Sets the button cell used to display the cancel-button image to *cell*.

```
public void setCancelButtonCell(NSButtonCell cell)
```

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

See Also
cancelButtonCell (page 1294)
resetCancelButtonCell (page 1295)

setMaximumRecents

Sets the maximum number of search strings that can appear in the search menu to *maxRecents*.

```
public void setMaximumRecents(int maxRecents)
```

Discussion
When the limit is exceeded, the oldest search string on the menu is dropped. *maxRecents* can be any integer between zero and 254. Specifying any integer less than zero requests the default, which is ten. Specifying any integer greater than 254 sets the maximum to 254.

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

See Also
maximumRecents (page 1294)

setRecentsAutosaveName

Sets an autosave name under which the receiver automatically archives the list of recent search strings.

```
public void setRecentsAutosaveName(String name)
```

Discussion
If *name* is `null` (the default) or an empty string, no autosave name is set.

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

See Also
recentsAutosaveName (page 1294)
setRecentSearches (page 1298)

setRecentSearches

Sets the list of recent search strings to list in the pop-up icon menu of the receiver.

```
public void setRecentSearches(NSArray searches)
```

Discussion
You might use this method to set the recent list from an archived copy.

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

See Also
`recentSearches` (page 1295)

setSearchButtonCell

Sets the button cell used to display the search-button image to *cell*.

```
public void setSearchButtonCell(NSButtonCell cell)
```

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

See Also
`searchButtonCell` (page 1296)
`resetSearchButtonCell` (page 1295)

setSearchMenuTemplate

Sets to *menu* the menu template the receiver uses to dynamically construct its pop-up icon menu.

```
public void setSearchMenuTemplate(NSMenu menu)
```

Discussion
The receiver looks for the tag constants described in "Constants" (page 1300) to determine how to populate the menu with items related to recent searches. See "Configuring a Search Menu" for a sample of how you might set up the search menu template.

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

See Also
`searchMenuTemplate` (page 1296)

setSendsSearchStringImmediately

If *flag* is `true`, the cell sends its action message to the target immediately upon notification of any changes to the search field text.

```
public void setSendsSearchStringImmediately(boolean flag)
```

Discussion
If *flag* is `false`, the cell pauses briefly before sending its action message to give the user the opportunity to type more text into the search field.

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

See Also

`sendsSearchStringImmediately` (page 1297)

setSendsWholeSearchString

Sets whether the receiver sends the search action message when the user clicks the search button or presses return (`true`) or sends the action message at each keystroke (`false`).

```
public void setSendsWholeSearchString(boolean flag)
```

Availability

Available in Mac OS X v10.3 and later.

See Also

`sendsWholeSearchString` (page 1297)

Constants

NSSearchFieldCell defines the following tag constants for identifying special menu items in the search-menu template set by `setSearchMenuTemplate` (page 1299). When NSSearchFieldCell dynamically constructs the actual search menu from this template, it shows or hides the tagged items as directed.

Constant	Description
`SearchFieldRecentsTitleMenuItemTag`	Identifies the menu item that is the title of the menu group for recent search strings. It is hidden if there are no recent strings. You may use this tagged item for separator characters that also do not appear if there are no recent strings to display.
`SearchFieldRecentsMenuItemTag`	Identifies where recent search strings should appear in the “recents” menu group.
`SearchFieldClearRecentsMenuItemTag`	Identifies the menu item for clearing the current set of recent string searches in the menu. This item is hidden if there are no recent strings.
`SearchFieldNoRecentsMenuItemTag`	Identifies the menu item that describes a lack of recent search strings (for example, “No recent searches”). It is hidden if there have been recent searches.

NSSecureTextField

Inherits from	NSTextField : NSControl : NSView : NSResponder : NSObject
Implements	NSCoding (NSResponder)
Package:	com.apple.cocoa.application
Companion guide	Text Fields

Overview

NSSecureTextField is a subclass of NSTextField that hides its text from display or other access via the user interface. It's suitable for use as a password-entry object or for any item in which a secure value must be kept.

NSSecureTextField uses NSSecureTextFieldCell (page 1303) to implement its user interface.

Tasks

Constructors

`NSSecureTextField` (page 1301)
Creates an NSSecureTextField with a zero-sized frame rectangle.

Constructors

NSSecureTextField

Creates an NSSecureTextField with a zero-sized frame rectangle.

```
public NSSecureTextField()
```

Creates an NSSecureTextField with *frameRect* as its frame rectangle.

```
public NSSecureTextField(NSRect frameRect)
```

NSSecureTextFieldCell

Inherits from	NSTextFieldCell : NSActionCell : NSCell : NSObject
Implements	NSCoding (NSCell)
Package:	com.apple.cocoa.application
Companion guide	Text Fields

Overview

NSSecureTextFieldCell works with NSSecureTextField (page 1301) to provide a text field whose value is guarded from user examination. It overrides the general cell use of the field editor to provide its own field editor, which doesn't display text or allow the user to cut, copy, or paste its value.

Tasks

Constructors

`NSSecureTextFieldCell` (page 1303)
Creates an empty NSSecureTextFieldCell.

Working with Character Echo

`echosBullets` (page 1304)
Returns whether the receiver echoes a bullet character rather than each character typed.

`setEchosBullets` (page 1304)
Sets whether the receiver echoes bullets for each character typed.

Constructors

NSSecureTextFieldCell

Creates an empty NSSecureTextFieldCell.

```
public NSSecureTextFieldCell()
```

Creates an NSSecureTextFieldCell initialized with *aString* and set to have the cell's default menu.

```
public NSSecureTextFieldCell(String aString)
```

Discussion
If no field editor has been created, one is created.

Creates an NSSecureTextFieldCell initialized with *anImage* and set to have the cell's default menu.

```
public NSSecureTextFieldCell(NSImage anImage)
```

Discussion
If *anImage* is `null`, no image is set.

Instance Methods

echosBullets

Returns whether the receiver echoes a bullet character rather than each character typed.

```
public boolean echosBullets()
```

Discussion
Default is `true`.

See Also
`setEchosBullets` (page 1304)

setEchosBullets

Sets whether the receiver echoes bullets for each character typed.

```
public void setEchosBullets(boolean flag)
```

Discussion
If `true`, bullets are echoed. If `false`, the cursor is moved for each character typed, but nothing is displayed.

See Also
`echosBullets` (page 1304)

NSSegmentedCell

Inherits from	NSActionCell : NSCell : NSObject
Implements	NSCoding (NSCell)
Package:	com.apple.cocoa.application
Availability	Available in Mac OS X v10.3 and later.
Companion guide	Segmented Controls Programming Guide for Cocoa

Overview

NSSegmentedCell implements the appearance and behavior of a horizontal button divided into multiple segments. NSSegmentedControl uses a single NSSegmentedCell. For more information on NSSegmentedCell's behavior, see the NSSegmentedControl (page 1317) class specification.

You can customize an NSSegmentedCell using its set… methods; for greater flexibility you can create a subclass and override `drawSegment` (page 1308).

Tasks

Constructors

`NSSegmentedCell` (page 1307)
Creates an empty NSSegmentedCell.

Specifying Number of Segments

`setSegmentCount` (page 1312)
Sets the number of segments in the receiver.

`segmentCount` (page 1310)
Returns the number of segments in the receiver.

Specifying Selected Segment

`setSelected` (page 1312)
Sets the selected state of the specified segment to *flag*.

`selectedSegment` (page 1310)
Returns the index of the currently selected segment, or –1 if no segment is selected.

`selectSegmentWithTag` (page 1310)
Selects the segment with the specified *tag*.

`makeNextSegmentKey` (page 1309)
Selects the next segment.

`makePreviousSegmentKey` (page 1309)
Selects the previous segment.

Specifying Tracking Mode

`setTrackingMode` (page 1313)
Sets the receiver's tracking mode.

`trackingMode` (page 1314)
Returns the receiver's tracking mode.

Working with Individual Segments

`setWidth` (page 1314)
Sets the width, in points, for the specified segment.

`width` (page 1315)
Returns the width, in points, of the specified segment.

`setImage` (page 1311)
Sets the image for the specified segment.

`image` (page 1308)
Returns the image for the specified *segment*, or `null` if the *segment* has no image.

`setLabel` (page 1311)
Sets the label for the specified segment.

`label` (page 1309)
Returns the label for the specified segment, or `null` if the segment has no label.

`isSelected` (page 1308)
Returns `true` if the specified segment for the receiver is selected, `false` otherwise.

`setEnabled` (page 1311)
Sets the enabled state of the specified segment to *flag*.

`isEnabled` (page 1308)
Returns true if the specified segment for the receiver is enabled, false otherwise.

`setMenu` (page 1312)
Sets the menu for the specified segment.

`menu` (page 1310)
    Returns the menu for the specified segment, or `null` if the segment has no menu.

`setToolTip` (page 1313)
    Sets the tool tip for the specified segment.

`toolTip` (page 1314)
    Returns the tool tip for the specified segment, or `null` if the segment has no tool tip.

`setTag` (page 1313)
    Sets the tag for the specified segment.

`tag` (page 1314)
    Returns the tag for the specified segment.

Drawing Custom Content

`drawSegment` (page 1308)
    Draws the specified *segment* in *controlView*.

Constructors

NSSegmentedCell

Creates an empty NSSegmentedCell.

```
public NSSegmentedCell()
```

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

Creates an NSSegmentedCell initialized with *aString* and set to have the cell's default menu

```
public NSSegmentedCell(String aString)
```

Discussion
.

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

Creates an NSSegmentedCell initialized with *anImage* and set to have the cell's default menu.

```
public NSSegmentedCell(NSImage anImage)
```

Discussion
If *anImage* is `null`, no image is set.

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

Instance Methods

drawSegment

Draws the specified *segment* in *controlView*.

```
public void drawSegment(int segment, NSRect frame, NSView controlView)
```

Discussion
The content area of the segment is defined by *frame*.

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

See Also
drawWithFrameInView (page 310) (NSCell)

image

Returns the image for the specified *segment*, or null if the *segment* has no image.

```
public NSImage image(int segment)
```

Discussion
Throws a RangeException if *segment* is out of bounds.

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

See Also
setImage (page 1311)

isEnabled

Returns true if the specified segment for the receiver is enabled, false otherwise.

```
public boolean isEnabled(int segment)
```

Discussion
Throws a RangeException if *segment* is out of bounds.

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

See Also
setEnabled (page 1311)

isSelected

Returns true if the specified segment for the receiver is selected, false otherwise.

```
public boolean isSelected(int segment)
```

Discussion

Throws a `RangeException` if *`segment`* is out of bounds.

Availability

Available in Mac OS X v10.3 and later.

label

Returns the label for the specified segment, or `null` if the segment has no label.

```
public String label(int segment)
```

Discussion

Throws a `RangeException` if *`segment`* is out of bounds.

Availability

Available in Mac OS X v10.3 and later.

See Also

`setLabel` (page 1311)

makeNextSegmentKey

Selects the next segment.

```
public void makeNextSegmentKey()
```

Discussion

The last segment wraps to the first segment.

Availability

Available in Mac OS X v10.3 and later.

See Also

`makePreviousSegmentKey` (page 1309)

makePreviousSegmentKey

Selects the previous segment.

```
public void makePreviousSegmentKey()
```

Discussion

The first segment wraps to the last segment.

Availability

Available in Mac OS X v10.3 and later.

See Also

`makeNextSegmentKey` (page 1309)

menu

Returns the menu for the specified segment, or `null` if the segment has no menu.

```
public NSMenu menu(int segment)
```

Discussion

Throws a `RangeException` if *segment* is out of bounds.

Availability

Available in Mac OS X v10.3 and later.

See Also

`setMenu` (page 1312)

segmentCount

Returns the number of segments in the receiver.

```
public int segmentCount()
```

Availability

Available in Mac OS X v10.3 and later.

See Also

`setSegmentCount` (page 1312)

selectedSegment

Returns the index of the currently selected segment, or –1 if no segment is selected.

```
public int selectedSegment()
```

Discussion

If the receiver allows multiple selections, this method returns the last clicked segment.

Availability

Available in Mac OS X v10.3 and later.

selectSegmentWithTag

Selects the segment with the specified *tag*.

```
public boolean selectSegmentWithTag(int tag)
```

Discussion

Returns `true` if successful; otherwise, returns `false`.

Typically, you use Interface Builder to specify the tag for each segment. You may also set this value programmatically using `setTag` (page 1313).

Availability

Available in Mac OS X v10.4 and later.

See Also

`setTag` (page 1313)

setEnabled

Sets the enabled state of the specified segment to *`flag`*.

```
public void setEnabled(boolean flag, int segment)
```

Discussion

Throws a `RangeException` if *`segment`* is out of bounds.

Availability

Available in Mac OS X v10.3 and later.

See Also

`isEnabled` (page 1308)

setImage

Sets the image for the specified segment.

```
public void setImage(NSImage image, int segment)
```

Discussion

Images are clipped if they do not fit within the space available in *`segment`*. Throws a `RangeException` if *`segment`* is out of bounds.

Availability

Available in Mac OS X v10.3 and later.

See Also

`image` (page 1308)

setLabel

Sets the label for the specified segment.

```
public void setLabel(String label, int segment)
```

Discussion

If *`label`* is too long to fit in the *`segment`*'s width, the text is truncated using an ellipsis. Throws a `RangeException` if *`segment`* is out of bounds.

Availability

Available in Mac OS X v10.3 and later.

See Also

`label` (page 1309)

setMenu

Sets the menu for the specified segment.

```
public void setMenu(NSMenu menu, int segment)
```

Discussion
This allows the *segment* to be used as a pop-up button. If a menu is set, the segment acts as a momentary control and cannot be toggled on by the user. Throws a `RangeException` if *segment* is out of bounds.

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

See Also
`menu` (page 1310)

setSegmentCount

Sets the number of segments in the receiver.

```
public void setSegmentCount(int count)
```

Discussion
If *count* is less than the current segment count, segments are removed from the right. If *count* is greater than the current segment count, empty segments are added to the right. The segment count must be between 0 and 2049.

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

See Also
`segmentCount` (page 1310)

setSelected

Sets the selected state of the specified segment to *flag*.

```
public void setSelected(boolean flag, int segment)
```

Discussion
If the receiver only allows a single selection, this method unselects any other selected segments. Throws a `RangeException` if *segment* is out of bounds.

Sets the receiver's selected segment to *selectedSegment*.

```
public void setSelected(int selectedSegment)
```

Discussion
If the receiver allows multiple selections, this method selects the specified segment using the variant form of the method. Throws a `RangeException` if *selectedSegment* is out of bounds.

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

See Also
`isSelected` (page 1308)
`selectedSegment` (page 1310)

setTag

Sets the tag for the specified segment.

```
public void setTag(int tag, int segment)
```

Discussion
Throws a `RangeException` if *`segment`* is out of bounds.

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

See Also
`tag` (page 1314)

setToolTip

Sets the tool tip for the specified segment.

```
public void setToolTip(String toolTip, int segment)
```

Discussion
Throws a `RangeException` if *`segment`* is out of bounds. Tool tips are currently not displayed.

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

See Also
`toolTip` (page 1314)

setTrackingMode

Sets the receiver's tracking mode.

```
public void setTrackingMode(int trackingMode)
```

Discussion
Possible values for *`trackingMode`* are described in "Constants" (page 1315). The default tracking mode is NSSegmentSwitchTrackingSelectOne.

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

See Also
`trackingMode` (page 1314)

setWidth

Sets the width, in points, for the specified segment.

```
public void setWidth(float width, int segment)
```

Discussion
Specifying `width` of 0 means autosize to fit. If you set a non-zero width no padding is added to the left or right of text, so you should ensure there is enough space for content. Throws a `RangeException` if *segment* is out of bounds.

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

See Also
`width` (page 1315)

tag

Returns the tag for the specified segment.

```
public int tag(int segment)
```

Discussion
Throws a `RangeException` if *segment* is out of bounds.

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

See Also
`setTag` (page 1313)

toolTip

Returns the tool tip for the specified segment, or `null` if the segment has no tool tip.

```
public String toolTip(int segment)
```

Discussion
Throws a `RangeException` if *segment* is out of bounds. Tool tips are currently not displayed.

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

See Also
`setToolTip` (page 1313)

trackingMode

Returns the receiver's tracking mode.

```
public int trackingMode()
```

Discussion
Possible return values are described in "Constants" (page 1315). The default tracking mode is NSSegmentSwitchTrackingSelectOne.

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

See Also
`setTrackingMode` (page 1313)

width

Returns the width, in points, of the specified segment.

```
public float width(int segment)
```

Discussion
Returns 0 if the segment should autofit. Throws a `RangeException` if *`segment`* is out of bounds.

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

See Also
`setWidth` (page 1314)

Constants

NSSegmentedCell defines the following constants that are used to describe the various tracking modes for a cell. You access these values using `setTrackingMode` (page 1313) and `trackingMode` (page 1314).

Constant	Description
`NSSegmentSwitchTrackingSelectOne`	Only one segment may be selected.
`NSSegmentSwitchTrackingSelectAny`	Any segment can be selected.
`NSSegmentSwitchTrackingMomentary`	A segment is selected only when tracking.

NSSegmentedControl

Inherits from	NSControl : NSView : NSResponder : NSObject
Implements	NSCoding (NSResponder)
Package:	com.apple.cocoa.application
Availability	Available in Mac OS X v10.3 and later.
Companion guide	Segmented Controls Programming Guide for Cocoa

Overview

NSSegmentedControl is a subclass of NSControl that implements a horizontal control made of multiple segments.

NSSegmentedControl uses an NSSegmentedCell (page 1305) to implement much of the control's functionality. NSSegmentedControl provides "cover methods" for most of NSSegmentedCell's methods, which invoke the corresponding cell method. NSSegmentedCell methods that don't have covers relate to accessing and setting values for tags and tool tips; programatically setting the key segment; and establishing the mode of the control.

NSSegmentedControl's features include:

- Each segment can have an image, text (label), menu, tooltip, and tag
- Either the whole control or individual segments can be enabled or disabled
- There are three modes: radio button (as illustrated by Finder's view mode selection control), momentary (as illustrated by Safari's toolbar buttons), or any on/off
- Each segment can be either a fixed width or autosized to fit the contents
- If a segment has text and is marked as autosizing, then the text may be truncated so that the control completely fits
- If an image is too large to fit in a segment, it will be clipped
- Full keyboard control of the user interface

Tasks

Constructors

NSSegmentedControl (page 1319)
Creates an NSSegmentedControl with a zero-sized frame rectangle.

Specifying Number of Segments

setSegmentCount (page 1322)
Sets the number of segments in the receiver.

segmentCount (page 1321)
Returns the number of segments in the receiver.

Specifying Selected Segment

setSelected (page 1323)
Sets the receiver's selected segment to *selectedSegment*.

selectedSegment (page 1321)
Returns the index of the currently selected segment, or –1 if no segment is selected.

isSelected (page 1320)
Returns true if the specified segment for the receiver is selected, false otherwise.

Working with Individual Segments

setWidth (page 1323)
Sets the width, in points, for the specified segment.

width (page 1323)
Returns the width, in points, of the specified segment.

setImage (page 1321)
Sets the image for the specified segment.

imageForSegment (page 1319)
Returns the image for the specified *segment*, or null if the segment has no image.

setLabel (page 1322)
Sets the label for the specified segment.

label (page 1320)
Returns the label for the specified segment, or null if the segment has no label.

setMenu (page 1322)
Sets the menu for the specified segment.

menu (page 1320)
Returns the menu for the specified segment, or null if the segment has no menu.

`setEnabled` (page 1321)
Sets the enabled state of the specified segment to *`flag`*.

`isEnabled` (page 1319)
Returns `true` if the specified segment for the receiver is enabled, `false` otherwise.

Constructors

NSSegmentedControl

Creates an NSSegmentedControl with a zero-sized frame rectangle.

```
public NSSegmentedControl()
```

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

Creates an NSSegmentedControl with *`frameRect`* as its frame rectangle.

```
public NSSegmentedControl(NSRect frameRect)
```

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

Instance Methods

imageForSegment

Returns the image for the specified *`segment`*, or `null` if the segment has no image.

```
public NSImage image(int segment)
```

Discussion
Throws a `RangeException` if *`segment`* is out of bounds.

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

See Also
`setImage` (page 1321)

isEnabled

Returns `true` if the specified segment for the receiver is enabled, `false` otherwise.

```
public boolean isEnabled(int segment)
```

Discussion
Throws a `RangeException` if *`segment`* is out of bounds.

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

See Also
`setEnabled` (page 1321)

isSelected

Returns `true` if the specified segment for the receiver is selected, `false` otherwise.

```
public boolean isSelected(int segment)
```

Discussion
Throws a `RangeException` if *`segment`* is out of bounds.

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

See Also
`setSelected` (page 1323)

label

Returns the label for the specified segment, or `null` if the segment has no label.

```
public String label(int segment)
```

Discussion
Throws a `RangeException` if *`segment`* is out of bounds.

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

See Also
`setLabel` (page 1322)

menu

Returns the menu for the specified segment, or `null` if the segment has no menu.

```
public NSMenu menu(int segment)
```

Discussion
Throws a `RangeException` if *`segment`* is out of bounds.

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

See Also
`setMenu` (page 1322)

segmentCount

Returns the number of segments in the receiver.

```
public int segmentCount()
```

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

See Also
`setSegmentCount` (page 1322)

selectedSegment

Returns the index of the currently selected segment, or –1 if no segment is selected.

```
public int selectedSegment()
```

Discussion
If the receiver allows multiple selections, this method returns the last clicked segment.

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

See Also
`setSelected` (page 1323)

setEnabled

Sets the enabled state of the specified segment to *flag*.

```
public void setEnabled(boolean flag, int segment)
```

Discussion
Throws a `RangeException` if *segment* is out of bounds.

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

See Also
`isEnabled` (page 1319)

setImage

Sets the image for the specified segment.

```
public void setImage(NSImage image, int segment)
```

Discussion
Images are clipped if they do not fit within the space available in the segment. Throws a `RangeException` if *`segment`* is out of bounds.

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

See Also
`imageForSegment` (page 1319)

setLabel

Sets the label for the specified segment.

```
public void setLabel(String label, int segment)
```

Discussion
If *`label`* is too long to fit in the segment's width, the text is truncated. Throws a `RangeException` if *`segment`* is out of bounds.

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

See Also
`label` (page 1320)

setMenu

Sets the menu for the specified segment.

```
public void setMenu(NSMenu menu, int segment)
```

Discussion
This allows the segment to be used as a pop-up button. Throws a `RangeException` if *`segment`* is out of bounds.

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

See Also
`menu` (page 1320)

setSegmentCount

Sets the number of segments in the receiver.

```
public void setSegmentCount(int count)
```

Discussion
If *`count`* is less than the current segment count, segments are removed from the right. If *`count`* is greater than the current segment count, empty segments are added to the right. The segment count must be between 0 and 2049.

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

See Also
`segmentCount` (page 1321)

setSelected

Sets the receiver's selected segment to *`selectedSegment`*.

```
public void setSelected(int selectedSegment)
```

Discussion
If the receiver allows multiple selections, this method selects the specified segment. Throws a `RangeException` if *`selectedSegment`* is out of bounds.

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

Sets the selected state of the specified segment to *`flag`*.

```
public void setSelected(boolean flag, int segment)
```

Discussion
If the receiver only allows a single selection, this method unselects any other selected segments. Throws a `RangeException` if *`segment`* is out of bounds.

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

See Also
`selectedSegment` (page 1321)
`isSelected` (page 1320)

setWidth

Sets the width, in points, for the specified segment.

```
public void setWidth(float width, int segment)
```

Discussion
Specifying *`width`* of 0 means autosize to fit. Throws a `RangeException` if *`segment`* is out of bounds.

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

See Also
`width` (page 1323)

width

Returns the width, in points, of the specified segment.

```
public float width(int segment)
```

Discussion

Returns 0 if the segment should autofit. Throws a `RangeException` if *`segment`* is out of bounds.

Availability

Available in Mac OS X v10.3 and later.

See Also

`setWidth` (page 1323)

NSShadow

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.application
Availability	Available in Mac OS X v10.3 and later.
Companion guide	Basic Drawing

Overview

An NSShadow object encapsulates the attributes of a drop shadow to be used in drawing operations.

Shadows are always drawn in default user space (also known as base space). This means that rotations, translations and so on of the current transformation matrix (the CTM) don't affect the resulting shadow. Another way to think about this is that changes to the CTM don't move or change the shadow’s light source.

There are two positional parameters for a shadow, an x-offset and a y-offset of the shadow, expressed as a single NSSize, in default user space units, with positive values being up and to the right. There is one additional parameter, the blur radius, which specifies how much an object's image mask is blurred before it is composited onto the destination. A 0 value means no blur, and larger values give correspondingly larger blurs, again in default user space units.

An NSShadow may be used in one of two ways. First, it may be set, like a color or a font, in which case it is applied to all drawing until another shadow is applied or until the next graphics state is restored. It may also be used as the value for the NSShadowAttributeName text attribute, in which case it will be applied to the glyphs corresponding to the characters bearing this attribute.

Tasks

Constructors

`NSShadow` (page 1326)
Creates an NSShadow object.

Managing a Shadow

`setShadowOffset` (page 1327)
Sets the offset, in default user space units, of the receiver from the original drawing where positive values are up and to the right to *offset*.

`shadowOffset` (page 1328)
Returns the offset, in default user space units, of the receiver from the original drawing where positive values are up and to the right.

`setShadowBlurRadius` (page 1327)
Sets the blur radius, in default user space units, of the receiver to *val*. A `0` value indicates no blur, and larger values give correspondingly larger blurs.

`shadowBlurRadius` (page 1328)
Returns the blur radius of the receiver in default user space units.

`setShadowColor:` (page 1327)
Sets the color for the receiver.

`shadowColor` (page 1328)
Returns the color for the receiver.

Setting the Shadow

`set` (page 1326)
Sets the shadow of subsequent drawing operations to the shadow represented by the receiver.

Constructors

NSShadow

Creates an NSShadow object.

```
public NSShadow()
```

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

Instance Methods

set

Sets the shadow of subsequent drawing operations to the shadow represented by the receiver.

```
public void set()
```

Discussion

The shadow attributes of the receiver are used until another shadow is set or until the graphics state is restored.

Availability

Available in Mac OS X v10.3 and later.

setShadowBlurRadius

Sets the blur radius, in default user space units, of the receiver to *val*. A 0 value indicates no blur, and larger values give correspondingly larger blurs.

```
public void setShadowBlurRadius(float val)
```

Availability

Available in Mac OS X v10.3 and later.

See Also

shadowBlurRadius (page 1328)

setShadowColor:

Sets the color for the receiver.

```
public void setShadowColor(NSColor color)
```

Discussion

The default shadow color is black with an alpha of 1/3. A null *color* indicates the shadow is not to be drawn. Currently only colors convertible to RGBA are supported.

Availability

Available in Mac OS X v10.3 and later.

See Also

shadowColor (page 1328)

setShadowOffset

Sets the offset, in default user space units, of the receiver from the original drawing where positive values are up and to the right to *offset*.

```
public void setShadowOffset(NSSize offset)
```

Availability

Available in Mac OS X v10.3 and later.

See Also

shadowOffset (page 1328)

shadowBlurRadius

Returns the blur radius of the receiver in default user space units.

```
public float shadowBlurRadius()
```

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

See Also
`setShadowBlurRadius` (page 1327)

shadowColor

Returns the color for the receiver.

```
public NSColor shadowColor()
```

Discussion
The default shadow color is black with an alpha of 1/3. A `null` shadow color indicates the shadow is not to be drawn.

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

See Also
`setShadowColor:` (page 1327)

shadowOffset

Returns the offset, in default user space units, of the receiver from the original drawing where positive values are up and to the right.

```
public NSSize shadowOffset()
```

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

See Also
`setShadowOffset` (page 1327)

NSSlider

Inherits from	NSControl : NSView : NSResponder : NSObject
Implements	NSCoding (NSResponder)
Package:	com.apple.cocoa.application
Companion guide	Slider Programming Topics for Cocoa

Overview

An NSSlider displays a range of values for something in the application. Sliders can be vertical or horizontal bars or circular dials. An indicator, or knob, notes the current setting. The user can move the knob in the slider's bar—or rotate the knob in a circular slider—to change the setting.

NSSlider uses NSSliderCell (page 1341) to implement its user interface.

Tasks

Constructors

`NSSlider` (page 1331)
Creates an NSSlider with a zero-sized frame rectangle.

Asking About the Slider's Appearance

`altIncrementValue` (page 1332)
Returns the amount the receiver will change its value when the user drags the knob with the Option key held down.

`image` (page 1333)
This method has been deprecated. Returns `null`.

`knobThickness` (page 1333)
Returns the knob's thickness, in pixels.

`isVertical` (page 1333)
Returns 1 if the receiver is vertical, 0 if it's horizontal, and –1 if the orientation can't be determined (for example, if the slider hasn't been displayed yet).

Changing the Slider's Appearance

`setAltIncrementValue` (page 1335)
Sets the amount by which the NSSliderCell modifies its value when the user Option-drags the knob.

`setImage` (page 1335)
This method has been deprecated. Sets the image the receiver displays in the bar behind its knob.

`setKnobThickness` (page 1336)
This method has been deprecated. Lets you set the knob's thickness, measured in pixels.

Asking About the Slider's Title

`title` (page 1338)
Returns the receiver's title.

`titleCell` (page 1339)
This method has been deprecated. Returns `null`.

`titleColor` (page 1339)
This method has been deprecated. Returns `null`.

`titleFont` (page 1339)
This method has been deprecated. Returns `null`.

Changing the Slider's Title

`setTitle` (page 1337)
This method has been deprecated. Sets the title the receiver displays in the bar behind its knob.

`setTitleCell` (page 1337)
This method has been deprecated. Sets the cell used to draw the receiver's title.

`setTitleColor` (page 1338)
This method has been deprecated. Sets the color used to draw the receiver's title to *`color`*.

`setTitleFont` (page 1338)
This method has been deprecated. Sets the font used to draw the receiver's title.

Asking About the Value Limits

`maxValue` (page 1334)
Returns the maximum value the receiver can send to its target.

`minValue` (page 1334)
Returns the minimum value the receiver can send to its target.

Changing the Value Limits

`setMaxValue` (page 1336)
Sets the maximum value the receiver can send to its target to *`maxValue`*.

setMinValue (page 1336)
Sets the minimum value the receiver can send to its target to *minValue*.

Handling Mouse-down Events

acceptsFirstMouse (page 1332)
Returns true by default, so a single mouse-down event *mouseDownEvent* can simultaneously activate the window and take hold of the slider's knob.

Managing Tick Marks

allowsTickMarkValuesOnly (page 1332)
Returns whether the receiver fixes its values to those values represented by its tick marks.

closestTickMarkValueToValue (page 1332)
Returns the value of the tick mark closest to *aValue*.

indexOfTickMarkAtPoint (page 1333)
Returns the index of the tick mark closest to the location of the receiver represented by *point*.

numberOfTickMarks (page 1334)
Returns the number of tick marks associated with the receiver.

rectOfTickMarkAtIndex (page 1334)
Returns the bounding rectangle of the tick mark identified by *index* (the minimum-value tick mark is at index 0).

setAllowsTickMarkValuesOnly (page 1335)
Sets whether the receiver's values are fixed to the values represented by the tick marks to *flag*.

setNumberOfTickMarks (page 1336)
Sets the number of tick marks displayed by the receiver (which include those assigned to the minimum and maximum values) to *numberOfTickMarks*.

setTickMarkPosition (page 1337)
Sets where tick marks appear relative to the receiver.

tickMarkPosition (page 1338)
Returns how the receiver's tick marks are aligned with it.

tickMarkValueAtIndex (page 1338)
Returns the receiver's value represented by the tick mark at *index* (the minimum-value tick mark has an index of 0).

Constructors

NSSlider

Creates an NSSlider with a zero-sized frame rectangle.

```
public NSSlider()
```

Creates an NSSlider with *frameRect* as its frame rectangle.

```
public NSSlider(NSRect frameRect)
```

Instance Methods

acceptsFirstMouse

Returns `true` by default, so a single mouse-down event *mouseDownEvent* can simultaneously activate the window and take hold of the slider's knob.

```
public boolean acceptsFirstMouse(NSEvent mouseDownEvent)
```

Discussion
If you want the receiver to wait for its own mouse-down event, you must override this method.

allowsTickMarkValuesOnly

Returns whether the receiver fixes its values to those values represented by its tick marks.

```
public boolean allowsTickMarkValuesOnly()
```

Discussion
In its implementation of this method, the receiving NSSlider simply invokes the method of the same name of its NSSliderCell.

See Also
`setAllowsTickMarkValuesOnly` (page 1335)

altIncrementValue

Returns the amount the receiver will change its value when the user drags the knob with the Option key held down.

```
public double altIncrementValue()
```

Discussion
Unless you call `setAltIncrementValue` (page 1335), `altIncrementValue` (page 1332) returns –1.0, and the receiver behaves no differently with the Option key down than with it up.

See Also
`setAltIncrementValue` (page 1335)

closestTickMarkValueToValue

Returns the value of the tick mark closest to *aValue*.

```
public double closestTickMarkValueToValue(double aValue)
```

Discussion

In its implementation of this method, the receiver simply invokes the method of the same name of its NSSliderCell.

See Also

`indexOfTickMarkAtPoint` (page 1333)

image

This method has been deprecated. Returns `null`.

```
public NSImage image()
```

See Also

`setImage` (page 1335)

indexOfTickMarkAtPoint

Returns the index of the tick mark closest to the location of the receiver represented by *`point`*.

```
public int indexOfTickMarkAtPoint(NSPoint point)
```

Discussion

If *`point`* is not within the bounding rectangle (plus an extra pixel of space) of any tick mark, the method returns `NSArray.NotFound`. In its implementation of this method, the receiving NSSlider simply invokes the method of the same name of its NSSliderCell. This method invokes `rectOfTickMarkAtIndex` (page 1334) for each tick mark on the slider until it finds a tick mark containing the point.

See Also

`closestTickMarkValueToValue` (page 1332)

isVertical

Returns 1 if the receiver is vertical, 0 if it's horizontal, and –1 if the orientation can't be determined (for example, if the slider hasn't been displayed yet).

```
public int isVertical()
```

Discussion

A slider is defined as vertical if its height is greater than its width.

knobThickness

Returns the knob's thickness, in pixels.

```
public float knobThickness()
```

Discussion

The thickness is defined to be the extent of the knob along the long dimension of the bar. In a vertical slider, then, a knob's thickness is its height; in a horizontal slider, a knob's thickness is its width.

See Also
`setKnobThickness` (page 1336)

maxValue

Returns the maximum value the receiver can send to its target.

```
public double maxValue()
```

Discussion
A horizontal slider sends its maximum value when the knob is at the right end of the bar; a vertical slider sends it when the knob is at the top.

See Also
`setMaxValue` (page 1336)

minValue

Returns the minimum value the receiver can send to its target.

```
public double minValue()
```

Discussion
A vertical slider sends its minimum value when its knob is at the bottom; a horizontal slider, when its knob is all the way to the left.

See Also
`setMinValue` (page 1336)

numberOfTickMarks

Returns the number of tick marks associated with the receiver.

```
public int numberOfTickMarks()
```

Discussion
The tick marks assigned to the minimum and maximum values are included. In its implementation of this method, the receiving NSSlider simply invokes the method of the same name of its NSSliderCell.

See Also
`setNumberOfTickMarks` (page 1336)

rectOfTickMarkAtIndex

Returns the bounding rectangle of the tick mark identified by *`index`* (the minimum-value tick mark is at index 0).

```
public NSRect rectOfTickMarkAtIndex(int index)
```

Discussion

If no tick mark is associated with *index*, the method throws RangeException. In its implementation of this method, the receiving NSSlider simply invokes the method of the same name of its NSSliderCell.

See Also

indexOfTickMarkAtPoint (page 1333)

setAllowsTickMarkValuesOnly

Sets whether the receiver's values are fixed to the values represented by the tick marks to *flag*.

```
public void setAllowsTickMarkValuesOnly(boolean flag)
```

Discussion

For example, if a slider has a minimum value of 0, a maximum value of 100, and five markers, the allowable values are 0, 25, 50, 75, and 100. When users move the slider's knob, it jumps to the tick mark nearest the cursor when the mouse button is released. This method has no effect if the slider has no tick marks. In its implementation of this method, the receiving NSSlider simply invokes the method of the same name of its NSSliderCell.

See Also

allowsTickMarkValuesOnly (page 1332)

setAltIncrementValue

Sets the amount by which the NSSliderCell modifies its value when the user Option-drags the knob.

```
public void setAltIncrementValue(double increment)
```

Discussion

increment must fit the range of values the slider can represent—for example, if the slider has a minimum value of 5 and a maximum value of 10, increment should be between 0 and 5. If *increment* is outside that range, the value is unchanged.

If you don't call this method, the slider behaves the same with the Option key down as with it up. This is also the result when you call setAltIncrementValue with an increment of –1.

See Also

maxValue (page 1334)

minValue (page 1334)

setImage

This method has been deprecated. Sets the image the receiver displays in the bar behind its knob.

```
public void setImage(NSImage barImage)
```

Discussion

The slider may scale and distort *barImage* to fit inside the bar.

The knob may cover part of the image. If you want the image to be visible all the time, you're better off placing it near the slider.

See Also

`setImage` (page 1335)

setKnobThickness

This method has been deprecated. Lets you set the knob's thickness, measured in pixels.

```
public void setKnobThickness(float thickness)
```

Discussion

The thickness is defined to be the extent of the knob along the long dimension of the bar. In a vertical slider, a knob's thickness is its height; in a horizontal slider, a knob's thickness is its width.

See Also

`knobThickness` (page 1333)

setMaxValue

Sets the maximum value the receiver can send to its target to *maxValue*.

```
public void setMaxValue(double maxValue)
```

Discussion

A horizontal slider sends its maximum value when its knob is all the way to the right; a vertical slider sends its maximum value when its knob is at the top.

See Also

`maxValue` (page 1334)

setMinValue

Sets the minimum value the receiver can send to its target to *minValue*.

```
public void setMinValue(double minValue)
```

Discussion

A horizontal slider sends its minimum value when its knob is all the way to the left; a vertical slider sends its minimum value when its knob is at the bottom.

See Also

`minValue` (page 1334)

setNumberOfTickMarks

Sets the number of tick marks displayed by the receiver (which include those assigned to the minimum and maximum values) to *numberOfTickMarks*.

```
public void setNumberOfTickMarks(int numberOfTickMarks)
```

Discussion

By default, this value is 0, and no tick marks appear. The number of tick marks assigned to a slider, along with the slider's minimum and maximum values, determines the values associated with the tick marks. In its implementation of this method, the receiving NSSlider simply invokes the method of the same name of its NSSliderCell.

See Also

`numberOfTickMarks` (page 1334)

setTickMarkPosition

Sets where tick marks appear relative to the receiver.

```
public void setTickMarkPosition(int position)
```

Discussion

For horizontal sliders, *position* can be `NSSliderCell.TickMarkBelow` (the default) or `NSSliderCell.TickMarkAbove`; for vertical sliders, *position* can be `NSSliderCell.TickMarkLeft` (the default) or `NSSliderCell.TickMarkRight`. This method has no effect if no tick marks have been assigned (that is, `numberOfTickMarks` (page 1334) returns 0). In its implementation of this method, the receiving NSSlider simply invokes the method of the same name of its NSSliderCell.

See Also

`tickMarkPosition` (page 1338)

setTitle

This method has been deprecated. Sets the title the receiver displays in the bar behind its knob.

```
public void setTitle(String barTitle)
```

Discussion

The knob may cover part or all of the title. If you want the title to be visible all the time, you're better off placing a label near the slider.

See Also

`title` (page 1338)

setTitleCell

This method has been deprecated. Sets the cell used to draw the receiver's title.

```
public void setTitleCell(NSCell titleCell)
```

Discussion

You only need to invoke this method if the default title cell, NSTextFieldCell, doesn't suit your needs—that is, you want to display the title in a manner that NSTextFieldCell doesn't permit. When you do choose to override the default, *titleCell* should be an instance of a subclass of NSTextFieldCell.

See Also

`titleCell` (page 1339)

setTitleColor

This method has been deprecated. Sets the color used to draw the receiver's title to *color*.

```
public void setTitleColor(NSColor color)
```

See Also
`titleColor` (page 1339)

setTitleFont

This method has been deprecated. Sets the font used to draw the receiver's title.

```
public void setTitleFont(NSFont font)
```

See Also
`titleFont` (page 1339)

tickMarkPosition

Returns how the receiver's tick marks are aligned with it.

```
public int tickMarkPosition()
```

Discussion
Possible values are `NSSliderCell.TickMarkBelow`, `NSSliderCell.TickMarkAbove`, `NSSliderCell.TickMarkLeft`, and `NSSliderCell.TickMarkRight` (the last two are for vertical sliders). The default alignments are `NSSliderCell.TickMarkBelow` and `NSSliderCell.TickMarkLeft`. In its implementation of this method, the receiving NSSlider simply invokes the method of the same name of its NSSliderCell.

See Also
`setTickMarkPosition` (page 1337)

tickMarkValueAtIndex

Returns the receiver's value represented by the tick mark at *index* (the minimum-value tick mark has an index of 0).

```
public double tickMarkValueAtIndex(int index)
```

Discussion
In its implementation of this method, the receiving NSSlider simply invokes the method of the same name of its NSSliderCell.

title

Returns the receiver's title.

```
public String title()
```

Discussion

The default title is the empty string (" ").

See Also

`setTitle` (page 1337)

titleCell

This method has been deprecated. Returns `null`.

```
public NSCell titleCell()
```

See Also

`setTitleCell` (page 1337)

titleColor

This method has been deprecated. Returns `null`.

```
public NSColor titleColor()
```

See Also

`setTitleColor` (page 1338)

titleFont

This method has been deprecated. Returns `null`.

```
public NSFont titleFont()
```

See Also

`setTitleFont` (page 1338)

NSSliderCell

Inherits from	NSActionCell : NSCell : NSObject
Implements	NSCoding (NSCell)
Package:	com.apple.cocoa.application
Companion guide	Slider Programming Topics for Cocoa

Overview

An NSSliderCell controls the appearance and behavior of an NSSlider (page 1329), or of a single slider in an NSMatrix.

You can customize an NSSliderCell to a certain degree, using its `set...` methods. If these methods do not allow you sufficient flexibility, you can create a subclass. In that subclass, you can override any of the following methods: `knobRectFlipped` (page 1346), `drawBarInside` (page 1345), `drawKnob` (page 1346), and `prefersTrackingUntilMouseUp` (page 1344).

Tasks

Constructors

`NSSliderCell` (page 1344)
Creates an empty NSSliderCell.

Asking About the Cell's Behavior

`altIncrementValue` (page 1345)
Returns the amount the slider will change its value when the user drags the knob with the Option key held down.

`prefersTrackingUntilMouseUp` (page 1344)
By default, this method returns `true`, so an NSSliderCell continues to track the cursor even after the cursor leaves the cell's tracking rectangle.

`trackRect` (page 1353)
Returns the rectangle within which the cell tracks the cursor while the mouse button is down.

Setting the Slider Type

`setSliderType` (page 1350)
: Sets the type of slider to a bar or a dial.

`sliderType` (page 1351)
: Returns the slider type; either a bar or a dial.

Changing the Cell's Behavior

`setAltIncrementValue` (page 1348)
: Sets the amount by which the receiver modifies its value when the knob is Option-dragged.

Displaying the Cell

`knobRectFlipped` (page 1346)
: Returns the rectangle in which the knob will be drawn, specified in the coordinate system of the NSSlider or NSMatrix with which the receiver is associated.

`drawBarInside` (page 1345)
: Draws the slider's bar—but not its bezel or knob—inside *aRect*.

`drawKnob` (page 1346)
: Calculates the rectangle in which the knob should be drawn, then invokes `drawKnobInRect` (page 1346) to actually draw the knob.

`drawKnobInRect` (page 1346)
: Draws the knob in *knobRect*.

Asking About the Cell's Appearance

`knobThickness` (page 1347)
: Returns the knob's thickness, in pixels.

`isVertical` (page 1346)
: Returns 1 if the slider is vertical, 0 if it's horizontal, and –1 if the orientation can't be determined (for example, if the slider hasn't been displayed yet).

`title` (page 1352)
: This method has been deprecated. Returns the slider's title.

`titleCell` (page 1352)
: This method has been deprecated. Returns `null`.

`titleFont` (page 1352)
: This method has been deprecated. Returns `null`.

`titleColor` (page 1352)
: This method has been deprecated. Returns `null`.

Changing the Cell's Appearance

`setKnobThickness` (page 1349)
This method has been deprecated. Lets you set the knob's thickness, measured in pixels.

`setTitle` (page 1350)
This method has been deprecated. Sets the title in the bar behind the slider's knob to *title*.

`setTitleCell` (page 1351)
This method has been deprecated. Sets the cell used to draw the slider's title.

`setTitleColor` (page 1351)
This method has been deprecated. Sets the color used to draw the slider's title.

`setTitleFont` (page 1351)
This method has been deprecated. Sets the font used to draw the slider's title.

Asking About the Value Limits

`maxValue` (page 1347)
Returns the maximum value the slider can send to its target.

`minValue` (page 1347)
Returns the minimum value the slider can send to its target.

Changing the Value Limits

`setMaxValue` (page 1349)
Sets the maximum value the slider can send to its target to *aDouble*.

`setMinValue` (page 1349)
Sets the minimum value the slider can send to its target to *aDouble*.

Managing Tick Marks

`allowsTickMarkValuesOnly` (page 1345)
Returns whether the receiver fixes its values to those values represented by its tick marks.

`closestTickMarkValueToValue` (page 1345)
Returns the value of the tick mark closest to *aValue*.

`indexOfTickMarkAtPoint` (page 1346)
Returns the index of the tick mark closest to the location of the slider represented by *point*.

`numberOfTickMarks` (page 1347)
Returns the number of tick marks associated with the slider.

`rectOfTickMarkAtIndex` (page 1348)
Returns the bounding rectangle of the tick mark identified by *index* (the minimum-value tick mark is at index 0).

`setAllowsTickMarkValuesOnly` (page 1348)
Sets whether the receiver's values are fixed to the values represented by the tick marks to *flag*.

`setNumberOfTickMarks` (page 1349)
: Sets the number of tick marks displayed by the receiver (which include those assigned to the minimum and maximum values) to *`numberOfTickMarks`*.

`setTickMarkPosition` (page 1350)
: Sets where tick marks appear relative to the receiver.

`tickMarkPosition` (page 1351)
: Returns how the receiver's tick marks are aligned with it.

`tickMarkValueAtIndex` (page 1352)
: Returns the receiver's value represented by the tick mark at *`index`* (the minimum-value tick mark has an index of 0).

Constructors

NSSliderCell

Creates an empty NSSliderCell.

```
public NSSliderCell()
```

Creates an NSSliderCell initialized with *`aString`*.

```
public NSSliderCell(String aString)
```

Creates an NSSliderCell initialized with *`anImage`*.

```
public NSSliderCell(NSImage anImage)
```

Discussion
If *`anImage`* is `null`, no image is set.

Static Methods

prefersTrackingUntilMouseUp

By default, this method returns `true`, so an NSSliderCell continues to track the cursor even after the cursor leaves the cell's tracking rectangle.

```
public static boolean prefersTrackingUntilMouseUp()
```

Discussion
This means that, once you take hold of a slider's knob (by putting the cursor inside the cell's frame rectangle and pressing the mouse button), you retain control of the knob until you release the mouse button, even if you drag the cursor clear to the other side of the screen.

Never call this method explicitly. Override it if you create a subclass of NSSliderCell that you want to track the mouse differently.

Instance Methods

allowsTickMarkValuesOnly

Returns whether the receiver fixes its values to those values represented by its tick marks.

```
public boolean allowsTickMarkValuesOnly()
```

See Also
`setAllowsTickMarkValuesOnly` (page 1348)

altIncrementValue

Returns the amount the slider will change its value when the user drags the knob with the Option key held down.

```
public double altIncrementValue()
```

Discussion
Unless you call `setAltIncrementValue` (page 1348), `altIncrementValue` (page 1345) returns –1.0, and the slider behaves no differently with the Option key down than with it up.

See Also
`setAltIncrementValue` (page 1348)

closestTickMarkValueToValue

Returns the value of the tick mark closest to *aValue*.

```
public double closestTickMarkValueToValue(double aValue)
```

See Also
`indexOfTickMarkAtPoint` (page 1346)

drawBarInside

Draws the slider's bar—but not its bezel or knob—inside *aRect*.

```
public void drawBarInside(NSRect aRect, boolean flipped)
```

Discussion
aRect is the bounds of the bar, not its interior rect.

The *flipped* argument indicates whether the cell's control view—that is, the NSSlider or NSMatrix associated with the NSSliderCell—has a flipped coordinate system.

You should never invoke this method explicitly. It's included so you can override it in a subclass.

See Also
`drawKnobInRect` (page 1346)

drawKnob

Calculates the rectangle in which the knob should be drawn, then invokes drawKnobInRect (page 1346) to actually draw the knob.

```
public void drawKnob()
```

Discussion
Before this message is sent, a lockFocus method must be sent to the cell's control view.

You might invoke this method if you override one of the display methods belonging to NSControl or NSCell.

If you create a subclass of NSSliderCell, don't override this method. Override drawKnobInRect (page 1346) instead.

drawKnobInRect

Draws the knob in *knobRect*.

```
public void drawKnobInRect(NSRect knobRect)
```

Discussion
Before this message is sent, a lockFocus (page 1759) message must be sent to the cell's control view.

You should never invoke this method explicitly. It's included so you can override it in a subclass.

indexOfTickMarkAtPoint

Returns the index of the tick mark closest to the location of the slider represented by *point*.

```
public int indexOfTickMarkAtPoint(NSPoint point)
```

Discussion
If *point* is not within the bounding rectangle (plus an extra pixel of space) of any tick mark, the method returns NSArray.NotFound. This method invokes rectOfTickMarkAtIndex (page 1348) for each tick mark on the slider until it finds a tick mark containing *point*.

isVertical

Returns 1 if the slider is vertical, 0 if it's horizontal, and –1 if the orientation can't be determined (for example, if the slider hasn't been displayed yet).

```
public int isVertical()
```

Discussion
A slider is defined as vertical if its height is greater than its width.

knobRectFlipped

Returns the rectangle in which the knob will be drawn, specified in the coordinate system of the NSSlider or NSMatrix with which the receiver is associated.

```
public NSRect knobRectFlipped(boolean flipped)
```

Discussion

flipped indicates whether that coordinate system is flipped, a question you can answer by sending NSView's `isFlipped` (page 1756) message to the NSMatrix or NSSlider.

The knob rectangle depends on where in the slider the knob belongs—that is, it depends on the receiver's minimum and maximum values and on the value the position of the knob will represent.

You should never invoke this method explicitly. It's included so you can override it in a subclass.

knobThickness

Returns the knob's thickness, in pixels.

```
public float knobThickness()
```

Discussion

The thickness is defined to be the extent of the knob along the long dimension of the bar. In a vertical slider, then, a knob's thickness is its height; in a horizontal slider, its thickness is its width.

See Also

`setKnobThickness` (page 1349)

maxValue

Returns the maximum value the slider can send to its target.

```
public double maxValue()
```

Discussion

A horizontal slider sends its maximum value when the knob is at the right end of the slider; a vertical slider sends it when the knob is at the top. The maximum selectable value for a circular slider is just below maxValue; for example, if maxValue is 360, you can set the dial up to 359.999.

See Also

`setMaxValue` (page 1349)

minValue

Returns the minimum value the slider can send to its target.

```
public double minValue()
```

Discussion

A vertical slider sends this value when its knob is at the bottom; a horizontal slider sends it when its knob is all the way to the left; a circular slider sends it when its knob is at the top.

numberOfTickMarks

Returns the number of tick marks associated with the slider.

```
public int numberOfTickMarks()
```

Discussion
The tick marks assigned to the minimum and maximum values are included.

See Also
`setNumberOfTickMarks` (page 1349)

rectOfTickMarkAtIndex

Returns the bounding rectangle of the tick mark identified by *index* (the minimum-value tick mark is at index 0).

```
public NSRect rectOfTickMarkAtIndex(int index)
```

Discussion
If no tick mark is associated with *index*, the method throws `RangeException`.

See Also
`indexOfTickMarkAtPoint` (page 1346)

setAllowsTickMarkValuesOnly

Sets whether the receiver's values are fixed to the values represented by the tick marks to *flag*.

```
public void setAllowsTickMarkValuesOnly(boolean flag)
```

Discussion
For example, if a slider has a minimum value of 0, a maximum value of 100, and five markers, the allowable values are 0, 25, 50, 75, and 100. When users move the slider's knob, it jumps to the tick mark nearest the cursor when the mouse button is released. This method has no effect if the slider has no tick marks.

See Also
`allowsTickMarkValuesOnly` (page 1345)

setAltIncrementValue

Sets the amount by which the receiver modifies its value when the knob is Option-dragged.

```
public void setAltIncrementValue(double increment)
```

Discussion
increment should fit the range of values the slider can represent—for example, if the slider has a minimum value of 5 and a maximum value of 10, *increment* should be between 0 and 5.

If you don't call this method, the slider behaves the same with the Option key down as with it up. This is also the result when you call `setAltIncrementValue` (page 1348) with an increment of –1.

See Also
`maxValue` (page 1347)
`minValue` (page 1347)

setKnobThickness

This method has been deprecated. Lets you set the knob's thickness, measured in pixels.

```
public void setKnobThickness(float thickness)
```

Discussion

The thickness is defined to be the extent of the knob along the long dimension of the bar. In a vertical slider, then, a knob's thickness is its height; in a horizontal slider, its thickness is its width.

See Also

`knobThickness` (page 1347)

setMaxValue

Sets the maximum value the slider can send to its target to *aDouble*.

```
public void setMaxValue(double aDouble)
```

Discussion

A horizontal slider sends its maximum value when its knob is all the way to the right; a vertical slider sends its maximum value when its knob is at the top. The maximum selectable value for a circular slider is just below maxValue; for example, if maxValue is 360, you can set the dial up to 359.999.

See Also

`maxValue` (page 1347)

setMinValue

Sets the minimum value the slider can send to its target to *aDouble*.

```
public void setMinValue(double aDouble)
```

Discussion

A horizontal slider sends its minimum value when its knob is all the way to the left; a vertical slider sends its minimum value when its knob is at the bottom; a circular slider sends it when its knob is at the top.

See Also

`minValue` (page 1347)

setNumberOfTickMarks

Sets the number of tick marks displayed by the receiver (which include those assigned to the minimum and maximum values) to *numberOfTickMarks*.

```
public void setNumberOfTickMarks(int numberOfTickMarks)
```

Discussion

By default, this value is 0, and no tick marks appear. The number of tick marks assigned to a slider, along with the slider's minimum and maximum values, determines the values associated with the tick marks.

See Also

`numberOfTickMarks` (page 1347)

setSliderType

Sets the type of slider to a bar or a dial.

```
public void setSliderType(int sliderType)
```

Discussion

Possible values for *sliderType* are described in "Constants" (page 1353). If you select `CircularSlider`, then you get a fixed-size round slider. The minimum value (minValue) is at the top, and the value increases as you go clockwise around the dial. The maximum selectable value is just below maxValue; for example, if maxValue is 360, you can set the dial up to 359.999. You can use the `setNumberOfTickMarks` (page 1349) method to display tick marks, and you can use the `setAllowsTickMarkValuesOnly` (page 1348) method to specify that values are limited to those values represented by tick marks. You can set this control to regular or small sizes; the mini size is not supported.

Availability

Available in Mac OS X v10.3 and later.

See Also

`sliderType` (page 1351)

`setNumberOfTickMarks` (page 1349)

`setAllowsTickMarkValuesOnly` (page 1348)

setTickMarkPosition

Sets where tick marks appear relative to the receiver.

```
public void setTickMarkPosition(int position)
```

Discussion

Possible values for *position* are described in "Constants" (page 1353). This method has no effect if no tick marks have been assigned (that is, `numberOfTickMarks` (page 1347) returns 0).

See Also

`tickMarkPosition` (page 1351)

setTitle

This method has been deprecated. Sets the title in the bar behind the slider's knob to *title*.

```
public void setTitle(String title)
```

See Also

`title` (page 1352)

setTitleCell

This method has been deprecated. Sets the cell used to draw the slider's title.

```
public void setTitleCell(NSCell aCell)
```

Discussion
You only need to invoke this method if the default title cell, NSTextFieldCell, doesn't suit your needs—that is, if you want to display the title in a manner that NSTextFieldCell doesn't permit. When you do choose to override the default, `aCell` should be an instance of a subclass of NSTextFieldCell.

See Also
`titleCell` (page 1352)

setTitleColor

This method has been deprecated. Sets the color used to draw the slider's title.

```
public void setTitleColor(NSColor color)
```

See Also
`titleColor` (page 1352)

setTitleFont

This method has been deprecated. Sets the font used to draw the slider's title.

```
public void setTitleFont(NSFont font)
```

See Also
`titleFont` (page 1352)

sliderType

Returns the slider type; either a bar or a dial.

```
public int sliderType()
```

Discussion
Possible return values are described in "Constants" (page 1353).

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

See Also
`setSliderType` (page 1350)

tickMarkPosition

Returns how the receiver's tick marks are aligned with it.

```
public int tickMarkPosition()
```

Discussion
Possible return values are described in "Constants" (page 1353). The default alignments are `TickMarkBelow` and `TickMarkLeft`.

See Also
`setTickMarkPosition` (page 1350)

tickMarkValueAtIndex

Returns the receiver's value represented by the tick mark at *`index`* (the minimum-value tick mark has an index of 0).

```
public double tickMarkValueAtIndex(int index)
```

title

This method has been deprecated. Returns the slider's title.

```
public String title()
```

Discussion
The default title is the empty string (" ").

See Also
`setTitle` (page 1350)

titleCell

This method has been deprecated. Returns `null`.

```
public NSCell titleCell()
```

See Also
`setTitleCell` (page 1351)

titleColor

This method has been deprecated. Returns `null`.

```
public NSColor titleColor()
```

See Also
`setTitleColor` (page 1351)

titleFont

This method has been deprecated. Returns `null`.

```
public NSFont titleFont()
```

See Also
setTitleFont (page 1351)

trackRect

Returns the rectangle within which the cell tracks the cursor while the mouse button is down.

```
public NSRect trackRect()
```

Discussion
This rectangle includes the slider bar, but not the bezel.

Constants

NSSliderCell defines the following constants to specify where the tick marks appear; they are used in setTickMarkPosition (page 1350) and tickMarkPosition (page 1351):

Constant	Description
TickMarkBelow	Tick marks below (for horizontal sliders); the default for horizontal sliders.
TickMarkAbove	Tick marks above (for horizontal sliders).
TickMarkLeft	Tick marks to the left (for vertical sliders); the default. for vertical sliders
TickMarkRight	Tick marks to the right (for vertical sliders).

NSSliderCell defines the following slider types, used by setSliderType (page 1350) and sliderType (page 1351):

Constant	Description
LinearSlider	A bar-shaped slider.
CircularSlider	A circular slider; that is, a dial.

NSSound

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.application
Companion guide	Sound Programming Topics for Cocoa

Overview

The NSSound class provides a simple interface for loading and playing AIFF, WAV, and NeXT “.snd” files. NSSound supports 16-bit, mono and stereo, 44.1KHz and 22.05KHz data.

Tasks

Constructors

NSSound (page 1356)
Creates an empty NSSound.

Playing

isPlaying (page 1358)
Returns true if the receiver is currently playing its audio data; otherwise it returns false.

pause (page 1358)
Pauses audio playback. Returns true if successful, otherwise false.

play (page 1359)
Initiates audio playback. Returns true if successful, otherwise false.

resume (page 1359)
Resumes audio playback; assumes the receiver has been previously paused by sending it pause (page 1358). Returns true if successful, otherwise false.

stop (page 1359)
Halts audio playback. Returns true if successful, otherwise false.

Working with Pasteboards

canInitWithPasteboard (page 1357)
Tests whether the receiver can create an instance of itself from the data represented by *pasteboard*. Returns true if the receiver can handle the data represented by *pasteboard*. soundUnfilteredPasteboardTypes (page 1358) is used to find out if the class can handle the data in *pasteboard*.

soundUnfilteredPasteboardTypes (page 1358)
Returns an array of pasteboard types that NSSound can accept.

writeToPasteboard (page 1359)
Writes the receiver's data to *pasteboard*.

Working with Delegates

delegate (page 1358)
Returns the receiver's delegate.

setDelegate (page 1359)
Set the receiver's delegate to be *aDelegate*.

Naming Sounds

soundNamed (page 1357)
Returns the NSSound instance associated with *name*.

soundUnfilteredFileTypes (page 1358)
Returns an array of strings representing those file types that NSSound understands. The array returned by this method may be passed directly to NSOpenPanel's runModalForTypes (page 1024).

name (page 1358)
Returns the name assigned to the receiver, or null if no name has been assigned.

setName (page 1359)
Registers the receiver under the name specified by *string*, provided that no other NSSound is registered using that name. If the receiver is already registered under another name, setName first unregisters the prior name. setName returns true unless another NSSound is registered using the name specified by *string*, in which case setName does nothing and returns false.

Constructors

NSSound

Creates an empty NSSound.

```
public NSSound()
```

Creates an NSSound instance using the data on *aPasteboard*. NSSound expects the data to have a proper magic number, sound header, and data for the formats it supports.

```
public NSSound(NSPasteboard aPasteboard)
```

Creates an NSSound instance with the contents of the file at *aString*. If *byRef* is `true`, only the name of the sound is stored with the NSSound instance when archived; otherwise the audio data is archived along with the instance.

```
public NSSound(String aString, boolean byRef)
```

Creates an `NSSound` instance with the audio data at *aURL*. If *byRef* is `true`, only the name of the sound is stored with the `NSSound` instance when archived; otherwise the audio data is archived along with the instance.

```
public NSSound(java.net.URL aURL, boolean flag)
```

Creates an `NSSound` instance and initializes it with the contents of *data*.

```
public NSSound(NSData data)
```

Discussion
`NSSound` expects the data to have a proper magic number, sound header, and data for the formats it supports.

Static Methods

canInitWithPasteboard

Tests whether the receiver can create an instance of itself from the data represented by *pasteboard*. Returns `true` if the receiver can handle the data represented by *pasteboard*. `soundUnfilteredPasteboardTypes` (page 1358) is used to find out if the class can handle the data in *pasteboard*.

```
public static boolean canInitWithPasteboard(NSPasteboard pasteboard)
```

soundNamed

Returns the NSSound instance associated with *name*.

```
public static NSSound soundNamed(String name)
```

Discussion
The returned object can be one of the following:

- One that's been assigned a name with `setName` (page 1359)
- One of the named system sounds provided by the Application Kit

If there's no known `NSSound` with *name*, this method tries to create one by searching for sound files in the application's main bundle (see NSBundle for a description of how the bundle's contents are searched). If no sound file can be located in the application main bundle, the `/Library/Sounds` and `~/Library/Sounds` directories are searched. If no data can be found for *name*, no object is created, and `null` is returned.

The preferred way to locate a sound is to pass a name without the file extension.

Note: AIFF files stored on disk must use the `.aiff` file extension (not `.aif`) in order to be located by `soundNamed`.

soundUnfilteredFileTypes

Returns an array of strings representing those file types that NSSound understands. The array returned by this method may be passed directly to NSOpenPanel's `runModalForTypes` (page 1024).

```
public static NSArray soundUnfilteredFileTypes()
```

soundUnfilteredPasteboardTypes

Returns an array of pasteboard types that NSSound can accept.

```
public static NSArray soundUnfilteredPasteboardTypes()
```

Instance Methods

delegate

Returns the receiver's delegate.

```
public Object delegate()
```

See Also
`setDelegate` (page 1359)

isPlaying

Returns `true` if the receiver is currently playing its audio data; otherwise it returns `false`.

```
public boolean isPlaying()
```

name

Returns the name assigned to the receiver, or `null` if no name has been assigned.

```
public String name()
```

See Also
`setName` (page 1359)

pause

Pauses audio playback. Returns `true` if successful, otherwise `false`.

```
public boolean pause()
```

play

Initiates audio playback. Returns `true` if successful, otherwise `false`.

```
public boolean play()
```

resume

Resumes audio playback; assumes the receiver has been previously paused by sending it `pause` (page 1358). Returns `true` if successful, otherwise `false`.

```
public boolean resume()
```

setDelegate

Set the receiver's delegate to be *`aDelegate`*.

```
public void setDelegate(Object aDelegate)
```

See Also

`delegate` (page 1358)

setName

Registers the receiver under the name specified by *`string`*, provided that no other NSSound is registered using that name. If the receiver is already registered under another name, `setName` first unregisters the prior name. `setName` returns `true` unless another NSSound is registered using the name specified by *`string`*, in which case `setName` does nothing and returns `false`.

```
public void setName(String string)
```

See Also

`name` (page 1358)

`soundNamed` (page 1357)

stop

Halts audio playback. Returns `true` if successful, otherwise `false`.

```
public boolean stop()
```

writeToPasteboard

Writes the receiver's data to *`pasteboard`*.

```
public void writeToPasteboard(NSPasteboard pasteboard)
```

NSSpeechRecognizer

Inherits from	NSObject
Package:	com.apple.cocoa.application
Availability	Available in Mac OS X v10.3 and later.
Companion guide	Speech

Overview

The NSSpeechRecognizer class is the Cocoa interface to Speech Recognition on Mac OS X. Speech Recognition is architected as a “command and control” voice recognition system. It uses a finite state grammar and listens for phrases in that grammar. When it recognizes a phrase, it notifies the client process. This architecture is different from that used to support dictation.

Through an NSSpeechRecognizer instance, Cocoa applications can use the speech recognition engine built into Mac OS X to recognize spoken commands. With speech recognition, users can accomplish complex, multi-step tasks with one spoken command—for example, “schedule a meeting with Adam and John tomorrow at ten o’clock.”

The NSSpeechRecognizer class has methods that let you specify which spoken words should be recognized as commands (`setCommands` (page 1365)) and to start and stop listening (`startListening` (page 1366) and `stopListening` (page 1366)). When the Speech Recognition facility recognizes one of the designated commands, NSSpeechRecognizer invokes the delegation method `speechRecognizerDidRecognizeCommand` (page 1367), allowing the delegate to perform the command.

Speech Recognition is just one of the Mac OS X speech technologies. The Speech Synthesis technology allows applications to “pronounce” written text in U.S. English; the NSSpeechSynthesizer class is the Cocoa interface to this technology. These technologies provide benefits for all users, and are particularly useful to those users who have difficulties seeing the screen or using the mouse and keyboard. By incorporating speech into your application, you can provide a concurrent mode of interaction for your users: In Mac OS X, your software can accept input and provide output without requiring users to change their working context.

Tasks

Constructors

`NSSpeechRecognizer` (page 1363)
Creates and returns an NSSpeechRecognizer.

Listening

`startListening` (page 1366)
Tells the speech recognition engine to begin listening for commands.

`stopListening` (page 1366)
Tells the speech recognition engine to suspend listening for commands.

Managing Delegates

`setDelegate` (page 1365)
Sets the receiver's delegate.

`delegate` (page 1363)
Returns the receiver's delegate.

Managing Recognizer Attributes

`setCommands` (page 1365)
Sets the list of commands for which the receiver should listen to *`commands`*.

`commands` (page 1363)
Returns an array of strings defining the commands for which the receiver should listen.

`setDisplayedCommandsTitle` (page 1365)
Sets whether the speech-recognition commands should be displayed indented under a section title in the Speech Commands window, and if so, sets the title string to display.

`displayedCommandsTitle` (page 1364)
Returns the title of the commands section or `null` if there is no title.

`setListensInForegroundOnly` (page 1366)
Sets whether the receiver should only enable its commands when the receiver's application is the frontmost one.

`listensInForegroundOnly` (page 1364)
Returns whether the receiver should only enable its commands when the receiver's application is the frontmost one.

`setBlocksOtherRecognizers` (page 1364)
Sets whether the receiver's commands should be the only enabled commands on the system.

`blocksOtherRecognizers` (page 1363)
Returns whether the receiver should block all other recognizers (that is, other applications attempting to understand spoken commands) when listening.

Recognizing speech

`speechRecognizerDidRecognizeCommand` (page 1367) *delegate method*
Invoked when the recognition engine has recognized the application command *`command`*.

Constructors

NSSpeechRecognizer

Creates and returns an NSSpeechRecognizer.

```
public NSSpeechRecognizer()
```

Discussion
Returns `null` if creation did not succeed.

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

Instance Methods

blocksOtherRecognizers

Returns whether the receiver should block all other recognizers (that is, other applications attempting to understand spoken commands) when listening.

```
public boolean blocksOtherRecognizers()
```

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

See Also
`setBlocksOtherRecognizers` (page 1364)

commands

Returns an array of strings defining the commands for which the receiver should listen.

```
public NSArray commands()
```

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

See Also
`setCommands` (page 1365)

delegate

Returns the receiver's delegate.

```
public Object delegate()
```

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

See Also
`setDelegate` (page 1365)

displayedCommandsTitle

Returns the title of the commands section or `null` if there is no title.

```
public String displayedCommandsTitle()
```

Discussion
Commands are displayed in the Speech Commands window indented under a section with this title.

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

See Also
`setDisplayedCommandsTitle` (page 1365)

listensInForegroundOnly

Returns whether the receiver should only enable its commands when the receiver's application is the frontmost one.

```
public boolean listensInForegroundOnly()
```

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

See Also
`setListensInForegroundOnly` (page 1366)

setBlocksOtherRecognizers

Sets whether the receiver's commands should be the only enabled commands on the system.

```
public void setBlocksOtherRecognizers(boolean flag)
```

Discussion
If *`flag`* is `true`, all other speech recognition commands on the system are disabled until the receiver object is released, listening is stopped, or this method is called again with *`flag`* as `false`. Because this option effectively takes over the computer at the expense of other applications using speech recognition, you should use it only in circumstances that warrant it, such as when listening for a response important to overall system operation or when an application is running in full-screen mode (such as games and presentation software). The default is `false`.

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

See Also

`blocksOtherRecognizers` (page 1363)

setCommands

Sets the list of commands for which the receiver should listen to *commands*.

```
public void setCommands(NSArray commands)
```

Discussion

If the receiver is already listening, the current command list is updated and listening continues. *commands* should be an array of String objects. The commands must be in U.S. English.

Availability

Available in Mac OS X v10.3 and later.

See Also

`commands` (page 1363)

setDelegate

Sets the receiver's delegate.

```
public void setDelegate(Object anObject)
```

Availability

Available in Mac OS X v10.3 and later.

See Also

`delegate` (page 1363)

setDisplayedCommandsTitle

Sets whether the speech-recognition commands should be displayed indented under a section title in the Speech Commands window, and if so, sets the title string to display.

```
public void setDisplayedCommandsTitle(String title)
```

Discussion

When *title* is a non-empty string, the receiver's commands are displayed under a section with *title*. If *title* is `null` or an empty string, the commands are displayed at the top level of the Speech Commands window. This default is not to display the commands under a section title.

Availability

Available in Mac OS X v10.3 and later.

See Also

`displayedCommandsTitle` (page 1364)

setListensInForegroundOnly

Sets whether the receiver should only enable its commands when the receiver's application is the frontmost one.

```
public void setListensInForegroundOnly(boolean flag)
```

Discussion

If *flag* is true, the receiver's commands are only recognized when the receiver's application is the frontmost application—normally the application displaying the menu bar. If *flag* is false, the commands are recognized regardless of the visibility of applications, including agent applications (agent applications, which have the LSUIElement property set, do not appear in the Dock or Force Quit window). The default is true.

Availability

Available in Mac OS X v10.3 and later.

See Also

listensInForegroundOnly (page 1364)

startListening

Tells the speech recognition engine to begin listening for commands.

```
public void startListening()
```

Discussion

When a command is recognized the message speechRecognizerDidRecognizeCommand (page 1367) is sent to the delegate.

Availability

Available in Mac OS X v10.3 and later.

See Also

stopListening (page 1366)

stopListening

Tells the speech recognition engine to suspend listening for commands.

```
public void stopListening()
```

Availability

Available in Mac OS X v10.3 and later.

See Also

startListening (page 1366)

Delegate Methods

speechRecognizerDidRecognizeCommand

Invoked when the recognition engine has recognized the application command *command*.

```
public abstract void speechRecognizerDidRecognizeCommand(NSSpeechRecognizer sender,
     Object command)
```

Discussion

command is one of the strings from the array passed to `setCommands` (page 1365). The delegate typically evaluates which command was recognized and performs the related action.

Availability

Available in Mac OS X v10.3 and later.

NSSpeechSynthesizer

Inherits from	NSObject
Package:	com.apple.cocoa.application
Availability	Available in Mac OS X v10.3 and later.
Companion guide	Speech

Overview

The NSSpeechSynthesizer class is the Cocoa interface to Speech Synthesis on Mac OS X.

Speech Synthesis, also called text-to-speech (TTS), parses text and converts it into audible speech. It offers a concurrent feedback mode that can be used in concert with or in place of traditional visual and aural notifications. For example, your application can use an NSSpeechSynthesizer object to "pronounce" the text of important alert dialogs. Synthesized speech has several advantages. It can provide urgent information to users without forcing them to shift attention from their current task. And because speech doesn't rely on visual elements for meaning, it is a crucial technology for users with vision or attention disabilities.

In addition, synthesized speech can help save system resources. Because sound samples can take up large amounts of room on disk, using text in place of sampled sound is extremely efficient, and so a multimedia application might use an NSSpeechSynthesizer object to provide a narration of a QuickTime movie instead of including sampled-sound data on a movie track.

When you create an NSSpeechSynthesizer, the class uses the default voice selected in Speech preferences pane of System Preferences. Alternatively, you can select a specific voice for an NSSpeechSynthesis instance . To begin speaking, send `startSpeakingString` (page 1375) to the instance. The message generates speech through the computer's default sound output deviceor saves the generated speech to a file. If you wish to be notified when the current speech concludes, set a delegate object using `setDelegate` (page 1373) and implement the delegate method `speechSynthesizerDidFinishSpeaking` (page 1377).

Speech Synthesis is just one of the Mac OS X speech technologies. The Speech Recognizer technology allows applications to "listen to" text spoken in U.S. English; the NSSpeechRecognizer class is the Cocoa interface to this technology. Both technologies provide benefits for all users, and are particularly useful to those users who have difficulties seeing the screen or using the mouse and keyboard.

Tasks

Constructors

`NSSpeechSynthesizer` (page 1371)
Creates and returns and NSSpeechSynthesizer with the default voice.

Testing for Speaking

`isAnyApplicationSpeaking` (page 1373)
Returns whether any other application is currently synthesizing speech.

Obtaining Voice Information

`availableVoices` (page 1372)
Returns an array of identifiers for each available voice on the system.

`attributesForVoice` (page 1372)
Returns a dictionary object whose elements describe the attributes of the voice specified by *`voice`* (a `VoiceIdentifier` attribute).

`defaultVoice` (page 1372)
Returns the `VoiceIdentifier` string identifying the current default voice.

Speaking

`isSpeaking` (page 1373)
Returns whether the receiver is currently generating synthesized speech.

`startSpeakingString` (page 1375)
Begins synthesizing the text in *`string`* through the computer's default sound output device.

`stopSpeaking` (page 1376)
If the receiver is currently generating speech, synthesis is halted, and the message `speechSynthesizerDidFinishSpeaking` (page 1377) is sent to the delegate.

Managing Delegates

`setDelegate` (page 1373)
Sets the receiver’s delegate.

`delegate` (page 1373)
Returns the receiver’s delegate.

Managing Synthesizer Attributes

`setUsesFeedbackWindow` (page 1374)
Sets whether the receiver displays the text and appropriate animation in the speech-feedback window while the user is speaking.

`usesFeedbackWindow` (page 1376)
Returns whether spoken text should be displayed in the speech-feedback window when that window is visible.

`setVoice` (page 1374)
Set the current voice of the receiver to the voice specified by the identifier *`voice`*, a `VoiceIdentifier` attribute.

`voice` (page 1376)
Returns the string identifier for the receiver's current voice (`VoiceIdentfier`).

Speaking

`speechSynthesizerWillSpeakWord` (page 1378) *delegate method*
Invoked just before the synthesized word, defined by the character range *`characterRange`* in *`string`*, is sent to the default sound output device (or to the `AIFF` file if saving to file)

`speechSynthesizerWillSpeakPhoneme` (page 1377) *delegate method*
Invoked just before the synthesized phoneme *`phonemeOpcode`* is sent to the default sound output device (or `AIFF` file if saving to file).

`speechSynthesizerDidFinishSpeaking` (page 1377) *delegate method*
Invoked when an NSSpeechSynthesizer object has finished speaking.

Constructors

NSSpeechSynthesizer

Creates and returns and NSSpeechSynthesizer with the default voice.

```
public NSSpeechSynthesizer()
```

Discussion
Users can change the default voice in the Speech pane of System Preferences. If there is an error, creation fails and the constructor returns `null`.

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

Creates and returns and NSSpeechSynthesizer with the voice having the identifier string *`voice`*, a `VoiceIdentifier` attribute.

```
public NSSpeechSynthesizer(String voice)
```

Discussion
If *voice* is `null`, this constructor uses the default voice. Users can change the default voice in the Speech pane of System Preferences. If *voice* is not an available voice or if there is an error, creation fails and the constructor returns `null`.

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

Static Methods

attributesForVoice

Returns a dictionary object whose elements describe the attributes of the voice specified by *voice* (a `VoiceIdentifier` attribute).

```
public static NSDictionary attributesForVoice(String voice)
```

Discussion
Dictionary keys and values are described in "Constants" (page 1376).

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

availableVoices

Returns an array of identifiers for each available voice on the system.

```
public static NSArray availableVoices()
```

Discussion
The identifier objects are `VoiceIdentifier` attributes (see "Constants" (page 1376)). You can use the voice-identifier string to find out about the attributes of the voice and to specify the voice to use for synthesis.

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

See Also
`attributesForVoice` (page 1372)
`setVoice` (page 1374)

defaultVoice

Returns the `VoiceIdentifier` string identifying the current default voice.

```
public static String defaultVoice()
```

Discussion
The user can change the default voice in the Speech pane in System Preferences.

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

isAnyApplicationSpeaking

Returns whether any other application is currently synthesizing speech.

```
public static boolean isAnyApplicationSpeaking()
```

Discussion
You usually invoke this method to prevent your application from speaking over speech being generated by another application or system component.

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

Instance Methods

delegate

Returns the receiver's delegate.

```
public Object delegate()
```

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

See Also
`setDelegate` (page 1373)

isSpeaking

Returns whether the receiver is currently generating synthesized speech.

```
public boolean isSpeaking()
```

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

See Also
`startSpeakingString` (page 1375)
`stopSpeaking` (page 1376)

setDelegate

Sets the receiver's delegate.

```
public void setDelegate(Object anObject)
```

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

See Also
`delegate` (page 1373)

setUsesFeedbackWindow

Sets whether the receiver displays the text and appropriate animation in the speech-feedback window while the user is speaking.

```
public void setUsesFeedbackWindow(boolean flag)
```

Discussion
If *`flag`* is `true`, the receiver uses the speech-feedback window if it is visible to the user when he or she begins speaking. If flag is `false`, the speech-feedback window is not used. The default is `true`.

Using the speech-feedback window to display the synthesized text allows for integration with the NSSpeechRecognizer class to create interactive spoken dialogues. For example, your application might use an NSSpeechRecognizer object to listen for the command “Play some music.” When it recognizes this command, your application might then respond by speaking “Which artist?” using an NSSpeechSynthesizer object. Then your application can listen for a list of artist names, and so on until the user is listening to the desired song. Having the synthesized text displayed in the speech-feedback window makes the spoken exchange more natural and helps the user understand the synthesized speech.

> **Note:** The delegate does not receive `speechSynthesizerWillSpeakWord` (page 1378) and `speechSynthesizerWillSpeakPhoneme` (page 1377) messages when speaking occurs through the feedback window.

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

See Also
`usesFeedbackWindow` (page 1376)

setVoice

Set the current voice of the receiver to the voice specified by the identifier *`voice`*, a `VoiceIdentifier` attribute.

```
public boolean setVoice(String voice)
```

Discussion
If *`voice`* is `null`, receiver uses the default voice. The method returns `true` if the voice was successfully set; otherwise, it returns `false`.

Note: An attempt to set the voice of a receiver that is currently synthesizing speech always fails and returns `false`.

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

See Also
`voice` (page 1376)
`defaultVoice` (page 1372)

startSpeakingString

Begins synthesizing the text in *string* through the computer's default sound output device.

```
public boolean startSpeakingString(String string)
```

Discussion
Returns `true` if synthesis successfully starts; otherwise, it returns `false` (as when *string* is `null` or an empty string). When synthesis of *string* finishes normally or is stopped, the message `speechSynthesizerDidFinishSpeaking` (page 1377) is sent to the delegate.

Note: If the receiver is currently synthesizing speech when `startSpeakingString` is called, the currently generated speech is stopped and synthesis of the text in *string* begins.

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

Begins synthesizing the text in *string*, but instead of speaking the text through the computer's default sound output device, the receiver saves the sound data in an `AIFF` file at *url*, which identifies a local or remote file-system location.

```
public boolean startSpeakingString(String string, java.net.URL url)
```

Discussion
Returns `true` if synthesis successfully starts; otherwise, it returns `false` (as when *string* is `null` or an empty string). When synthesis of string finishes normally or is stopped, the message `speechSynthesizerDidFinishSpeaking` (page 1377) is sent to the delegate. One example of how you might use this method is in an email program that automatically converts new messages into sound files that can be stored on an iPod for later listening.

Note: The delegate does not receive `speechSynthesizerWillSpeakWord` (page 1378) and `speechSynthesizerWillSpeakPhoneme` (page 1377) messages when text is being synthesized to a file.

See Also
`isSpeaking` (page 1373)
`stopSpeaking` (page 1376)

stopSpeaking

If the receiver is currently generating speech, synthesis is halted, and the message `speechSynthesizerDidFinishSpeaking` (page 1377) is sent to the delegate.

```
public void stopSpeaking()
```

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

See Also
`isSpeaking` (page 1373)
`startSpeakingString` (page 1375)

usesFeedbackWindow

Returns whether spoken text should be displayed in the speech-feedback window when that window is visible.

```
public boolean usesFeedbackWindow()
```

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

See Also
`setUsesFeedbackWindow` (page 1374)

voice

Returns the string identifier for the receiver's current voice (`VoiceIdentfier`).

```
public String voice()
```

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

See Also
`setVoice` (page 1374)

Constants

The following voice attributes are defined by the NSSpeechSynthesizer class for the dictionary returned by `attributesForVoice` (page 1372) (the values are all String objects):

Constant	Description
`VoiceName`	The name of the voice suitable for display
`VoiceIdentifier`	A unique string identifying the voice

Constant	Description
`VoiceAge`	The perceived age (in years) of the voice
`VoiceGender`	The perceived gender of the voice. May be either `VoiceGenderNeuter`, `VoiceGenderFemale`, or `VoiceGenderMale`
`VoiceDemoText`	A demonstration string to speak
`VoiceLanguage`	The language of the voice (currently US English only)

NSSpeechSynthesizer defines the following voice gender attributes, which are the allowable values of the `VoiceGender` key:

Constant	Description
`VoiceGenderNeuter`	A neutral voice (neither male or female)
`VoiceGenderMale`	A male voice
`VoiceGenderFemale`	A female voice

Delegate Methods

speechSynthesizerDidFinishSpeaking

Invoked when an NSSpeechSynthesizer object has finished speaking.

```
public abstract void speechSynthesizerDidFinishSpeaking(NSSpeechSynthesizer sender,
     boolean finishedSpeaking)
```

Discussion

finishedSpeaking is `true` if speaking completed normally; otherwise, it is `false` if speaking was prematurely stopped for any reason.

Availability

Available in Mac OS X v10.3 and later.

See Also

`startSpeakingString` (page 1375)

`stopSpeaking` (page 1376)

speechSynthesizerWillSpeakPhoneme

Invoked just before the synthesized phoneme *phonemeOpcode* is sent to the default sound output device (or `AIFF` file if saving to file).

```
public abstract void speechSynthesizerWillSpeakPhoneme(NSSpeechSynthesizer sender,
     short phonemeOpcode)
```

Discussion

One use of this method might be to animate a mouth on screen to match the generated speech.

Note: The delegate is not sent this message when the NSSpeechSynthesizer object is synthesizing speech to a file (`startSpeakingString` (page 1375) with two parameters).

Availability

Available in Mac OS X v10.3 and later.

See Also

`startSpeakingString` (page 1375)

speechSynthesizerWillSpeakWord

Invoked just before the synthesized word, defined by the character range *characterRange* in *string*, is sent to the default sound output device (or to the `AIFF` file if saving to file)

```
public abstract void speechSynthesizerWillSpeakWord(NSSpeechSynthesizer sender,
    NSRange characterRange, String string)
```

Discussion

. One use of this method might be to visually highlight the word being spoken.

Note: The delegate is not sent this message when the NSSpeechSynthesizer object is synthesizing speech to a file (`startSpeakingString` (page 1375) with two parameters).

Availability

Available in Mac OS X v10.3 and later.

See Also

`startSpeakingString` (page 1375)

NSSpellChecker

Inherits from	NSObject
Package:	com.apple.cocoa.application
Companion guide	Spell Checking

Overview

The NSSpellChecker class gives any application an interface to the Cocoa 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.

The spell checker also offers the ability to provide word completions to augment the text completion system in Mac OS X version 10.3.

Tasks

Constructors

`NSSpellChecker` (page 1381)
Creates an empty NSSpellChecker.

Getting the Spell Checker

`sharedSpellChecker` (page 1381)
Returns the NSSpellChecker (one per application).

`sharedSpellCheckerExists` (page 1381)
Returns whether the application's NSSpellChecker has already been created.

Managing the Spelling Panel

`setAccessoryView` (page 1383)
Makes an NSView object an accessory of the Spelling panel by making it a subview of the panel's content view.

`accessoryView` (page 1381)
Returns the Spelling panel's accessory NSView object.

`spellingPanel` (page 1384)
Returns the spell checker's panel.

Checking Spelling

`countWordsInString` (page 1382)
Returns the number of words in *stringToCount*.

`checkSpellingOfString` (page 1382)
Starts the search for a misspelled word in *stringToCheck* starting at *startingOffset* within the string object.

`guessesForWord` (page 1382)
Returns an array of suggested spellings for the misspelled word *word*.

Setting the Language

`setLanguage` (page 1384)
Sets the language to use in spell checking to *language*.

`language` (page 1383)
Returns the current language used in spell checking.

Managing the Spelling Process

`uniqueSpellDocumentTag` (page 1381)
Returns a guaranteed unique tag to use as the spell-document tag for a document.

`closeSpellDocumentWithTag` (page 1382)
Notifies the receiver that the user has finished with the ignored-word document identified by *tag*, causing it to throw that dictionary away.

`ignoreWord` (page 1383)
Instructs the spell checker to ignore all future occurrences of *wordToIgnore* in the document identified by *tag*.

`setIgnoredWords` (page 1384)
Initializes the ignored-words document (a dictionary identified by *tag* with *someWords*), an array of words to ignore.

`ignoredWords` (page 1383)
Returns the array of ignored words for a document identified by *tag*.

`setWordFieldStringValue` (page 1384)
Sets the string that appears in the misspelled word field, using the string object *aString*.

`updateSpellingPanelWithMisspelledWord` (page 1384)
Causes the spell checker to update the Spelling panel's misspelled-word field to reflect *word*.

`completionsForPartialWordRange` (page 1382)

Constructors

NSSpellChecker

Creates an empty NSSpellChecker.

```
public NSSpellChecker()
```

Static Methods

sharedSpellChecker

Returns the NSSpellChecker (one per application).

```
public static NSSpellChecker sharedSpellChecker()
```

See Also
`sharedSpellCheckerExists` (page 1381)

sharedSpellCheckerExists

Returns whether the application's NSSpellChecker has already been created.

```
public static boolean sharedSpellCheckerExists()
```

See Also
`sharedSpellChecker` (page 1381)

uniqueSpellDocumentTag

Returns a guaranteed unique tag to use as the spell-document tag for a document.

```
public static int uniqueSpellDocumentTag()
```

Discussion
Use this method to generate tags to avoid collisions with other objects that can be spell checked.

Instance Methods

accessoryView

Returns the Spelling panel's accessory NSView object.

```
public NSView accessoryView()
```

See Also
`setAccessoryView` (page 1383)

checkSpellingOfString

Starts the search for a misspelled word in *stringToCheck* starting at *startingOffset* within the string object.

```
public NSRange checkSpellingOfString(String stringToCheck, int startingOffset)
```

Discussion
Returns the range of the first misspelled word. Wrapping occurs, but no ignored-words dictionary is used.

closeSpellDocumentWithTag

Notifies the receiver that the user has finished with the ignored-word document identified by *tag*, causing it to throw that dictionary away.

```
public void closeSpellDocumentWithTag(int tag)
```

completionsForPartialWordRange

```
public NSArray completionsForPartialWordRange(NSRange range, String string, String
     language, int tag)
```

Discussion
Returns an array of strings from the spell checker dictionary representing complete words that the user might be trying to type, based off a partial word at the given range. Strings are returned in the order they should be presented. The *language* argument specifies the language used in the string. If *language* is `null`, the current selection in the Spelling panel's pop-up menu is used.

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

countWordsInString

Returns the number of words in *stringToCount*.

```
public int countWordsInString(String stringToCount, String language)
```

Discussion
The *language* argument specifies the language used in the string. If *language* is `null`, the current selection in the Spelling panel's pop-up menu is used.

guessesForWord

Returns an array of suggested spellings for the misspelled word *word*.

```
public NSArray guessesForWord(String word)
```

Discussion

If *word* contains all capital letters, or its first letter is capitalized, the suggested words are capitalized in the same way.

ignoredWords

Returns the array of ignored words for a document identified by *tag*.

```
public NSArray ignoredWords(int tag)
```

Discussion

Invoke this method before closeSpellDocumentWithTag (page 1382) if you want to store the ignored words.

See Also

setIgnoredWords (page 1384)

ignoreWord

Instructs the spell checker to ignore all future occurrences of *wordToIgnore* in the document identified by *tag*.

```
public void ignoreWord(String wordToIgnore, int tag)
```

Discussion

You should invoke this method from within your implementation of the NSIgnoreMisspelledWords interface's ignoreSpelling (page 1974) method.

language

Returns the current language used in spell checking.

```
public String language()
```

See Also

setLanguage (page 1384)

setAccessoryView

Makes an NSView object an accessory of the Spelling panel by making it a subview of the panel's content view.

```
public void setAccessoryView(NSView aView)
```

Discussion

This method posts a WindowDidResizeNotification (page 1885) with the Spelling panel object to the default notification center.

See Also

accessoryView (page 1381)

setIgnoredWords

Initializes the ignored-words document (a dictionary identified by *tag* with *someWords*), an array of words to ignore.

```
public void setIgnoredWords(NSArray someWords, int tag)
```

See Also

ignoredWords (page 1383)

setLanguage

Sets the language to use in spell checking to *language*.

```
public boolean setLanguage(String language)
```

Discussion

Returns whether the Language pop-up list in the Spelling panel lists *language*.

See Also

language (page 1383)

setWordFieldStringValue

Sets the string that appears in the misspelled word field, using the string object *aString*.

```
public void setWordFieldStringValue(String aString)
```

spellingPanel

Returns the spell checker's panel.

```
public NSPanel spellingPanel()
```

updateSpellingPanelWithMisspelledWord

Causes the spell checker to update the Spelling panel's misspelled-word field to reflect *word*.

```
public void updateSpellingPanelWithMisspelledWord(String word)
```

Discussion

You are responsible for highlighting *word* in the document and for extracting it from the document using the range returned by checkSpellingOfString (page 1382). Pass the empty string as *word* to have the system beep, indicating no misspelled words were found.

NSSplitView

Inherits from NSView : NSResponder : NSObject

Implements NSCoding (NSResponder)

Package: com.apple.cocoa.application

Companion guide Drawing and Views

Overview

An NSSplitView object stacks several subviews within one view so that the user can change their relative sizes. By default, the split bars between the views are horizontal, so the views are one on top of the other.

Tasks

Constructors

`NSSplitView` (page 1387)
Creates an NSSplitView with a zero-sized frame rectangle.

Managing Component Views

`adjustSubviews` (page 1387)
Adjusts the sizes of the receiver's subviews so they (plus the dividers) fill the receiver.

`dividerThickness` (page 1387)
Returns the thickness of the divider.

`drawDividerInRect` (page 1388)
Draws the divider between two of the receiver's subviews.

`isSubviewCollapsed` (page 1388)
Returns `true` if *`subview`* is in a collapsed state, `false` otherwise.

Managing Orientation

`isVertical` (page 1388)
Returns `true` if the split bars are vertical (subviews are side by side), `false` if they are horizontal (views are one on top of the other).

`setVertical` (page 1389)
Sets whether the split bars are vertical.

Assigning a Delegate

`delegate` (page 1387)
Returns the receiver's delegate.

`setDelegate` (page 1389)
Makes *`anObject`* the receiver's delegate.

Managing Pane Splitters

`isPaneSplitter` (page 1388)
Returns `true` if the receiver's splitter is a bar that goes across the split view. Returns `false` if the splitter is a thumb on the regular background pattern.

`setIsPaneSplitter` (page 1389)
Sets the type of splitter.

Resizing subviews

`splitViewResizeSubviews` (page 1391) *delegate method*
Allows the delegate to specify custom sizing behavior for the subviews of the NSSplitView *`sender`*.

`splitViewWillResizeSubviews` (page 1391) *delegate method*

`splitViewDidResizeSubviews` (page 1391) *delegate method*

Constraining split position

`splitViewConstrainMaxSplitPosition` (page 1390) *delegate method*
Allows the delegate for *`sender`* to constrain the maximum coordinate limit of a divider when the user drags it.

`splitViewConstrainMinSplitPosition` (page 1390) *delegate method*
Allows the delegate for *`sender`* to constrain the minimum coordinate limit of a divider when the user drags it.

`splitViewConstrainSplitPosition` (page 1391) *delegate method*
Allows the delegate for *`sender`* to constrain the divider to certain positions.

Collapsing subview

`splitViewCanCollapseSubview` (page 1389) *delegate method*
Allows the delegate to determine whether the user can collapse and uncollapse *`subview`*.

Constructors

NSSplitView

Creates an NSSplitView with a zero-sized frame rectangle.

```
public NSSplitView()
```

Creates an NSSplitView with *`frameRect`* as its frame rectangle.

```
public NSSplitView(NSRect frameRect)
```

Instance Methods

adjustSubviews

Adjusts the sizes of the receiver's subviews so they (plus the dividers) fill the receiver.

```
public void adjustSubviews()
```

Discussion
The subviews are resized proportionally; the size of a subview relative to the other subviews doesn't change.

See Also
`setDelegate` (page 1389)
`setFrame` (page 1776) (NSView)

delegate

Returns the receiver's delegate.

```
public Object delegate()
```

See Also
`setDelegate` (page 1389)

dividerThickness

Returns the thickness of the divider.

```
public float dividerThickness()
```

Discussion

You can subclass NSSplitView and override this method to change the divider's size, if necessary.

See Also

`drawDividerInRect` (page 1388)

drawDividerInRect

Draws the divider between two of the receiver's subviews.

```
public void drawDividerInRect(NSRect aRect)
```

Discussion

aRect describes the entire divider rectangle in the receiver's coordinates, which are flipped. If you override this method and use a custom icon to identify the divider, you may need to change the size of the divider.

See Also

`dividerThickness` (page 1387)

`compositeToPoint` (page 756) (NSImage)

isPaneSplitter

Returns `true` if the receiver's splitter is a bar that goes across the split view. Returns `false` if the splitter is a thumb on the regular background pattern.

```
public boolean isPaneSplitter()
```

See Also

`setIsPaneSplitter` (page 1389)

isSubviewCollapsed

Returns `true` if *subview* is in a collapsed state, `false` otherwise.

```
public boolean isSubviewCollapsed(NSView subview)
```

isVertical

Returns `true` if the split bars are vertical (subviews are side by side), `false` if they are horizontal (views are one on top of the other).

```
public boolean isVertical()
```

Discussion

By default, split bars are vertical.

See Also

`setVertical` (page 1389)

setDelegate

Makes *anObject* the receiver's delegate.

```
public void setDelegate(Object anObject)
```

Discussion
The notification messages the delegate can expect to receive are listed in "Notifications" (page 1392). The delegate doesn't need to implement all of the delegate methods.

See Also
`delegate` (page 1387)

setIsPaneSplitter

Sets the type of splitter.

```
public void setIsPaneSplitter(boolean flag)
```

Discussion
If *flag* is `true`, the receiver's splitter is a bar that goes across the split view. If *flag* is `false`, the splitter is a thumb on the regular background pattern.

See Also
`isPaneSplitter` (page 1388)

setVertical

Sets whether the split bars are vertical.

```
public void setVertical(boolean flag)
```

Discussion
If *flag* is `true`, they're vertical (views are side by side); if it's `false`, they're horizontal (views are one on top of the other). Split bars are horizontal by default.

See Also
`isVertical` (page 1388)

Delegate Methods

splitViewCanCollapseSubview

Allows the delegate to determine whether the user can collapse and uncollapse *subview*.

```
public abstract boolean splitViewCanCollapseSubview(NSSplitView sender, NSView
    subview)
```

Discussion
If this method returns `false` or is undefined, *subview* can't be collapsed. If this method returns `true`, *subview* collapses when the user drags a divider beyond the halfway mark between its minimum size and its edge. *subview* uncollapses when the user drags the divider back beyond that point. To specify the minimum size, define the methods `splitViewConstrainMaxSplitPosition` (page 1390) and `splitViewConstrainMinSplitPosition` (page 1390). Note that a subview can collapse only if you also define `splitViewConstrainMinSplitPosition` (page 1390).

A collapsed subview is hidden by the NSSplitView object, with the same size it had before it was collapsed.

splitViewConstrainMaxSplitPosition

Allows the delegate for *sender* to constrain the maximum coordinate limit of a divider when the user drags it.

```
public abstract float splitViewConstrainMaxSplitPosition(NSSplitView sender, float
     proposedMax, int offset)
```

Discussion
This method is invoked before the NSSplitView begins tracking the mouse to position a divider. You may further constrain the limits that have been already set, but you cannot extend the divider limits. *proposedMax* is specified in the NSSplitView's flipped coordinate system. If the split bars are horizontal (views are one on top of the other), *proposedMax* is the bottom limit. If the split bars are vertical (views are side by side), *proposedMax* is the right limit. The initial value of *proposedMax* is the bottom (or right side) of the subview after the divider. *offset* specifies the divider the user is moving, with the first divider being 0 and going up from top to bottom (or left to right).

See Also
`isVertical` (page 1388)

splitViewConstrainMinSplitPosition

Allows the delegate for *sender* to constrain the minimum coordinate limit of a divider when the user drags it.

```
public abstract float splitViewConstrainMinSplitPosition(NSSplitView sender, float
     proposedMin, int offset)
```

Discussion
This method is invoked before the NSSplitView begins tracking the cursor to position a divider. You may further constrain the limits that have been already set, but you cannot extend the divider limits. *proposedMin* is specified in the NSSplitView's flipped coordinate system. If the split bars are horizontal (views are one on top of the other), *proposedMin* is the top limit. If the split bars are vertical (views are side by side), *proposedMin* is the left limit. The initial value of *proposedMin* is the top (or left side) of the subview before the divider. *offset* specifies the divider the user is moving, with the first divider being 0 and going up from top to bottom (or left to right).

See Also
`isVertical` (page 1388)

splitViewConstrainSplitPosition

Allows the delegate for *sender* to constrain the divider to certain positions.

```
public abstract float splitViewConstrainSplitPosition(NSSplitView sender, float
    proposedPosition, int offset)
```

Discussion

If the delegate implements this method, the NSSplitView calls it repeatedly as the user moves the divider. This method returns where you want the divider to be, given *proposedPosition*, the cursor's current position. *offset* is the divider the user is moving, with the first divider being 0 and going up from top to bottom (or from left to right).

For example, if a subview's height must be a multiple of a certain number, use this method to return the multiple nearest to *proposedPosition*.

splitViewDidResizeSubviews

```
public abstract void splitViewDidResizeSubviews(NSNotification aNotification)
```

Discussion

Sent by the default notification center to the delegate; *aNotification* is always a `SplitViewDidResizeSubviewsNotification` (page 1392). If the delegate implements this method, the delegate is automatically registered to receive this notification. This method is invoked after the NSSplitView resizes two of its subviews in response to the repositioning of a divider.

splitViewResizeSubviews

Allows the delegate to specify custom sizing behavior for the subviews of the NSSplitView *sender*.

```
public abstract void splitViewResizeSubviews(NSSplitView sender, NSSize oldSize)
```

Discussion

If the delegate implements this method, `splitViewResizeSubviews` is invoked after the NSSplitView is resized. The size of the NSSplitView before the user resized it is indicated by *oldSize*; the subviews should be resized such that the sum of the sizes of the subviews plus the sum of the thickness of the dividers equals the size of the NSSplitView's new frame. You can get the thickness of a divider through the *dividerThickness* method.

Note that if you implement this delegate method to resize subviews on your own, the NSSplitView does not perform any error checking for you. However, you can invoke `adjustSubviews` (page 1387) to perform the default sizing behavior.

See Also

`adjustSubviews` (page 1387)

`setFrame` (page 1776) (NSView)

splitViewWillResizeSubviews

```
public abstract void splitViewWillResizeSubviews(NSNotification aNotification)
```

Discussion
Sent by the default notification center to the delegate; *aNotification* is always a `SplitViewWillResizeSubviewsNotification` (page 1392). If the delegate implements this method, the delegate is automatically registered to receive this notification. This method is invoked before the NSSplitView resizes two of its subviews in response to the repositioning of a divider.

Notifications

NSSplitView declares and posts the following notifications. In addition, it posts notifications declared by its superclass, NSView. See the NSView (page 1725) class specification for more information.

SplitViewDidResizeSubviewsNotification

Posted after an NSSplitView changes the sizes of some or all of its subviews. The notification object is the NSSplitView that resized its subviews. This notification does not contain a *userInfo* dictionary.

See Also
`splitViewDidResizeSubviews` (page 1391)

SplitViewWillResizeSubviewsNotification

Posted before an NSSplitView changes the sizes of some or all of its subviews. The notification object is the NSSplitView object that is about to resize its subviews. This notification does not contain a *userInfo* dictionary.

See Also
`splitViewWillResizeSubviews` (page 1391)

NSStatusBar

Inherits from	NSObject
Package:	com.apple.cocoa.application
Companion guide	Status Bars

Overview

The NSStatusBar class defines an object that manages a collection of NSStatusItems displayed within the system-wide menu bar. A status item can be displayed with text or an icon, can provide a menu and a target-action message when clicked, or can be a fully customized view that you create.

Use status items sparingly and only if the alternatives (such as a Dock menu, preference pane, or status window) are not suitable. Because there is limited space in which to display status items, status items are not guaranteed to be available at all times. For this reason, do not rely on them being available and always provide a user preference for hiding your application's status items to free up space in the menu bar.

Tasks

Constructors

`NSStatusBar` (page 1394)
Creates an empty NSStatusBar.

Accessing the System-wide Instance

`systemStatusBar` (page 1394)
Returns the system-wide status bar located in the menu bar.

Getting the Orientation

`isVertical` (page 1394)
Returns `true` if the receiver has a vertical orientation.

`thickness` (page 1395)
Returns the thickness of the status bar.

Creating and Removing Items

`removeStatusItem` (page 1395)
    Removes *`item`* from the receiver.

`statusItem` (page 1395)
    Returns a newly created NSStatusItem object that has been allotted *`length`* pixels of space within the status bar.

Constructors

NSStatusBar

Creates an empty NSStatusBar.

```
public NSStatusBar()
```

Discussion
Use `systemStatusBar` (page 1394) to obtain the system-wide status bar in the menu bar.

Static Methods

systemStatusBar

Returns the system-wide status bar located in the menu bar.

```
public static NSStatusBar systemStatusBar()
```

Discussion
The status bar begins at the right side of the menu bar (to the left of Menu Extras and the menu bar clock) and grows to the left as NSStatusItems are added to it.

Instance Methods

isVertical

Returns `true` if the receiver has a vertical orientation.

```
public boolean isVertical()
```

Discussion
The status bar returned by `systemStatusBar` (page 1394) is horizontal, so it always returns `false`.

removeStatusItem

Removes *item* from the receiver.

```
public void removeStatusItem(NSStatusItem item)
```

Discussion
Status items to the left of it in the status bar shift to the right to reclaim its space.

See Also
statusItem (page 1395)

statusItem

Returns a newly created NSStatusItem object that has been allotted *length* pixels of space within the status bar.

```
public NSStatusItem statusItem(float length)
```

See Also
removeStatusItem (page 1395)

thickness

Returns the thickness of the status bar.

```
public float thickness()
```

Discussion
The status bar returned by systemStatusBar (page 1394) has a thickness of 22 pixels, the thickness of the menu bar.

Constants

The following constants are defined as a convenience by NSStatusBar:

Constant	Description
SquareStatusItemLength	Sets the status item length to the status bar thickness.
VariableStatusItemLength	Makes the status item length dynamic, adjusting to the width of its contents.

NSStatusItem

Inherits from NSObject

Package: com.apple.cocoa.application

Companion guide Status Bars

Overview

The NSStatusItem class represents the individual elements displayed within an NSStatusBar object. Instances are created by the NSStatusBar method statusItem (page 1395), which automatically adds the new status item to the status bar. The appearance and behavior of the status item are then set using the various NSStatusItem methods, such as setTitle (page 1405) and setAction (page 1402).

Tasks

Constructors

NSStatusItem (page 1399)
Creates an empty NSStatusItem.

Getting the Item's Status Bar

statusBar (page 1406)
Returns the status bar in which the receiver is displayed.

Setting the Status Item's Appearance

setAlternateImage (page 1403)
Sets an alternate image to be displayed when a status bar item is highlighted.

setAttributedTitle (page 1403)
Sets the attributed string that is displayed at the receiver's position in the status bar.

setHighlightMode (page 1404)
Sets whether the receiver is highlighted when it is clicked to *flag*.

setImage (page 1404)
Sets the image that is displayed at the receiver's position in the status bar to *image*.

`setLength` (page 1404)
: Sets the amount of space in the status bar that should be allocated to the receiver.

`setTitle` (page 1405)
: Sets the string that is displayed at the receiver's position in the status bar to *`title`*.

`setToolTip` (page 1406)
: Sets the tool tip string that is displayed when the cursor pauses over the receiver to *`toolTip`*.

Getting the Status Item's Appearance

`alternateImage` (page 1400)
: Returns the alternate image that is displayed when a status bar item is highlighted.

`attributedTitle` (page 1400)
: Returns the attributed string that is displayed at the receiver's position in the status bar

`highlightMode` (page 1401)
: Returns whether the receiver is highlighted when clicked.

`image` (page 1401)
: Returns the image that is displayed at the receiver's position in the status bar.

`length` (page 1401)
: Returns the amount of space allocated to the receiver within its status bar.

`title` (page 1406)
: Returns the string that is displayed at the receiver's position in the status bar.

`toolTip` (page 1407)
: Returns the tool tip string that is displayed when the cursor pauses over the receiver.

Setting the Status Item's Behavior

`popUpStatusItemMenu` (page 1402)
: Displays a menu under a custom status bar item.

`setAction` (page 1402)
: Sets the selector that is sent to the receiver's target when the receiver is clicked.

`setDoubleAction` (page 1403)
: Sets the selector that is sent to the receiver's target when the receiver is double-clicked.

`setEnabled` (page 1403)
: Sets whether the receiver is enabled to respond to clicks to *`flag`*.

`setEventMaskForSendingAction` (page 1404)
: Sets the conditions on which the receiver sends action messages to its target.

`setMenu` (page 1405)
: Sets the pull-down menu that is displayed when the receiver is clicked.

`setTarget` (page 1405)
: Sets the target object to which the receiver's action message is sent when the receiver is clicked.

Getting Status Item Behavior

`action` (page 1399)
Returns the selector that is sent to the receiver's target when the user clicks the receiver.

`doubleAction` (page 1400)
Returns the selector that is sent to the receiver's target when the user double-clicks the receiver.

`isEnabled` (page 1401)
Returns whether the receiver is enabled and responding to clicks.

`menu` (page 1402)
Returns the drop-down menu that is displayed when the receiver is clicked.

`target` (page 1406)
Returns the target to which the receiver's action message is sent when the user clicks the receiver.

Using a Custom View

`setView` (page 1406)
Sets the custom view that is displayed at the receiver's position in the status bar to *`view`*.

`view` (page 1407)
Returns the custom view that is displayed at the receiver's position in the status bar.

Drawing

`drawStatusBarBackgroundInRect` (page 1400)
Draws the menu background pattern for a custom status bar item in regular or highlight pattern.

Constructors

NSStatusItem

Creates an empty NSStatusItem.

```
public NSStatusItem()
```

Discussion
Use the NSStatusBar instance method `statusItem` (page 1395) to create an NSStatusItem object within a status bar.

Instance Methods

action

Returns the selector that is sent to the receiver's target when the user clicks the receiver.

```
public NSSelector action()
```

See Also

`setAction` (page 1402)

`target` (page 1406)

alternateImage

Returns the alternate image that is displayed when a status bar item is highlighted.

```
public NSImage alternateImage()
```

Availability

Available in Mac OS X v10.3 and later.

See Also

`setAlternateImage` (page 1403)

`image` (page 1401)

attributedTitle

Returns the attributed string that is displayed at the receiver's position in the status bar

```
public NSAttributedString attributedTitle()
```

Discussion

.

See Also

`setAttributedTitle` (page 1403)

`setTitle` (page 1405)

`title` (page 1406)

doubleAction

Returns the selector that is sent to the receiver's target when the user double-clicks the receiver.

```
public NSSelector doubleAction()
```

Availability

Available in Mac OS X v10.4 and later.

See Also

`setDoubleAction` (page 1403)

`target` (page 1406)

drawStatusBarBackgroundInRect

Draws the menu background pattern for a custom status bar item in regular or highlight pattern.

```
public void drawStatusBarBackgroundInRect(NSRect rect, boolean highlight)
```

Discussion
You can use this method to help a custom status bar item emulate the behavior of a standard item. Set `highlight` to `true` to draw the background pattern in the standard highlight pattern.

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

See Also
`setView` (page 1406)

highlightMode

Returns whether the receiver is highlighted when clicked.

```
public boolean highlightMode()
```

See Also
`setHighlightMode` (page 1404)

image

Returns the image that is displayed at the receiver's position in the status bar.

```
public NSImage image()
```

Discussion
Returns `null` if an image has not been set.

See Also
`setImage` (page 1404)

isEnabled

Returns whether the receiver is enabled and responding to clicks.

```
public boolean isEnabled()
```

See Also
`setEnabled` (page 1403)

length

Returns the amount of space allocated to the receiver within its status bar.

```
public float length()
```

Discussion
If the status bar is horizontal, the return value is the width of the status item. Besides a physical length, the return value may be `NSStatusBar.SquareStatusItemLength` or `NSStatusBar.VariableStatusItemLength` (see NSStatusBar "Constants" (page 1395)), if the status item size is either determined by the status bar thickness or allowed to vary according to the status item's true size, respectively.

See Also
`setLength` (page 1404)
`statusItem` (page 1395) (NSStatusBar)

menu

Returns the drop-down menu that is displayed when the receiver is clicked.

```
public NSMenu menu()
```

See Also
`setMenu` (page 1405)

popUpStatusItemMenu

Displays a menu under a custom status bar item.

```
public void popUpStatusItemMenu(NSMenu menu)
```

Discussion
You can use this method to cause a popup menu to appear under a custom status bar item when the user clicks the item. Note that [self view] must exist (that is, it must not be `nil`).

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

See Also
`setMenu` (page 1405)
`setView` (page 1406)

setAction

Sets the selector that is sent to the receiver's target when the receiver is clicked.

```
public void setAction(NSSelector action)
```

Discussion
If the receiver has a menu set, *`action`* is not sent to the target when the receiver is clicked; instead, the click causes the menu to appear.

See *Action Messages* for additional information on action messages.

See Also
`action` (page 1399)

`setMenu` (page 1405)

setAlternateImage

Sets an alternate image to be displayed when a status bar item is highlighted.

```
public void setAlternateImage(NSImage image)
```

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

See Also
`alternateImage` (page 1400)
`setImage` (page 1404)

setAttributedTitle

Sets the attributed string that is displayed at the receiver's position in the status bar.

```
public void setAttributedTitle(NSAttributedString title)
```

Discussion
If an image is also set, the title appears to the right of the image.

See Also
`attributedTitle` (page 1400)
`setImage` (page 1404)
`setTitle` (page 1405)

setDoubleAction

Sets the selector that is sent to the receiver's target when the receiver is double-clicked.

```
public void setDoubleAction(NSSelector action)
```

Discussion
For the method to have any effect, the receiver's action and target must be set to the class in which the selector is declared. See *Action Messages* for additional information on action messages.

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

See Also
`doubleAction` (page 1400)

setEnabled

Sets whether the receiver is enabled to respond to clicks to `flag`.

```
public void setEnabled(boolean flag)
```

See Also

`isEnabled` (page 1401)

setEventMaskForSendingAction

Sets the conditions on which the receiver sends action messages to its target.

```
public void setEventMaskForSendingAction(int mask)
```

Discussion

mask is set with one or more of the following bit masks described in NSEvent "Constants" (page 623): `NSEvent.LeftMouseUpMask`, `NSEvent.LeftMouseDownMask`, `NSEvent.LeftMouseDraggedMask`, and `NSEvent.PeriodicMask`. The default is `NSEvent.LeftMouseUpMask`.

setHighlightMode

Sets whether the receiver is highlighted when it is clicked to *flag*.

```
public void setHighlightMode(boolean flag)
```

Discussion

The default is `false`.

See Also

`highlightMode` (page 1401)

setImage

Sets the image that is displayed at the receiver's position in the status bar to *image*.

```
public void setImage(NSImage image)
```

Discussion

If a title is also set, the image appears to the left of the title.

See Also

`image` (page 1401)

`setAttributedTitle` (page 1403)

`setTitle` (page 1405)

setLength

Sets the amount of space in the status bar that should be allocated to the receiver.

```
public void setLength(float len)
```

Discussion

If the status bar is horizontal, *len* is the horizontal space to allocate. In addition to a fixed length, *len* can be `NSStatusBar.SquareStatusItemLength` or `NSStatusBar.VariableStatusItemLength` (see NSStatusBar "Constants" (page 1395)) to allow the status bar to allocate (and adjust) the space according to either the status bar's thickness or the status item's true size.

See Also

`length` (page 1401)

`statusItem` (page 1395) (NSStatusBar)

setMenu

Sets the pull-down menu that is displayed when the receiver is clicked.

```
public void setMenu(NSMenu menu)
```

Discussion

When set, the receiver's single click action behavior is not used. The menu can be removed by setting *menu* to `null`.

See Also

`menu` (page 1402)

`setAction` (page 1402)

`setTarget` (page 1405)

setTarget

Sets the target object to which the receiver's action message is sent when the receiver is clicked.

```
public void setTarget(Object target)
```

Discussion

If the receiver has a menu set, the action is not sent to *target* when the receiver is clicked; instead, the click causes the menu to appear.

See Also

`target` (page 1406)

`setMenu` (page 1405)

setTitle

Sets the string that is displayed at the receiver's position in the status bar to *title*.

```
public void setTitle(String title)
```

Discussion

If an image is also set, the title appears to the right of the image.

See Also

`title` (page 1406)

`setAttributedTitle` (page 1403)

`setImage` (page 1404)

setToolTip

Sets the tool tip string that is displayed when the cursor pauses over the receiver to *toolTip*.

```
public void setToolTip(String toolTip)
```

See Also
`toolTip` (page 1407)

setView

Sets the custom view that is displayed at the receiver's position in the status bar to *view*.

```
public void setView(NSView view)
```

Discussion
Setting a custom view overrides all the other appearance and behavior settings defined by `NSStatusItem`. The custom view is responsible for drawing itself and providing its own behaviors, such as processing mouse clicks and sending action messages.

See Also
`view` (page 1407)

statusBar

Returns the status bar in which the receiver is displayed.

```
public NSStatusBar statusBar()
```

target

Returns the target to which the receiver's action message is sent when the user clicks the receiver.

```
public Object target()
```

See Also
`setTarget` (page 1405)
`action` (page 1399)

title

Returns the string that is displayed at the receiver's position in the status bar.

```
public String title()
```

See Also
`setAttributedTitle` (page 1403)

setTitle (page 1405)

toolTip

Returns the tool tip string that is displayed when the cursor pauses over the receiver.

```
public String toolTip()
```

See Also

setToolTip (page 1406)

view

Returns the custom view that is displayed at the receiver's position in the status bar.

```
public NSView view()
```

See Also

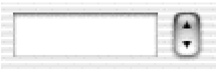
setView (page 1406)

NSStepper

Inherits from	NSControl : NSView : NSResponder : NSObject
Implements	NSCoding (NSResponder)
Package:	com.apple.cocoa.application
Companion guide	Steppers

Overview

A stepper consists of two small arrows that can increment and decrement a value that appears beside it, such as a date or time. The illustration below shows an NSStepper to the right of a text field, which would show the stepper's value.



NSStepper uses NSStepperCell (page 1415) to implement its user interface.

Tasks

Constructors

`NSStepper` (page 1410)
Creates an NSStepper with a zero-sized frame rectangle.

Specifying Value Range

`maxValue` (page 1411)
Returns the maximum value for the receiver.

`setMaxValue` (page 1412)
Sets the maximum value for the receiver to *`maxValue`*.

`minValue` (page 1411)
Returns the minimum value for the receiver.

`setMinValue` (page 1412)
Specifies the minimum value for the receiver to *`minValue`*.

`increment` (page 1411)
  Returns the amount by which the receiver will change per increment (decrement).

`setIncrement` (page 1412)
  Sets the amount by which the receiver will change per increment (decrement) to *`increment`*.

Specifying How Stepper Responds

`autorepeat` (page 1410)
  Returns how the receiver responds to mouse events.

`setAutorepeat` (page 1411)
  Sets how the receiver responds to mouse events.

`valueWraps` (page 1412)
  Returns whether the receiver wraps around the minimum and maximum values.

`setValueWraps` (page 1412)
  Sets whether the receiver wraps around the minimum and maximum values.

Constructors

NSStepper

Creates an NSStepper with a zero-sized frame rectangle.

```
public NSStepper()
```

Creates an NSStepper with *`frameRect`* as its frame rectangle.

```
public NSStepper(NSRect frameRect)
```

Instance Methods

autorepeat

Returns how the receiver responds to mouse events.

```
public boolean autorepeat()
```

Discussion
If `true`, the first mouse down will do one increment (decrement) and, after a delay of 0.5 seconds, will increment (decrement) at a rate of ten times per second. If `false`, the receiver will do one increment (decrement) on a mouse up. The default is `true`.

See Also
`setAutorepeat` (page 1411)

increment

Returns the amount by which the receiver will change per increment (decrement).

```
public double increment()
```

Discussion
The default is 1.

See Also
`setIncrement` (page 1412)

maxValue

Returns the maximum value for the receiver.

```
public double maxValue()
```

Discussion
The default is 59.

See Also
`setMaxValue` (page 1412)

minValue

Returns the minimum value for the receiver.

```
public double minValue()
```

Discussion
The default is 0.

See Also
`setMinValue` (page 1412)

setAutorepeat

Sets how the receiver responds to mouse events.

```
public void setAutorepeat(boolean autorepeat)
```

Discussion
If *autorepeat* is `true`, the first mouse down will do one increment (decrement) and, after a delay of 0.5 seconds, will increment (decrement) at a rate of ten times per second. If *autorepeat* is `false`, the receiver will do one increment (decrement) on a mouse up.

See Also
`autorepeat` (page 1410)

setIncrement

Sets the amount by which the receiver will change per increment (decrement) to *increment*.

```
public void setIncrement(double increment)
```

See Also
increment (page 1411)

setMaxValue

Sets the maximum value for the receiver to *maxValue*.

```
public void setMaxValue(double maxValue)
```

See Also
maxValue (page 1411)

setMinValue

Specifies the minimum value for the receiver to *minValue*.

```
public void setMinValue(double minValue)
```

See Also
minValue (page 1411)

setValueWraps

Sets whether the receiver wraps around the minimum and maximum values.

```
public void setValueWraps(boolean valueWraps)
```

Discussion
If *valueWraps* is `true`, then when incrementing or decrementing, the value will wrap around to the minimum or maximum. If *valueWraps* is `false`, the value will stay pinned at the minimum or maximum.

See Also
valueWraps (page 1412)

valueWraps

Returns whether the receiver wraps around the minimum and maximum values.

```
public boolean valueWraps()
```

Discussion
If `true`, then when incrementing or decrementing, the value will wrap around to the minimum or maximum. If `false`, the value will stay pinned at the minimum or maximum. The default is `true`.

See Also

`setValueWraps` (page 1412)

NSStepperCell

Inherits from	NSActionCell : NSCell : NSObject
Implements	NSCoding (NSCell)
Package:	com.apple.cocoa.application
Companion guide	Steppers

Overview

An NSStepperCell controls the appearance and behavior of an NSStepper (page 1409).

Tasks

Constructors

`NSStepperCell` (page 1416)
Creates an empty NSStepperCell.

Specifying Value Range

`maxValue` (page 1417)
Returns the maximum value for the receiver.

`setMaxValue` (page 1418)
Sets the maximum value for the receiver to *`maxValue`*.

`minValue` (page 1417)
Returns the minimum value for the receiver.

`setMinValue` (page 1418)
Specifies the minimum value for the receiver to *`minValue`*.

`increment` (page 1417)
Returns the amount by which the receiver will change per increment (decrement).

`setIncrement` (page 1418)
Sets the amount by which the receiver will change per increment (decrement) to *`increment`*.

Specifying How Stepper Cell Responds

`autorepeat` (page 1416)
Returns how the receiver responds to mouse events.

`setAutorepeat` (page 1417)
Sets how the receiver responds to mouse events.

`valueWraps` (page 1418)
Returns whether the receiver wraps around the minimum and maximum values.

`setValueWraps` (page 1418)
Sets whether the receiver wraps around the minimum and maximum values.

Constructors

NSStepperCell

Creates an empty NSStepperCell.

```
public NSStepperCell()
```

Creates an NSStepperCell initialized with *aString*.

```
public NSStepperCell(String aString)
```

Creates an NSStepperCell initialized with *anImage*.

```
public NSStepperCell(NSImage anImage)
```

Discussion
If *anImage* is `null`, no image is set.

Instance Methods

autorepeat

Returns how the receiver responds to mouse events.

```
public boolean autorepeat()
```

Discussion
If `true`, the first mouse down will do one increment (decrement), and, after a delay of 0.5 seconds, will increment (decrement) at a rate of ten times per second. If `false`, the receiver will do one increment (decrement) on a mouse up. The default is `true`.

See Also
`setAutorepeat` (page 1417)

increment

Returns the amount by which the receiver will change per increment (decrement).

```
public double increment()
```

Discussion
The default is 1.

See Also
`setIncrement` (page 1418)

maxValue

Returns the maximum value for the receiver.

```
public double maxValue()
```

Discussion
The default is 59.

See Also
`setMaxValue` (page 1418)

minValue

Returns the minimum value for the receiver.

```
public double minValue()
```

Discussion
The default is 0.

See Also
`setMinValue` (page 1418)

setAutorepeat

Sets how the receiver responds to mouse events.

```
public void setAutorepeat(boolean autorepeat)
```

Discussion
If *autorepeat* is `true`, the first mouse down will do one increment (decrement) and, after a delay of 0.5 seconds, will increment (decrement) at a rate of ten times per second. If *autorepeat* is `false`, the receiver will do one increment (decrement) on a mouse up.

See Also
`autorepeat` (page 1416)

setIncrement

Sets the amount by which the receiver will change per increment (decrement) to *increment*.

```
public void setIncrement(double increment)
```

See Also

increment (page 1417)

setMaxValue

Sets the maximum value for the receiver to *maxValue*.

```
public void setMaxValue(double maxValue)
```

See Also

maxValue (page 1417)

setMinValue

Specifies the minimum value for the receiver to *minValue*.

```
public void setMinValue(double minValue)
```

See Also

minValue (page 1417)

setValueWraps

Sets whether the receiver wraps around the minimum and maximum values.

```
public void setValueWraps(boolean valueWraps)
```

Discussion

If *valueWraps* is `true`, then when incrementing or decrementing, the value will wrap around to the minimum or maximum. If *valueWraps* is `false`, the value will stay pinned at the minimum or maximum.

See Also

valueWraps (page 1418)

valueWraps

Returns whether the receiver wraps around the minimum and maximum values.

```
public boolean valueWraps()
```

Discussion

If `true`, then when incrementing or decrementing, the value will wrap around to the minimum or maximum. If `false`, the value will stay pinned at the minimum or maximum. The default is `true`.

See Also

`setValueWraps` (page 1418)

NSTableColumn

Inherits from NSObject

Implements NSCoding

Package: com.apple.cocoa.application

Companion guide Table View Programming Guide

Overview

An NSTableColumn stores the display characteristics and attribute identifier for a column in an NSTableView. The NSTableColumn determines the width and width limits, resizability, and editability of its column in the NSTableView. It also stores two NSCell objects: the header cell, which is used to draw the column header, and the data cell, used to draw the values for each row. You can control the display of the column by setting the subclasses of NSCell used and by setting the font and other display characteristics for these NSCells. For example, you can use the default NSTextFieldCell for displaying string values or substitute an NSImageCell to display pictures.

Tasks

Constructors

`NSTableColumn` (page 1423)
Creates an empty NSTableColumn.

Setting the Identifier

`setIdentifier` (page 1427)
Sets the receiver's identifier to *anObject*.

`identifier` (page 1424)
Returns the object used by the data source to identify the attribute corresponding to the receiver.

Setting the NSTableView

`setTableView` (page 1428)
Sets *aTableView* as the receiver's NSTableView.

`tableView` (page 1430)
Returns the NSTableView the receiver belongs to.

Controlling Size

`setWidth` (page 1429)
Sets the receiver's width to *`newWidth`*.

`width` (page 1430)
Returns the width of the receiver.

`setMinWidth` (page 1427)
Sets the receiver's minimum width to *`minWidth`*, also adjusting the current width if it's less than this value.

`minWidth` (page 1425)
Returns the minimum width for the receiver.

`setMaxWidth` (page 1427)
Sets the receiver's maximum width to *`maxWidth`*, also adjusting the current width if it's greater than this value.

`maxWidth` (page 1425)
Returns the maximum width of the receiver.

`setResizingMask` (page 1428)
Sets the resizing mask for the receiver to *`resizingMask`*.

`resizingMask` (page 1426)
Returns the receiver's resizing mask.

`sizeToFit` (page 1429)
Resizes the receiver to fit the width of its header cell.

Controlling Editability

`setEditable` (page 1426)
Controls whether the user can edit cells in the receiver by double-clicking them.

`isEditable` (page 1424)
Returns `true` if the user can edit cells associated with the receiver by double-clicking the column in the NSTableView, `false` otherwise.

Setting Component Cells

`setHeaderCell` (page 1426)
Sets the NSCell used to draw the receiver's header to *`aCell`*.

`headerCell` (page 1424)
Returns the NSTableHeaderCell object used to draw the header of the receiver.

`setDataCell` (page 1426)
Sets the NSCell used by the NSTableView to draw individual values for the receiver to *`aCell`*.

`dataCell` (page 1423)
Returns the NSCell object used by the NSTableView to draw values for the receiver.

`dataCellForRow` (page 1424)
Returns the NSCell object used by the NSTableView to draw values for the receiver.

Sorting

`setSortDescriptorPrototype` (page 1428)
Sets the receiver's sort descriptor prototype.

`sortDescriptorPrototype` (page 1429)
Returns the receiver's sort descriptor prototype.

Deprecated Methods

`isResizable` (page 1425)
Returns `true` if the user is allowed to resize the receiver in its NSTableView, `false` otherwise.

`setResizable` (page 1427)
Sets whether the user can resize the receiver in its NSTableView.

Constructors

NSTableColumn

Creates an empty NSTableColumn.

```
public NSTableColumn()
```

Creates an NSTableColumn with *anObject* as its identifier and with an NSTextFieldCell as its data cell.

```
public NSTableColumn(Object anObject)
```

Discussion
Send `setStringValue` (page 331) to the header cell to set the column title.

See the NSTableView (page 1437) class specification for information on identifiers.

Instance Methods

dataCell

Returns the NSCell object used by the NSTableView to draw values for the receiver.

```
public NSCell dataCell()
```

See Also

`setDataCell` (page 1426)

dataCellForRow

Returns the NSCell object used by the NSTableView to draw values for the receiver.

```
public NSCell dataCellForRow(int row)
```

Discussion

NSTableView always calls this method. By default, this method just calls `dataCell` (page 1423). Subclassers can override if they need to potentially use different cells for different rows. Subclasses should expect this method to be invoked with *`row`* equal to –1 in cases where no actual row is involved but the table view needs to get some generic cell info.

headerCell

Returns the NSTableHeaderCell object used to draw the header of the receiver.

```
public NSCell headerCell()
```

Discussion

You can set the column title by sending `setStringValue` (page 331) to this object.

See Also

`setHeaderCell` (page 1426)

identifier

Returns the object used by the data source to identify the attribute corresponding to the receiver.

```
public Object identifier()
```

See Also

`setIdentifier` (page 1427)

isEditable

Returns `true` if the user can edit cells associated with the receiver by double-clicking the column in the NSTableView, `false` otherwise.

```
public boolean isEditable()
```

Discussion

You can initiate editing programmatically regardless of this setting with NSTableView's `editLocation` (page 1456) method.

See Also

`setEditable` (page 1426)

isResizable

Returns `true` if the user is allowed to resize the receiver in its NSTableView, `false` otherwise.

```
public boolean isResizable()
```

Discussion

You can change the size programmatically regardless of this setting.

This method is deprecated. You should use `resizingMask` (page 1426) instead.

Availability

Deprecated in Mac OS X v10.4 and later.

See Also

`setWidth` (page 1429)
`setMinWidth` (page 1427)
`setMaxWidth` (page 1427)
`setResizable` (page 1427)

maxWidth

Returns the maximum width of the receiver.

```
public float maxWidth()
```

Discussion

The receiver's width can't be made larger than this size either by the user or programmatically.

See Also

`minWidth` (page 1425)
`width` (page 1430)
`setMaxWidth` (page 1427)
`autoresizesAllColumnsToFit` (page 1449) (NSTableView)

minWidth

Returns the minimum width for the receiver.

```
public float minWidth()
```

Discussion

The receiver's width can't be made less than this size either by the user or programmatically.

See Also

`maxWidth` (page 1425)
`width` (page 1430)
`setMinWidth` (page 1427)
`autoresizesAllColumnsToFit` (page 1449) (NSTableView)

resizingMask

Returns the receiver's resizing mask.

```
public int resizingMask()
```

Discussion

See "Constants" (page 1430) for a description of the resizing mask constants.

Availability

Available in Mac OS X v10.4 and later.

See Also

`setResizingMask` (page 1428)

setDataCell

Sets the NSCell used by the NSTableView to draw individual values for the receiver to *aCell*.

```
public void setDataCell(NSCell aCell)
```

Discussion

You can use this method to control the font, alignment, and other text attributes for an NSTableColumn. You can also assign a cell to display things other than text—for example, an NSImageCell to display images.

See Also

`dataCell` (page 1423)

setEditable

Controls whether the user can edit cells in the receiver by double-clicking them.

```
public void setEditable(boolean flag)
```

Discussion

If *flag* is `true` a double click initiates editing; if *flag* is `false` it merely sends the double-click action to the NSTableView's target. You can initiate editing programmatically regardless of this setting with NSTableView's `editLocation` (page 1456) method.

See Also

`isEditable` (page 1424)

setHeaderCell

Sets the NSCell used to draw the receiver's header to *aCell*.

```
public void setHeaderCell(NSCell aCell)
```

Discussion

aCell should never be `null`.

See Also

`headerCell` (page 1424)

setIdentifier

Sets the receiver's identifier to *anObject*.

```
public void setIdentifier(Object anObject)
```

Discussion
This object is used by the data source to identify the attribute corresponding to the NSTableColumn.

See Also
`identifier` (page 1424)

setMaxWidth

Sets the receiver's maximum width to *maxWidth*, also adjusting the current width if it's greater than this value.

```
public void setMaxWidth(float maxWidth)
```

Discussion
The NSTableView can be made no wider than this size, either by the user or programmatically.

See Also
`setMinWidth` (page 1427)
`setWidth` (page 1429)
`maxWidth` (page 1425)
`autoresizesAllColumnsToFit` (page 1449) (NSTableView)

setMinWidth

Sets the receiver's minimum width to *minWidth*, also adjusting the current width if it's less than this value.

```
public void setMinWidth(float minWidth)
```

Discussion
The NSTableView can be made no less wide than this size, either by the user or programmatically.

See Also
`setMaxWidth` (page 1427)
`setWidth` (page 1429)
`minWidth` (page 1425)
`autoresizesAllColumnsToFit` (page 1449) (NSTableView)

setResizable

Sets whether the user can resize the receiver in its NSTableView.

```
public void setResizable(boolean flag)
```

Discussion

If *flag* is `true` the user can resize the receiver; if *flag* is `false` the user can't resize it. You can always set the size programmatically.

This method is deprecated. You should use `setResizingMask` (page 1428) instead.

Availability

Deprecated in Mac OS X v10.4 and later.

See Also

`isResizable` (page 1425)
`setWidth` (page 1429)
`setMinWidth` (page 1427)
`setMaxWidth` (page 1427)

setResizingMask

Sets the resizing mask for the receiver to *resizingMask*.

```
public void setResizingMask(int resizingMask)
```

Discussion

If *resizingMask* is `0`, the column is not resizable. See "Constants" (page 1430) for the appropriate mask values.

Availability

Available in Mac OS X v10.4 and later.

See Also

`resizingMask` (page 1426)

setSortDescriptorPrototype

Sets the receiver's sort descriptor prototype.

```
public void setSortDescriptorPrototype(NSSortDescriptor sortDescriptor)
```

Discussion

A table column is considered sortable if it has a sort descriptor that specifies the sorting direction, a key to sort by, and a selector defining how to sort.

Availability

Available in Mac OS X v10.3 and later.

See Also

`sortDescriptorPrototype` (page 1429)

setTableView

Sets *aTableView* as the receiver's NSTableView.

```
public void setTableView(NSTableView aTableView)
```

Discussion

You should never need to invoke this method; it's invoked automatically when you add an NSTableColumn to an NSTableView.

See Also

`tableView` (page 1430)

`addTableColumn` (page 1447) (NSTableView)

setWidth

Sets the receiver's width to *`newWidth`*.

```
public void setWidth(float newWidth)
```

Discussion

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 `TableViewColumnDidResizeNotification` (page 1482) on behalf of the receiver's NSTableView.

See Also

`width` (page 1430)

`setMinWidth` (page 1427)

`setMaxWidth` (page 1427)

`autoresizesAllColumnsToFit` (page 1449) (NSTableView)

sizeToFit

Resizes the receiver to fit the width of its header cell.

```
public void sizeToFit()
```

Discussion

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.

See Also

`width` (page 1430)

`minWidth` (page 1425)

`maxWidth` (page 1425)

`autoresizesAllColumnsToFit` (page 1449) (NSTableView)

sortDescriptorPrototype

Returns the receiver's sort descriptor prototype.

```
public NSSortDescriptor sortDescriptorPrototype()
```

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

See Also
`setSortDescriptorPrototype` (page 1428)

tableView

Returns the NSTableView the receiver belongs to.

```
public NSTableView tableView()
```

See Also
`setTableView` (page 1428)

width

Returns the width of the receiver.

```
public float width()
```

Constants

These constants specify the resizing modes available for the table column. You specify either NSTableColumnNoResizing or a resizing a mask created using the C bitwise OR operator. These values are then used by the `setResizingMask` (page 1428) method:

Key	Description
NoResizing	Prevents the table column from resizing. Available in Mac OS X v10.4 and later.
AutoresizingMask	Allows the table column to resize automatically in response to resizing the tableview. Available in Mac OS X v10.4 and later.
UserResizingMask	Allows the table column to be resized explicitly by the user. Available in Mac OS X v10.4 and later.

NSTableHeaderCell

Inherits from	NSTextFieldCell : NSActionCell : NSCell : NSObject
Implements	NSCoding (NSCell)
Package:	com.apple.cocoa.application
Companion guide	Table View Programming Guide

Overview

An NSTableHeaderCell is used by an NSTableHeaderView (page 1433) to draw its column headers. See the NSTableView (page 1437) class specification for more information on how it's used.

Subclasses of NSTableHeaderCell can override `drawInteriorWithFrameInView` (page 309), `editWithFrameInView` (page 310), and `highlightWithFrameInView` (page 312) to change the way headers appear. See the NSCell (page 295) class specification, and the following description, for information on these methods.

Tasks

Constructors

`NSTableHeaderCell` (page 1432)
Creates an empty NSTableHeaderCell.

Sorting

`drawSortIndicatorWithFrameInView` (page 1432)
Draws a sorting indicator given a *`cellFrame`* contained inside *`controlView`*.

`sortIndicatorRectForBounds` (page 1432)
Returns the location to display the sorting indicator given *`theRect`*.

Constructors

NSTableHeaderCell

Creates an empty NSTableHeaderCell.

```
public NSTableHeaderCell()
```

Creates an NSTableHeaderCell initialized with *aString* and set to have the cell's default menu.

```
public NSTableHeaderCell(String aString)
```

Discussion
If no field editor has been created, one is created.

Creates an NSTableHeaderCell initialized with *anImage* and set to have the cell's default menu.

```
public NSTableHeaderCell(NSImage anImage)
```

Discussion
If *anImage* is `null`, no image is set.

Instance Methods

drawSortIndicatorWithFrameInView

Draws a sorting indicator given a *cellFrame* contained inside *controlView*.

```
public void drawSortIndicatorWithFrameInView(NSRect cellFrame, NSView controlView,
     boolean ascending, int priority)
```

Discussion
If *priority* is 0, this is the primary sort indicator. If *ascending* is `true`, a "^" indicator will be drawn. Override this method to customize the sorting user interface.

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

sortIndicatorRectForBounds

Returns the location to display the sorting indicator given *theRect*.

```
public NSRect sortIndicatorRectForBounds(NSRect theRect)
```

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

NSTableHeaderView

Inherits from	NSView : NSResponder : NSObject
Implements	NSCoding (NSResponder)
Package:	com.apple.cocoa.application
Companion guide	Table View Programming Guide

Overview

An NSTableHeaderView is used by an NSTableView to draw headers over its columns and to handle mouse events in those headers.

NSTableHeaderView uses NSTableHeaderCell (page 1431) to implement its user interface.

Tasks

Constructors

`NSTableHeaderView` (page 1434)
Creates an NSTableHeaderView with a zero-sized frame rectangle.

Setting the Table View

`setTableView` (page 1435)
Sets *aTableView* as the receiver's NSTableView.

`tableView` (page 1435)
Returns the NSTableView the receiver belongs to.

Checking Altered Columns

`draggedColumn` (page 1434)
If the user is dragging a column in the receiver, returns the index of that column.

`draggedDistance` (page 1435)
If the user is dragging a column in the receiver, returns the column's horizontal distance from its original position.

`resizedColumn` (page 1435)
    If the user is resizing a column in the receiver, returns the index of that column.

Utility Methods

`columnAtPoint` (page 1434)
    Returns the index of the column whose header lies under *aPoint* in the receiver, or –1 if no such column is found.

`headerRectOfColumn` (page 1435)
    Returns the rectangle containing the header tile for the column at *columnIndex*.

Constructors

NSTableHeaderView

Creates an NSTableHeaderView with a zero-sized frame rectangle.

```
public NSTableHeaderView()
```

Creates an NSTableHeaderView with *frameRect* as its frame rectangle.

```
public NSTableHeaderView(NSRect frameRect)
```

Instance Methods

columnAtPoint

Returns the index of the column whose header lies under *aPoint* in the receiver, or –1 if no such column is found.

```
public int columnAtPoint(NSPoint aPoint)
```

Discussion
aPoint is expressed in the receiver's coordinate system.

draggedColumn

If the user is dragging a column in the receiver, returns the index of that column.

```
public int draggedColumn()
```

Discussion
Otherwise returns –1.

See Also
`draggedDistance` (page 1435)

draggedDistance

If the user is dragging a column in the receiver, returns the column's horizontal distance from its original position.

```
public float draggedDistance()
```

Discussion
Otherwise the return value is meaningless.

See Also
`draggedColumn` (page 1434)

headerRectOfColumn

Returns the rectangle containing the header tile for the column at *`columnIndex`*.

```
public NSRect headerRectOfColumn(int columnIndex)
```

Discussion
Throws an `InternalInconsistencyException` if *`columnIndex`* is out of bounds.

See Also
`rectOfColumn` (page 1461) (NSTableView)

resizedColumn

If the user is resizing a column in the receiver, returns the index of that column.

```
public int resizedColumn()
```

Discussion
Otherwise returns –1.

setTableView

Sets *`aTableView`* as the receiver's NSTableView.

```
public void setTableView(NSTableView aTableView)
```

Discussion
You should never need to invoke this method; it's invoked automatically when you set the header view for an NSTableView.

See Also
`setHeaderView` (page 1472) (NSTableView)

tableView

Returns the NSTableView the receiver belongs to.

```
public NSTableView tableView()
```

NSTableView

Inherits from	NSControl : NSView : NSResponder : NSObject
Implements	NSCoding (NSResponder)
Package:	com.apple.cocoa.application
Companion guides	Table View Programming Guide Drag and Drop Programming Topics for Cocoa

Class at a Glance

An NSTableView object displays record-oriented data in a table and allows the user to edit values and resize and rearrange columns.

Principal Attributes

- A data source
- Table columns

Commonly Used Methods

`dataSource` (page 1452)
Returns the object providing the data that the NSTableView displays.

`tableColumns` (page 1475)
Returns the NSTableColumn objects representing attributes for the NSTableView.

`selectedColumn` (page 1464)
Returns the index of the selected column.

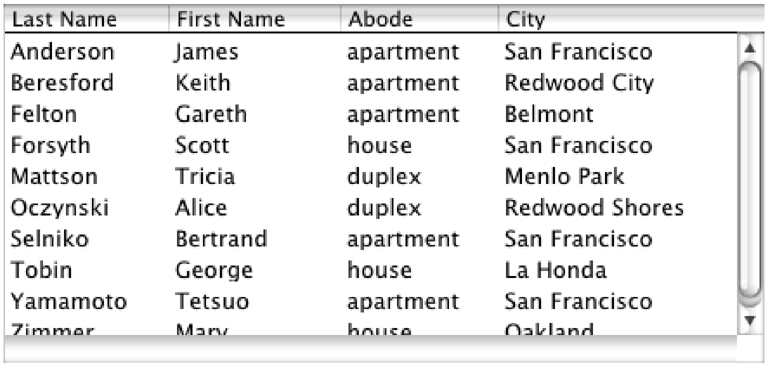
`selectedRow` (page 1465)
Returns the index of the selected row.

`numberOfRows` (page 1460)
Returns the number of rows in the NSTableView.

`reloadData` (page 1462)
Informs the NSTableView that data has changed and needs to be retrieved and displayed again.

Overview

An NSTableView displays data for a set of related records, with rows representing individual records and columns representing the attributes of those records. An NSTableView is usually displayed in an NSScrollView, like this:



Last Name	First Name	Abode	City
Anderson	James	apartment	San Francisco
Beresford	Keith	apartment	Redwood City
Felton	Gareth	apartment	Belmont
Forsyth	Scott	house	San Francisco
Mattson	Tricia	duplex	Menlo Park
Oczynski	Alice	duplex	Redwood Shores
Selniko	Bertrand	apartment	San Francisco
Tobin	George	house	La Honda
Yamamoto	Tetsuo	apartment	San Francisco
Zimmer	Mary	house	Oakland

Also see the NSTableView.DataSource (page 2019) interface, which declares the methods that an NSTableView uses to access the contents of its data source object.

Tasks

Constructors

`NSTableView` (page 1447)
Creates an NSTableView with a zero-sized frame rectangle.

Setting the Data Source

`setDataSource` (page 1470)
Sets the receiver's data source to *`anObject`* and invokes `tile` (page 1476).

`dataSource` (page 1452)
Returns the object that provides the data displayed by the receiver.

Loading Data

`reloadData` (page 1462)
Marks the receiver as needing redisplay, so it will reload the data for visible cells and draw the new values.

Target-action Behavior

`setDoubleAction` (page 1470)
: Sets the message sent to the target when the user double-clicks an uneditable cell or a column header to *`aSelector`*.

`doubleAction` (page 1454)
: Returns the message sent to the target when the user double-clicks a column header or an uneditable cell.

`clickedColumn` (page 1451)
: Returns the index of the column the user clicked to trigger an action message.

`clickedRow` (page 1451)
: Returns the index of the row the user clicked to trigger an action message.

Configuring Behavior

`setAllowsColumnReordering` (page 1467)
: Controls whether the user can drag column headers to reorder columns.

`allowsColumnReordering` (page 1448)
: Returns `true` if the receiver allows the user to rearrange columns by dragging their headers, `false` otherwise.

`setAllowsColumnResizing` (page 1467)
: Controls whether the user can resize columns by dragging between headers.

`allowsColumnResizing` (page 1448)
: Returns `true` if the receiver allows the user to resize columns by dragging between their headers, `false` otherwise.

`setAllowsMultipleSelection` (page 1468)
: Controls whether the user can select more than one row or column at a time.

`allowsMultipleSelection` (page 1449)
: Returns `true` if the receiver allows the user to select more than one column or row at a time, `false` otherwise.

`setAllowsEmptySelection` (page 1467)
: Controls whether the receiver allows zero rows or columns to be selected.

`allowsEmptySelection` (page 1449)
: Returns `true` if the receiver allows the user to select zero columns or rows, `false` otherwise.

`setAllowsColumnSelection` (page 1467)
: Controls whether the user can select an entire column by clicking its header.

`allowsColumnSelection` (page 1448)
: Returns `true` if the receiver allows the user to select columns by clicking their headers, `false` otherwise.

Setting Display Attributes

`setIntercellSpacing` (page 1473)
: Sets the width and height between cells to those in *`aSize`* and redisplays the receiver.

`intercellSpacing` (page 1459)
: Returns the horizontal and vertical spacing between cells.

`setRowHeight` (page 1473)
: Sets the height for rows to *rowHeight* and invokes `tile` (page 1476).

`rowHeight` (page 1462)
: Returns the height of each row in the receiver.

`setBackgroundColor` (page 1469)
: Sets the receiver's background color to *aColor*.

`backgroundColor` (page 1450)
: Returns the color used to draw the background of the receiver.

`setUsesAlternatingRowBackgroundColors` (page 1474)
: Sets whether the receiver uses the standard alternating row colors, or a solid color, for its background.

`usesAlternatingRowBackgroundColors` (page 1477)
: Returns whether the receiver uses the standard alternating row colors, or a solid color, for its background.

Manipulating Columns

`addTableColumn` (page 1447)
: Adds *aColumn* as the last column of the receiver.

`removeTableColumn` (page 1462)
: Removes *aTableColumn* from the receiver.

`moveColumnToColumn` (page 1459)
: Moves the column and heading at *columnIndex* to *newIndex*.

`tableColumns` (page 1475)
: Returns the NSTableColumns in the receiver.

`columnWithIdentifier` (page 1452)
: Returns the index of the first column in the receiver whose identifier is equal to *anObject*, when compared using `equals`, or –1 if no columns are found with the specified identifier.

`tableColumnWithIdentifier` (page 1475)
: Returns the NSTableColumn object for the first column whose identifier is equal to *anObject*, as compared using `equals`, or `null` if no columns are found with the specified identifier.

Selecting Columns and Rows

`selectColumn` (page 1464)
: This method has been deprecated. Use `selectColumnIndexes` (page 1464) instead.

`selectRow` (page 1466)
: This method has been deprecated. Use `selectRowIndexes` (page 1466) instead.

`selectColumnIndexes` (page 1464)
: Sets the column selection using *indexes*.

`selectRowIndexes` (page 1466)
: Sets the row selection using *indexes*.

selectedColumnIndexes (page 1465)
Returns the selected columns.

selectedRowIndexes (page 1466)
Returns the selected rows.

deselectColumn (page 1453)
Deselects the column at *columnIndex* if it's selected, regardless of whether empty selection is allowed.

deselectRow (page 1454)
Deselects the row at *rowIndex* if it's selected, regardless of whether empty selection is allowed.

numberOfSelectedColumns (page 1461)
Returns the number of selected columns.

numberOfSelectedRows (page 1461)
Returns the number of selected rows.

selectedColumn (page 1464)
Returns the index of the last column selected or added to the selection, or –1 if no column is selected.

selectedRow (page 1465)
Returns the index of the last row selected or added to the selection, or –1 if no row is selected.

isColumnSelected (page 1459)
Returns `true` if the column at *columnIndex* is selected, `false` otherwise.

isRowSelected (page 1459)
Returns `true` if the row at *rowIndex* is selected, `false` otherwise.

selectedColumnEnumerator (page 1465)
This method has been deprecated. Use selectedColumnIndexes (page 1465) instead.

selectedRowEnumerator (page 1465)
This method has been deprecated. Use selectedRowIndexes (page 1466) instead.

selectAll (page 1463)

deselectAll (page 1453)
Deselects all selected rows or columns if empty selection is allowed; otherwise does nothing.

Getting the Dimensions of the Table

numberOfColumns (page 1460)
Returns the number of columns in the receiver.

numberOfRows (page 1460)
Returns the number of rows in the receiver.

Setting Grid Attributes

setDrawsGrid (page 1471)

drawsGrid (page 1456)

`setGridColor` (page 1472)
    Sets the color used to draw grid lines to *aColor*.

`gridColor` (page 1457)
    Returns the color used to draw grid lines.

`setGridStyleMask` (page 1472)
    Sets the grid style mask to specify if no grid lines, vertical grid lines, or horizontal grid lines should be displayed.

`gridStyleMask` (page 1457)
    Returns the receiver's grid style mask.

Editing Cells

`editLocation` (page 1456)
    Edits the cell at *columnIndex* and *rowIndex*, selecting its entire contents if *flag* is `true`.

`editedRow` (page 1456)
    If sent during `editLocation` (page 1456), returns the index of the row being edited; otherwise returns –1.

`editedColumn` (page 1456)
    If sent during `editLocation` (page 1456), returns the index of the column being edited; otherwise returns –1.

Setting Auxiliary Views

`setHeaderView` (page 1472)
    Sets the receiver's header view to *aHeaderView*.

`headerView` (page 1458)
    Returns the NSTableHeaderView used to draw headers over columns, or `null` if the receiver has no header view.

`setCornerView` (page 1470)
    Sets the receiver's corner view to *aView*.

`cornerView` (page 1452)
    Returns the NSView used to draw the area to the right of the column headers and above the vertical scroller of the enclosing NSScrollView.

Layout Support

`rectOfColumn` (page 1461)
    Returns the rectangle containing the column at *columnIndex*.

`rectOfRow` (page 1461)
    Returns the rectangle containing the row at *rowIndex*.

`columnsInRect` (page 1452)
    Returns a range of indices for the receiver's columns that lie wholly or partially within the horizontal boundaries of *aRect*.

`rowsInRect` (page 1462)
Returns a range of indices for the rows that lie wholly or partially within the vertical boundaries of *aRect*.

`columnAtPoint` (page 1451)
Returns the index of the column *aPoint* lies in, or –1 if *aPoint* lies outside the receiver's bounds.

`rowAtPoint` (page 1462)
Returns the index of the row *aPoint* lies in, or –1 if *aPoint* lies outside the receiver's bounds.

`frameOfCellAtLocation` (page 1457)
Returns a rectangle locating the cell that lies at the intersection of *columnIndex* and *rowIndex*.

`setColumnAutoresizingStyle` (page 1469)
Sets the column autoresizing style of the receiver to style.

`columnAutoresizingStyle` (page 1451)
Returns the receiver's column autoresizing style.

`sizeLastColumnToFit` (page 1474)
Resizes the last column if there's room so the receiver fits exactly within its enclosing NSClipView.

`noteNumberOfRowsChanged` (page 1460)
Informs the receiver that the number of records in its data source has changed, allowing the receiver to update the scrollers in its NSScrollView without actually reloading data into the receiver.

`noteHeightOfRowsWithIndexesChanged` (page 1460)
Informs the receiver that the rows specified in *indexSet* have changed height.

`tile` (page 1476)
Properly sizes the receiver and its header view and marks it as needing display.

`sizeToFit` (page 1474)
Changes the width of columns in the receiver so all columns are visible.

Drawing

`drawRow` (page 1456)
Draws the cells for the row at *rowIndex* in the columns that intersect *clipRect*.

`drawGridInClipRect` (page 1455)
Draws the grid lines within *aRect*, using the grid color set with `setGridColor` (page 1472).

`highlightSelectionInClipRect` (page 1458)
Highlights the region of the receiver in *clipRect*.

`drawBackgroundInClipRect` (page 1455)
Draws the background in the clip rect specified by *clipRect*.

Scrolling

`scrollRowToVisible` (page 1463)
Scrolls the receiver vertically in an enclosing NSClipView so the row specified by *rowIndex* is visible.

`scrollColumnToVisible` (page 1463)
Scrolls the receiver and header view horizontally in an enclosing NSClipView so the column specified by *columnIndex* is visible.

Text Delegate Methods

textShouldBeginEditing (page 1476)
Queries the delegate using controlTextShouldBeginEditing (page 465), returning the delegate's response, or simply returning true to allow editing of *textObject* if the delegate doesn't respond to that method.

textDidBeginEditing (page 1475)
Posts a ControlTextDidBeginEditingNotification (page 466) to the default notification center, as described in the NSControl class specification.

textDidChange (page 1475)
Sends textDidChange (page 1475) to the edited cell and posts a ControlTextDidChangeNotification (page 467) to the default notification center, as described in the NSControl class specification.

textShouldEndEditing (page 1476)
Validates the *textObject* cell being edited and queries the delegate using controlTextShouldEndEditing (page 465), returning the delegate's response if it responds to that method.

textDidEndEditing (page 1476)
Updates the data source based on the newly edited value and selects another cell for editing if possible according to the character that ended editing (Return, Tab, Backtab).

Persistence

autosaveName (page 1449)
Returns the name under which table information is automatically saved.

autosaveTableColumns (page 1450)
Returns whether the order and width of the receiver's columns are automatically saved.

setAutosaveName (page 1469)
Sets the name under which table information is automatically saved to *name*.

setAutosaveTableColumns (page 1469)
Sets whether the order and width of this table view's columns are automatically saved.

Setting the Delegate

setDelegate (page 1470)
Sets the receiver's delegate to *anObject*.

delegate (page 1453)
Returns the receiver's delegate.

Setting the Indicator Image

indicatorImage (page 1458)
Returns the indicator image of *aTableColumn*.

setIndicatorImage (page 1473)
Sets the indicator image of *aTableColumn* to *anImage*.

Supporting Highlightable Column Headers

highlightedTableColumn (page 1458)
Returns the table column highlighted in the receiver.

setHighlightedTableColumn (page 1472)
Sets *aTableColumn* to be the currently highlighted column header.

Dragging

dragImageForRowsWithIndexes (page 1455)
Computes and returns an image to use for dragging.

canDragRowsWithIndexes (page 1450)
Returns whether the receiver allows dragging the rows at *rowIndexes* with a drag initiated at *mousedDownPoint*.

setDraggingSourceOperationMask (page 1471)
Sets the default operation mask returned by `draggingSourceOperationMaskForLocal:` to *mask*

setDropRowAndDropOperation (page 1471)
Used if you wish to “retarget” the proposed drop.

setVerticalMotionCanBeginDrag (page 1474)
Sets whether vertical motion is treated as a drag or selection change to *flag*.

verticalMotionCanBeginDrag (page 1477)
Returns whether vertical motion is treated as a drag or selection change.

Sorting

setSortDescriptors (page 1473)
Sets the receiver’s sort descriptors to the NSSortDescriptor objects in *array*.

sortDescriptors (page 1475)
Returns the receiver’s sort descriptors.

Deprecated Methods

dragImageForRows (page 1454)
Computes and returns an image to use for dragging.

setAutoresizesAllColumnsToFit (page 1468)
Controls whether the receiver proportionally resizes its columns to fit when its superview’s frame changes.

autoresizesAllColumnsToFit (page 1449)
Returns `true` if the receiver proportionally resizes its columns to fit when its superview’s frame changes, `false` if it only resizes the last column.

Moving and resizing columns

`tableViewDidDragTableColumn` (page 1479) *delegate method*
Sent at the time the mouse button goes up in *`tableView`* and *`tableColumn`* has been dragged during the time the mouse button was down.

`tableViewColumnDidMove` (page 1479) *delegate method*
Informs the delegate that a column was moved by user action in the NSTableView.

`tableViewColumnDidResize` (page 1479) *delegate method*
Informs the delegate that a column was resized in the NSTableView.

Selecting in table

`selectionShouldChangeInTableView` (page 1478) *delegate method*
Returns `true` to permit *`aTableView`* to change its selection (typically a row being edited), `false` to deny permission.

`tableViewShouldSelectRow` (page 1480) *delegate method*
Returns `true` to permit *`aTableView`* to select the row at *`rowIndex`*, `false` to deny permission.

`tableViewShouldSelectTableColumn` (page 1481) *delegate method*
Returns `true` to permit *`aTableView`* to select *`aTableColumn`*, `false` to deny permission.

`tableViewSelectionIsChanging` (page 1480) *delegate method*
Informs the delegate that the NSTableView's selection is in the process of changing (typically because the user is dragging the cursor across a number of rows).

`tableViewSelectionDidChange` (page 1480) *delegate method*
Informs the delegate that the NSTableView's selection has changed.

Responding to mouse events

`tableViewDidClickTableColumn` (page 1479) *delegate method*
Sent at the time the mouse button subsequently goes up in *`tableView`* and *`tableColumn`* has been "clicked" without having been dragged anywhere.

`tableViewMouseDownInHeaderOfTableColumn` (page 1480) *delegate method*
Sent to the delegate whenever the mouse button is clicked in *`tableView`* while the cursor is in a column header *`tableColumn`*.

Editing a cell

`tableViewShouldEditLocation` (page 1480) *delegate method*
Returns `true` to permit *`aTableView`* to edit the cell at *`rowIndex`* in *`aTableColumn`*, `false` to deny permission.

Displaying a cell

`tableViewWillDisplayCell` (page 1481) *delegate method*
Informs the delegate that *aTableView* will display the cell at *rowIndex* in *aTableColumn* using *aCell*.

Displaying tooltips

`tableViewToolTipForCell` (page 1481) *delegate method*
Returns a string that is displayed as a tooltip for *aCell* in *aTableColumn* of *aTableView*.

Allowing variable height rows

`tableViewHeightOfRow` (page 1479) *delegate method*
Returns the height of *row* in *tableView*.

Constructors

NSTableView

Creates an NSTableView with a zero-sized frame rectangle.

```
public NSTableView()
```

Creates a new NSTableView with *frameRect* as its frame rectangle.

```
public NSTableView(NSRect frameRect)
```

Discussion
In both constructors, the new NSTableView has a header view but has no columns; you can create NSTableColumn objects, set their titles and attributes, and add them to the new NSTableView with `addTableColumn` (page 1447). You must also set the NSTableView up in an NSScrollView with NSScrollView's `setDocumentView` (page 1279) method.

It's usually more convenient to create an NSTableView using Interface Builder. Interface Builder lets you create an NSTableView already embedded in an NSScrollView, add and name the columns, and set up a data source.

Instance Methods

addTableColumn

Adds *aColumn* as the last column of the receiver.

```
public void addTableColumn(NSTableColumn aColumn)
```

Discussion
The column is retained by the receiver.

See Also
`sizeLastColumnToFit` (page 1474)
`removeTableColumn` (page 1462)

allowsColumnReordering

Returns `true` if the receiver allows the user to rearrange columns by dragging their headers, `false` otherwise.

```
public boolean allowsColumnReordering()
```

Discussion
The default is `true`. You can rearrange columns programmatically regardless of this setting.

See Also
`moveColumnToColumn` (page 1459)
`setAllowsColumnReordering` (page 1467)

allowsColumnResizing

Returns `true` if the receiver allows the user to resize columns by dragging between their headers, `false` otherwise.

```
public boolean allowsColumnResizing()
```

Discussion
The default is `true`. You can resize columns programmatically regardless of this setting.

See Also
`setWidth` (page 1429) (NSTableColumn)
`setAllowsColumnResizing` (page 1467)

allowsColumnSelection

Returns `true` if the receiver allows the user to select columns by clicking their headers, `false` otherwise.

```
public boolean allowsColumnSelection()
```

Discussion
The default is `true`. You can select columns programmatically regardless of this setting.

See Also
`selectColumn` (page 1464)
`allowsColumnReordering` (page 1448)
`setAllowsColumnSelection` (page 1467)

allowsEmptySelection

Returns true if the receiver allows the user to select zero columns or rows, false otherwise.

```
public boolean allowsEmptySelection()
```

Discussion

The default is true.

You cannot set an empty selection programmatically if this setting is false, unlike with the other settings that affect selection behavior.

See Also

deselectAll (page 1453)
deselectColumn (page 1453)
deselectRow (page 1454)
setAllowsEmptySelection (page 1467)

allowsMultipleSelection

Returns true if the receiver allows the user to select more than one column or row at a time, false otherwise.

```
public boolean allowsMultipleSelection()
```

Discussion

The default is false. You can select multiple columns or rows programmatically regardless of this setting.

See Also

selectColumn (page 1464)
selectRow (page 1466)
setAllowsMultipleSelection (page 1468)

autoresizesAllColumnsToFit

Returns true if the receiver proportionally resizes its columns to fit when its superview's frame changes, false if it only resizes the last column.

```
public boolean autoresizesAllColumnsToFit()
```

Discussion

This method is deprecated. You should use columnAutoresizingStyle (page 1451) instead.

See Also

columnAutoresizingStyle (page 1451)
setColumnAutoresizingStyle (page 1469)

autosaveName

Returns the name under which table information is automatically saved.

```
public String autosaveName()
```

Discussion

If no name has been set, this method returns `null`. The table information is saved separately for each user and for each application that user uses.

Note that even when a table view has an autosave name, it may not be saving table information automatically. To check whether table information is being saved automatically, use `autosaveTableColumns` (page 1450).

See Also

`autosaveTableColumns` (page 1450)

`setAutosaveName` (page 1469)

autosaveTableColumns

Returns whether the order and width of the receiver's columns are automatically saved.

```
public boolean autosaveTableColumns()
```

Discussion

The table information is saved separately for each user and for each application that user uses. Note that if `autosaveName` (page 1449) returns `null`, this setting is ignored and table information isn't saved.

See Also

`autosaveName` (page 1449)

`setAutosaveTableColumns` (page 1469)

`setAutosaveName` (page 1469)

backgroundColor

Returns the color used to draw the background of the receiver.

```
public NSColor backgroundColor()
```

Discussion

The default background color is light gray.

See Also

`setBackgroundColor` (page 1469)

canDragRowsWithIndexes

Returns whether the receiver allows dragging the rows at *rowIndexes* with a drag initiated at *mousedDownPoint*.

```
public boolean canDragRowsWithIndexes(NSIndexSet rowIndexes, NSPoint mouseDownPoint)
```

Discussion

Return `false` to disallow the drag.

Availability

Available in Mac OS X v10.4 and later.

clickedColumn

Returns the index of the column the user clicked to trigger an action message.

```
public int clickedColumn()
```

Discussion

The return value of this method is meaningful only in the target's implementation of the action or double-action method. Returns -1 if the user clicked in an area of the table view not occupied by columns.

See Also

`clickedRow` (page 1451)

`setAction` (page 455) (NSControl)

`setDoubleAction` (page 1470)

clickedRow

Returns the index of the row the user clicked to trigger an action message.

```
public int clickedRow()
```

Discussion

The return value of this method is meaningful only in the target's implementation of the action or double-action method. Returns -1 if the user clicked in an area of the table view not occupied by table rows.

See Also

`clickedColumn` (page 1451)

`setAction` (page 455) (NSControl)

`setDoubleAction` (page 1470)

columnAtPoint

Returns the index of the column *aPoint* lies in, or –1 if *aPoint* lies outside the receiver's bounds.

```
public int columnAtPoint(NSPoint aPoint)
```

Discussion

aPoint is in the coordinate system of the receiver.

See Also

`rowAtPoint` (page 1462)

columnAutoresizingStyle

Returns the receiver's column autoresizing style.

```
public int columnAutoresizingStyle()
```

Availability

Available in Mac OS X v10.4 and later.

See Also
`setColumnAutoresizingStyle` (page 1469)

columnsInRect

Returns a range of indices for the receiver's columns that lie wholly or partially within the horizontal boundaries of *aRect*.

```
public NSRange columnsInRect(NSRect aRect)
```

Discussion
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` (page 1452) returns an NSRange whose length is 0.

See Also
`rowsInRect` (page 1462)

columnWithIdentifier

Returns the index of the first column in the receiver whose identifier is equal to *anObject*, when compared using `equals`, or –1 if no columns are found with the specified identifier.

```
public int columnWithIdentifier(Object anObject)
```

See Also
`tableColumnWithIdentifier` (page 1475)

cornerView

Returns the NSView used to draw the area to the right of the column headers and above the vertical scroller of the enclosing NSScrollView.

```
public NSView cornerView()
```

Discussion
This is by default a simple view that merely fills in its frame, but you can replace it with a custom view using `setCornerView` (page 1470).

See Also
`headerView` (page 1458)

dataSource

Returns the object that provides the data displayed by the receiver.

```
public Object dataSource()
```

Discussion

See "Using a Table Data Source" and the NSTableView.DataSource (page 2019) `interface` specification for more information.

See Also

`setDataSource` (page 1470)

delegate

Returns the receiver's delegate.

```
public Object delegate()
```

See Also

`setDelegate` (page 1470)

deselectAll

Deselects all selected rows or columns if empty selection is allowed; otherwise does nothing.

```
public void deselectAll(Object sender)
```

Discussion

Posts `TableViewSelectionDidChangeNotification` (page 1482) 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` (page 1478).

See Also

`allowsEmptySelection` (page 1449)

`selectAll` (page 1463)

`selectColumn` (page 1464)

deselectColumn

Deselects the column at *`columnIndex`* if it's selected, regardless of whether empty selection is allowed.

```
public void deselectColumn(int columnIndex)
```

Discussion

If the selection does in fact change, posts `TableViewSelectionDidChangeNotification` (page 1482) 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

`selectedColumn` (page 1464)

`allowsEmptySelection` (page 1449)
`selectRow` (page 1466)

deselectRow

Deselects the row at *rowIndex* if it's selected, regardless of whether empty selection is allowed.

```
public void deselectRow(int rowIndex)
```

Discussion
If the selection does in fact change, posts `TableViewSelectionDidChangeNotification` (page 1482) 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
`selectedRow` (page 1465)
`allowsEmptySelection` (page 1449)

doubleAction

Returns the message sent to the target when the user double-clicks a column header or an uneditable cell.

```
public NSSelector doubleAction()
```

See Also
`action` (page 448) (NSControl)
`target` (page 463) (NSControl)
`setDoubleAction` (page 1470)

dragImageForRows

Computes and returns an image to use for dragging.

```
public NSImage dragImageForRows(NSArray dragRows, NSEvent dragEvent, NSMutablePoint
     dragImageOffset)
```

Discussion
Override this to return a custom image. *dragRows* represents the rows participating in the drag. *dragEvent* is a reference to the mouse-down event that began the drag. *dragImageOffset* is an in/out parameter.

This method is called with *dragImageOffset* set to `NSPoint.ZeroPoint`, but it can be modified to reposition the returned image. A *dragImageOffset* of `NSPoint.ZeroPoint` will cause the image to be centered under the cursor.

This method is deprecated. You should use `dragImageForRowsWithIndexes` (page 1455) instead.

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

dragImageForRowsWithIndexes

Computes and returns an image to use for dragging.

```
public NSImage dragImageForRowsWithIndexes(NSIndexSet dragRows, NSArray tableColumns,
     NSEvent dragEvent, NSMutablePoint dragImageOffset)
```

Discussion
Override this to return a custom image. *dragRows* represents the rows participating in the drag. *tableColumns* represents the table columsn that should be in the output image. *dragEvent* is a reference to the mouse-down event that began the drag. *dragImageOffset* is an in/out parameter.

This method is called with *dragImageOffset* set to `NSPoint.ZeroPoint`, but it can be modified to reposition the returned image. A *dragImageOffset* of `NSPoint.ZeroPoint` will cause the image to be centered under the cursor.

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

drawBackgroundInClipRect

Draws the background in the clip rect specified by *clipRect*.

```
public void drawBackgroundInClipRect(NSRect clipRect)
```

Availability
Available in Mac OS X v10.3 and later

drawGridInClipRect

Draws the grid lines within *aRect*, using the grid color set with `setGridColor` (page 1472).

```
public void drawGridInClipRect(NSRect aRect)
```

Discussion
This method draws a grid regardless of whether the receiver is set to draw one automatically.

Subclasses can override this method to draw grid lines other than the standard ones.

See Also
`gridColor` (page 1457)
`setIntercellSpacing` (page 1473)
`drawsGrid` (page 1456)
`drawRow` (page 1456)
`highlightSelectionInClipRect` (page 1458)

drawRow

Draws the cells for the row at *rowIndex* in the columns that intersect *clipRect*.

```
public void drawRow(int rowIndex, NSRect clipRect)
```

Discussion

Sends `tableViewWillDisplayCell` (page 1481) to the delegate before drawing each cell.

Subclasses can override this method to customize their appearance.

See Also

`columnsInRect` (page 1452)
`highlightSelectionInClipRect` (page 1458)
`drawGridInClipRect` (page 1455)

drawsGrid

```
public boolean drawsGrid()
```

Discussion

This method has been deprecated. Use `gridStyleMask` (page 1457) instead.

Availability

Deprecated in Mac OS X v10.3.

editedColumn

If sent during `editLocation` (page 1456), returns the index of the column being edited; otherwise returns –1.

```
public int editedColumn()
```

editedRow

If sent during `editLocation` (page 1456), returns the index of the row being edited; otherwise returns –1.

```
public int editedRow()
```

editLocation

Edits the cell at *columnIndex* and *rowIndex*, selecting its entire contents if *flag* is `true`.

```
public void editLocation(int columnIndex, int rowIndex, NSEvent theEvent, boolean
     flag)
```

Discussion

This method is invoked automatically in response to user actions; you should rarely need to invoke it directly. *theEvent* is usually the mouse event that triggered editing; it can be `null` when starting an edit programmatically.

This method scrolls the receiver so that the cell is visible, sets up the field editor, and sends `selectAndEditWithFrameInView` (page 319) and `editWithFrameInView` (page 310) to the field editor's NSCell object with the NSTableView as the text delegate.

The row at *`rowIndex`* must be selected prior to calling `editLocation`, or an exception will be thrown.

See Also

`editedColumn` (page 1456)

`editedRow` (page 1456)

frameOfCellAtLocation

Returns a rectangle locating the cell that lies at the intersection of *`columnIndex`* and *`rowIndex`*.

```
public NSRect frameOfCellAtLocation(int columnIndex, int rowIndex)
```

Discussion

Returns `NSRect.ZeroRect` if *`columnIndex`* or *`rowIndex`* is greater than the number of columns or rows in the NSTableView.

The result of this method is used in a `drawWithFrameInView` (page 310) message to the NSTableColumn's data cell.

See Also

`rectOfColumn` (page 1461)

`rectOfRow` (page 1461)

gridColor

Returns the color used to draw grid lines.

```
public NSColor gridColor()
```

Discussion

The default color is gray.

See Also

`drawsGrid` (page 1456)

`drawGridInClipRect` (page 1455)

`setGridColor` (page 1472)

gridStyleMask

Returns the receiver's grid style mask.

```
public int gridStyleMask()
```

Discussion

Possible return values are described in "Constants" (page 1477).

Availability

Available in Mac OS X v10.3 and later.

See Also

`setGridStyleMask` (page 1472)

headerView

Returns the NSTableHeaderView used to draw headers over columns, or `null` if the receiver has no header view.

```
public NSTableHeaderView headerView()
```

Discussion

See "The Parts of a Table" and the NSTableHeaderView (page 1433) class specification for more information.

See Also

`setHeaderView` (page 1472)

highlightedTableColumn

Returns the table column highlighted in the receiver.

```
public NSTableColumn highlightedTableColumn()
```

Discussion

A highlightable column header can be used in conjunction with row selection to highlight a particular column of the table. An example of this is how Mail indicates the currently sorted column.

See Also

`setHighlightedTableColumn` (page 1472)

highlightSelectionInClipRect

Highlights the region of the receiver in *clipRect*.

```
public void highlightSelectionInClipRect(NSRect clipRect)
```

Discussion

This method is invoked before `drawRow` (page 1456).

Subclasses can override this method to change the manner in which they highlight selections.

See Also

`drawGridInClipRect` (page 1455)

indicatorImage

Returns the indicator image of *aTableColumn*.

```
public NSImage indicatorImage(NSTableColumn aTableColumn)
```

Discussion
An indicator image is an arbitrary (small) image that is rendered on the right side of the column header. An example of its use is in Mail to indicate the sorting direction of the currently sorted column in a mailbox.

See Also
`setIndicatorImage` (page 1473)

intercellSpacing

Returns the horizontal and vertical spacing between cells.

```
public NSSize intercellSpacing()
```

Discussion
The default spacing is (3.0, 2.0).

See Also
`setDrawsGrid` (page 1471)
`setIntercellSpacing` (page 1473)

isColumnSelected

Returns `true` if the column at *`columnIndex`* is selected, `false` otherwise.

```
public boolean isColumnSelected(int columnIndex)
```

See Also
`selectedColumn` (page 1464)
`selectedColumnEnumerator` (page 1465)
`selectColumn` (page 1464)

isRowSelected

Returns `true` if the row at *`rowIndex`* is selected, `false` otherwise.

```
public boolean isRowSelected(int rowIndex)
```

See Also
`selectedRow` (page 1465)
`selectedRowEnumerator` (page 1465)
`selectRow` (page 1466)

moveColumnToColumn

Moves the column and heading at *`columnIndex`* to *`newIndex`*.

```
public void moveColumnToColumn(int columnIndex, int newIndex)
```

Discussion

This method posts `TableViewColumnDidMoveNotification` (page 1482) to the default notification center.

noteHeightOfRowsWithIndexesChanged

Informs the receiver that the rows specified in *indexSet* have changed height.

```
public void noteHeightOfRowsWithIndexesChanged(NSIndexSet indexSet)
```

Discussion

If the delegate implements `tableViewHeightOfRow` (page 1479) this method immediately re-tiles the tableview using the row heights the delegate provides.

Availability

Available in Mac OS X v10.4 and later.

noteNumberOfRowsChanged

Informs the receiver that the number of records in its data source has changed, allowing the receiver to update the scrollers in its NSScrollView without actually reloading data into the receiver.

```
public void noteNumberOfRowsChanged()
```

Discussion

It's useful for a data source that continually receives data in the background over a period of time, in which case the NSTableView can remain responsive to the user while the data is received.

See the NSTableView.DataSource (page 2019) interface specification for information on the messages an NSTableView sends to its data source.

See Also

`reloadData` (page 1462)

`numberOfRowsInTableView` (page 2020) (NSTableView.DataSource interface)

numberOfColumns

Returns the number of columns in the receiver.

```
public int numberOfColumns()
```

See Also

`numberOfRows` (page 1460)

numberOfRows

Returns the number of rows in the receiver.

```
public int numberOfRows()
```

See Also

`numberOfColumns` (page 1460)

numberOfRowsInTableView (page 2020) (NSTableView.DataSource interface)

numberOfSelectedColumns

Returns the number of selected columns.

```
public int numberOfSelectedColumns()
```

See Also

numberOfSelectedRows (page 1461)
selectedColumnEnumerator (page 1465)

numberOfSelectedRows

Returns the number of selected rows.

```
public int numberOfSelectedRows()
```

See Also

numberOfSelectedColumns (page 1461)
selectedRowEnumerator (page 1465)

rectOfColumn

Returns the rectangle containing the column at *columnIndex*.

```
public NSRect rectOfColumn(int columnIndex)
```

Discussion

Returns NSRect.ZeroRect if *columnIndex* lies outside the range of valid column indices for the receiver.

See Also

frameOfCellAtLocation (page 1457)
rectOfRow (page 1461)
headerRectOfColumn (page 1435) (NSTableHeaderView)

rectOfRow

Returns the rectangle containing the row at *rowIndex*.

```
public NSRect rectOfRow(int rowIndex)
```

Discussion

Returns NSRect.ZeroRect if *rowIndex* lies outside the range of valid row indices for the receiver.

See Also

frameOfCellAtLocation (page 1457)
rectOfColumn (page 1461)

reloadData

Marks the receiver as needing redisplay, so it will reload the data for visible cells and draw the new values.

```
public void reloadData()
```

See Also
`noteNumberOfRowsChanged` (page 1460)

removeTableColumn

Removes *aTableColumn* from the receiver.

```
public void removeTableColumn(NSTableColumn aTableColumn)
```

See Also
`sizeLastColumnToFit` (page 1474)
`addTableColumn` (page 1447)

rowAtPoint

Returns the index of the row *aPoint* lies in, or –1 if *aPoint* lies outside the receiver's bounds.

```
public int rowAtPoint(NSPoint aPoint)
```

Discussion
aPoint is in the coordinate system of the receiver.

See Also
`columnAtPoint` (page 1451)

rowHeight

Returns the height of each row in the receiver.

```
public float rowHeight()
```

Discussion
The default row height is 16.0.

See Also
`setRowHeight` (page 1473)

rowsInRect

Returns a range of indices for the rows that lie wholly or partially within the vertical boundaries of *aRect*.

```
public NSRange rowsInRect(NSRect aRect)
```

Discussion

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 this method returns an NSRange whose length is 0.

See Also

columnsInRect (page 1452)

scrollColumnToVisible

Scrolls the receiver and header view horizontally in an enclosing NSClipView so the column specified by *columnIndex* is visible.

```
public void scrollColumnToVisible(int columnIndex)
```

See Also

scrollRowToVisible (page 1463)

scrollToPoint (page 347) (NSClipView)

scrollRowToVisible

Scrolls the receiver vertically in an enclosing NSClipView so the row specified by *rowIndex* is visible.

```
public void scrollRowToVisible(int rowIndex)
```

See Also

scrollColumnToVisible (page 1463)

scrollToPoint (page 347) (NSClipView)

selectAll

```
public void selectAll(Object sender)
```

Discussion

If the table allows multiple selection, this action method selects all rows or all columns, according to whether rows or columns were most recently selected. If nothing has been recently selected, this method selects all rows. If this table doesn't allow multiple selection, this method does nothing.

If the selection does change, this method posts TableViewSelectionDidChangeNotification (page 1482) to the default notification center.

As a target-action method, selectAll checks with the delegate before changing the selection.

See Also

allowsMultipleSelection (page 1449)

deselectAll (page 1453)

selectColumn (page 1464)

selectColumn

This method has been deprecated. Use `selectColumnIndexes` (page 1464) instead.

```
public void selectColumn(int columnIndex, boolean flag)
```

Availability
Deprecated in Mac OS X v10.3.

See Also
`allowsMultipleSelection` (page 1449)
`allowsColumnSelection` (page 1448)
`deselectColumn` (page 1453)
`selectedColumn` (page 1464)
`selectRow` (page 1466)

selectColumnIndexes

Sets the column selection using *indexes*.

```
public void selectColumnIndexes(NSIndexSet indexes, boolean extend)
```

Discussion
If the *extend* flag is `false` the selected columns are specified by *indexes*. If *extend* is `true`, the columns indicated by *indexes* are added to the collection of already selected columns, providing multiple selection.

If a subclass implements only the deprecated `selectColumn` (page 1464) method, then this method will be invoked in a loop. If a subclass implements this method, then `selectColumn` is not used. This allows subclasses that already implement `selectColumn` to still receive all selection messages. To avoid cycles, implementations of this method and `selectColumn` should not invoke each other.

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

See Also
`selectRowIndexes` (page 1466)

selectedColumn

Returns the index of the last column selected or added to the selection, or –1 if no column is selected.

```
public int selectedColumn()
```

See Also
`selectedColumnEnumerator` (page 1465)
`numberOfSelectedColumns` (page 1461)
`selectColumn` (page 1464)
`deselectColumn` (page 1453)

selectedColumnEnumerator

This method has been deprecated. Use selectedColumnIndexes (page 1465) instead.

```
public NSEnumerator selectedColumnEnumerator()
```

Availability
Deprecated in Mac OS X v10.3.

See Also
numberOfSelectedColumns (page 1461)
selectedColumn (page 1464)
selectedRowEnumerator (page 1465)

selectedColumnIndexes

Returns the selected columns.

```
public NSIndexSet selectedColumnIndexes()
```

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

See Also
selectedRowIndexes (page 1466)
selectColumnIndexes (page 1464)

selectedRow

Returns the index of the last row selected or added to the selection, or –1 if no row is selected.

```
public int selectedRow()
```

See Also
selectedRowEnumerator (page 1465)
numberOfSelectedRows (page 1461)
selectRow (page 1466)
deselectRow (page 1454)

selectedRowEnumerator

This method has been deprecated. Use selectedRowIndexes (page 1466) instead.

```
public NSEnumerator selectedRowEnumerator()
```

Availability
Deprecated in Mac OS X v10.3.

See Also
numberOfSelectedRows (page 1461)

`selectedRow` (page 1465)
`selectedColumnEnumerator` (page 1465)

selectedRowIndexes

Returns the selected rows.

```
public NSIndexSet selectedRowIndexes()
```

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

See Also
`selectedColumnIndexes` (page 1465)
`selectRowIndexes` (page 1466)

selectRow

This method has been deprecated. Use `selectRowIndexes` (page 1466) instead.

```
public void selectRow(int rowIndex, boolean flag)
```

Availability
Deprecated in Mac OS X v10.3.

See Also
`allowsMultipleSelection` (page 1449)
`deselectRow` (page 1454)
`selectedRow` (page 1465)
`selectColumn` (page 1464)

selectRowIndexes

Sets the row selection using *`indexes`*.

```
public void selectRowIndexes(NSIndexSet indexes, boolean extend)
```

Discussion
If the *`extend`* flag is `false` the selected rows are specified by *`indexes`*. If *`extend`* is `true`, the rows indicated by *`indexes`* are added to the collection of already selected rows, providing multiple selection.

If a subclass implements only the deprecated `selectRow` (page 1466) method, then that method will be invoked in a loop. This allows subclasses that already implement `selectRow` to still receive all selection messages. If a subclass implements `selectRowIndexes`, then `selectRow` is not used. Note that to avoid cycles, implementations of this method and `selectRow` should not invoke each other.

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

See Also

selectColumnIndexes (page 1464)

setAllowsColumnReordering

Controls whether the user can drag column headers to reorder columns.

```
public void setAllowsColumnReordering(boolean flag)
```

Discussion

If *flag* is true the user can reorder columns; if *flag* is false the user can't. The default is true. You can rearrange columns programmatically regardless of this setting.

See Also

moveColumnToColumn (page 1459)

allowsColumnReordering (page 1448)

setAllowsColumnResizing

Controls whether the user can resize columns by dragging between headers.

```
public void setAllowsColumnResizing(boolean flag)
```

Discussion

If *flag* is true the user can resize columns; if *flag* is false the user can't. The default is true. You can resize columns programmatically regardless of this setting.

See Also

setWidth (page 1429) (NSTableColumn)

allowsColumnResizing (page 1448)

setAllowsColumnSelection

Controls whether the user can select an entire column by clicking its header.

```
public void setAllowsColumnSelection(boolean flag)
```

Discussion

If *flag* is true the user can select columns; if *flag* is false the user can't. The default is true. You can select columns programmatically regardless of this setting.

See Also

selectColumn (page 1464)

setAllowsColumnReordering (page 1467)

allowsColumnSelection (page 1448)

setAllowsEmptySelection

Controls whether the receiver allows zero rows or columns to be selected.

```
public void setAllowsEmptySelection(boolean flag)
```

Discussion

If *flag* is true empty selection is allowed; if *flag* is false it isn't. The default is true.

You cannot set an empty selection programmatically if empty selection is disallowed, unlike with the other settings that affect selection behavior.

See Also

deselectAll (page 1453)

deselectColumn (page 1453)

deselectRow (page 1454)

allowsEmptySelection (page 1449)

setAllowsMultipleSelection

Controls whether the user can select more than one row or column at a time.

```
public void setAllowsMultipleSelection(boolean flag)
```

Discussion

If *flag* is true the user can select multiple rows or columns; if *flag* is false the user can't. The default is false. You can select multiple columns or rows programmatically regardless of this setting.

See Also

selectColumn (page 1464)

selectRow (page 1466)

allowsMultipleSelection (page 1449)

setAutoresizesAllColumnsToFit

Controls whether the receiver proportionally resizes its columns to fit when its superview's frame changes.

```
public void setAutoresizesAllColumnsToFit(boolean flag)
```

Discussion

If *flag* is true, the difference in width is distributed among the receiver's table columns; if *flag* is false, only the last column is resized to fit.

This method is deprecated. You should use setColumnAutoresizingStyle (page 1469) instead. To preserve compatibility this method sets the autoresizing style to NSTableViewUniformColumnAutoresizingStyle , if *flag* is true. Otherwise the autoresizing style is set to NSTableViewLastColumnOnlyAutoresizingStyle.

Availability

Deprecated in Mac OS X v10.4 and later.

See Also

setColumnAutoresizingStyle (page 1469)

setAutosaveName

Sets the name under which table information is automatically saved to *name*.

```
public void setAutosaveName(String name)
```

Discussion

If *name* is different from the current name, this method also reads in the saved information and sets the order and width of this table view's columns to match.

The table information is saved separately for each user and for each application that user uses. Note that even though a table view has an autosave name, it may not be saving table information automatically. To set whether table information is being saved automatically, use `setAutosaveTableColumns` (page 1469).

See Also

`autosaveName` (page 1449)

`setAutosaveTableColumns` (page 1469)

setAutosaveTableColumns

Sets whether the order and width of this table view's columns are automatically saved.

```
public void setAutosaveTableColumns(boolean flag)
```

Discussion

If *flag* is different from the current value, this method also reads in the saved information and sets the table options to match.

The table information is saved separately for each user and for each application that user uses. Note that if `autosaveName` (page 1449) returns `null`, this setting is ignored and table information isn't saved.

See Also

`autosaveTableColumns` (page 1450)

`setAutosaveName` (page 1469)

setBackgroundColor

Sets the receiver's background color to *aColor*.

```
public void setBackgroundColor(NSColor aColor)
```

See Also

`setNeedsDisplay` (page 1779) (NSView)

`backgroundColor` (page 1450)

setColumnAutoresizingStyle

Sets the column autoresizing style of the receiver to style.

```
public void setColumnAutoresizingStyle(int style)
```

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

See Also
`columnAutoresizingStyle` (page 1451)

setCornerView

Sets the receiver's corner view to *aView*.

```
public void setCornerView(NSView aView)
```

Discussion
The default corner view merely draws a bezeled rectangle using a blank NSTableHeaderCell, 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 receiver's header view.

See Also
`setHeaderView` (page 1472)
`cornerView` (page 1452)

setDataSource

Sets the receiver's data source to *anObject* and invokes `tile` (page 1476).

```
public void setDataSource(Object anObject)
```

Discussion
anObject should implement the appropriate methods of the NSTableView.DataSource (page 2019) interface.

This method throws an `InternalInconsistencyException` if *anObject* doesn't respond to either `numberOfRowsInTableView` (page 2020) or `tableViewObjectValueForLocation` (page 2021).

See Also
`dataSource` (page 1452)

setDelegate

Sets the receiver's delegate to *anObject*.

```
public void setDelegate(Object anObject)
```

See Also
`delegate` (page 1453)

setDoubleAction

Sets the message sent to the target when the user double-clicks an uneditable cell or a column header to *aSelector*.

```
public void setDoubleAction(NSSelector aSelector)
```

Discussion

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.

For the method to have any effect, the receiver's action and target must be set to the class in which the selector is declared. See *Action Messages* for additional information on action messages.

See Also

`setAction` (page 455) (NSControl)

`setTarget` (page 460) (NSControl)

`doubleAction` (page 1454)

setDraggingSourceOperationMask

Sets the default operation mask returned by `draggingSourceOperationMaskForLocal:` to *mask*

```
public void setDraggingSourceOperationMask(int mask, boolean isLocal)
```

Discussion

If *isLocal* is `true` then *mask* applies when the destination object is in the same application. If *isLocal* is `false` then *mask* applies when the destination object in an application outside the receiver's application. NSTableView will archive the operation mask you set for each *isLocal* setting.

Availability

Available in Mac OS X v10.4 and later.

setDrawsGrid

```
public void setDrawsGrid(boolean flag)
```

Discussion

This method has been deprecated. Use `setGridStyleMask` (page 1472) instead.

Availability

Deprecated in Mac OS X v10.3.

setDropRowAndDropOperation

Used if you wish to "retarget" the proposed drop.

```
public void setDropRowAndDropOperation(int row, int operation)
```

Discussion

To specify a drop on the second row, one would specify *row* as 2, and *operation* as `DropOn`. To specify a drop below the last row, one would specify *row* as `[tv.numberOfRows()]` and *operation* as `DropAbove`.

setGridColor

Sets the color used to draw grid lines to *aColor*.

```
public void setGridColor(NSColor aColor)
```

Discussion

The default color is gray.

See Also

setDrawsGrid (page 1471)

drawGridInClipRect (page 1455)

gridColor (page 1457)

setGridStyleMask

Sets the grid style mask to specify if no grid lines, vertical grid lines, or horizontal grid lines should be displayed.

```
public void setGridStyleMask(int gridType)
```

Discussion

Possible values for *gridType* are described in "Constants" (page 1477).

Availability

Available in Mac OS X v10.3 and later.

See Also

gridStyleMask (page 1457)

setHeaderView

Sets the receiver's header view to *aHeaderView*.

```
public void setHeaderView(NSTableHeaderView aHeaderView)
```

Discussion

If *aHeaderView* is `null`, the current header view is removed.

See Also

setCornerView (page 1470)

headerView (page 1458)

setHighlightedTableColumn

Sets *aTableColumn* to be the currently highlighted column header.

```
public void setHighlightedTableColumn(NSTableColumn aTableColumn)
```

See Also

highlightedTableColumn (page 1458)

setIndicatorImage

Sets the indicator image of *aTableColumn* to *anImage*.

```
public void setIndicatorImage(NSImage anImage, NSTableColumn aTableColumn)
```

See Also

indicatorImage (page 1458)

setIntercellSpacing

Sets the width and height between cells to those in *aSize* and redisplays the receiver.

```
public void setIntercellSpacing(NSSize aSize)
```

Discussion

The default intercell spacing is (3.0, 2.0).

Table views normally have a 1 pixel separation between consecutively selected rows or columns. An intercell spacing of (1.0, 1.0) or greater is required if you want this separation. An intercell spacing of (0.0, 0.0) will force there to be no separation between consecutive selections.

See Also

intercellSpacing (page 1459)

setRowHeight

Sets the height for rows to *rowHeight* and invokes tile (page 1476).

```
public void setRowHeight(float rowHeight)
```

See Also

rowHeight (page 1462)

setSortDescriptors

Sets the receiver's sort descriptors to the NSSortDescriptor objects in *array*.

```
public void setSortDescriptors(NSArray array)
```

Discussion

A table column is considered sortable if it has a sort descriptor that specifies the sorting direction, a key to sort by, and a selector defining how to sort. The array of sort descriptors is archived. Sort descriptors persist along with other column information if an autosave name is set.

Availability

Available in Mac OS X v10.3 and later.

See Also

sortDescriptors (page 1475)

setUsesAlternatingRowBackgroundColors

Sets whether the receiver uses the standard alternating row colors, or a solid color, for its background.

```
public void setUsesAlternatingRowBackgroundColors(boolean useAlternatingRowColors)
```

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

See Also
`usesAlternatingRowBackgroundColors` (page 1477)

setVerticalMotionCanBeginDrag

Sets whether vertical motion is treated as a drag or selection change to *`flag`*.

```
public void setVerticalMotionCanBeginDrag(boolean flag)
```

Discussion
If *`flag`* is `false` then vertical motion will not start a drag. The default is `true`.

Note that horizontal motion is always a valid motion to begin a drag. Most often, you would want to disable vertical dragging when it's expected that horizontal dragging is the natural motion.

See Also
`verticalMotionCanBeginDrag` (page 1477)

sizeLastColumnToFit

Resizes the last column if there's room so the receiver fits exactly within its enclosing NSClipView.

```
public void sizeLastColumnToFit()
```

See Also
`setAutoresizesAllColumnsToFit` (page 1468)
`minWidth` (page 1425) (NSTableColumn)
`maxWidth` (page 1425) (NSTableColumn)

sizeToFit

Changes the width of columns in the receiver so all columns are visible.

```
public void sizeToFit()
```

Discussion
All columns are resized to the same size, up to a column's maximum size. This method then invokes `tile` (page 1476).

sortDescriptors

Returns the receiver's sort descriptors.

```
public NSArray sortDescriptors()
```

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

See Also
`setSortDescriptors` (page 1473)

tableColumns

Returns the NSTableColumns in the receiver.

```
public NSArray tableColumns()
```

tableColumnWithIdentifier

Returns the NSTableColumn object for the first column whose identifier is equal to *anObject*, as compared using `equals`, or `null` if no columns are found with the specified identifier.

```
public NSTableColumn tableColumnWithIdentifier(Object anObject)
```

See Also
`columnWithIdentifier` (page 1452)

textDidBeginEditing

Posts a `ControlTextDidBeginEditingNotification` (page 466) to the default notification center, as described in the NSControl class specification.

```
public void textDidBeginEditing(NSNotification aNotification)
```

Discussion
aNotification is the NSNotification posted by the field editor; see the NSText (page 1505) class specifications for more information on this text delegate method.

See Also
`textShouldBeginEditing` (page 1476)

textDidChange

Sends `textDidChange` (page 1475) to the edited cell and posts a `ControlTextDidChangeNotification` (page 467) to the default notification center, as described in the NSControl class specification.

```
public void textDidChange(NSNotification aNotification)
```

Discussion

aNotification is the NSNotification posted by the field editor; see the NSText (page 1505) class specification for more information on this text delegate method.

textDidEndEditing

Updates the data source based on the newly edited value and selects another cell for editing if possible according to the character that ended editing (Return, Tab, Backtab).

```
public void textDidEndEditing(NSNotification aNotification)
```

Discussion

aNotification is the NSNotification posted by the field editor; see the NSText (page 1505) class specification for more information on this text delegate method.

See Also

`textShouldEndEditing` (page 1476)

textShouldBeginEditing

Queries the delegate using `controlTextShouldBeginEditing` (page 465), returning the delegate's response, or simply returning `true` to allow editing of *textObject* if the delegate doesn't respond to that method.

```
public boolean textShouldBeginEditing(NSText textObject)
```

Discussion

See the NSText (page 1505) class specification for more information on this text delegate method.

See Also

`textDidBeginEditing` (page 1475)

textShouldEndEditing

Validates the *textObject* cell being edited and queries the delegate using `controlTextShouldEndEditing` (page 465), returning the delegate's response if it responds to that method.

```
public boolean textShouldEndEditing(NSText textObject)
```

Discussion

If it doesn't, it returns `true` if the cell's new value is valid and `false` if it isn't. See the NSText (page 1505) class specification for more information on this text delegate method.

See Also

`textDidEndEditing` (page 1476)

tile

Properly sizes the receiver and its header view and marks it as needing display.

```
public void tile()
```

Discussion
Also resets cursor rectangles for the header view and line scroll amounts for the NSScrollView.

See Also
`setNeedsDisplay` (page 1779) (NSView)

usesAlternatingRowBackgroundColors

Returns whether the receiver uses the standard alternating row colors, or a solid color, for its background.

```
public boolean usesAlternatingRowBackgroundColors()
```

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

See Also
`setUsesAlternatingRowBackgroundColors` (page 1474)

verticalMotionCanBeginDrag

Returns whether vertical motion is treated as a drag or selection change.

```
public boolean verticalMotionCanBeginDrag()
```

Discussion
`false` means that vertical motion will not start a drag. Note that horizontal motion is always a valid motion to begin a drag.

See Also
`setVerticalMotionCanBeginDrag` (page 1474)

Constants

NSTableView defines the following constants to specify drop operations. For example, given a table with n rows (numbered with row 0 at the top visually), a row of n–1 and operation of `DropOn` would specify a drop on the last row. To specify a drop below the last row, one would use a row of n and `DropAbove` for the operation.

Constant	Description
`DropOn`	Specifies that the drop should occur on the specified row.
`DropAbove`	Specifies that the drop should occur above the specified row.

NSTableView defines the following constants to specify grid styles. Either it can be `GridNone` (page 1478) or it can contain any of the following options, combined using the C bitwise OR operator. These constants are used by `gridStyleMask` (page 1457) and `setGridStyleMask` (page 1472).

Constant	Description
GridNone	Specifies that no grid lines should be displayed.
SolidVerticalGridLineMask	Specifies that vertical grid lines should be displayed.
SolidHorizontalGridLineMask	Specifies that horizontal grid lines should be displayed.

The following constants specify the autoresizing styles. These constants are used by columnAutoresizingStyle (page 1451) and setColumnAutoresizingStyle (page 1469).

Constant	Description
NoColumnAutoresizinge	Disable table column autoresizing. Available in Mac OS X v10.4 and later.
UniformColumnAutoresizingStyle	Autoresize all columns by distributing space equally, simultaeously. Available in Mac OS X v10.4 and later.
SequentialColumnAutoresizingStyle	Autoresize each table column sequentially, from left to right. Proceed to the next column when the current column has reached its minimum or maximum size. Available in Mac OS X v10.4 and later.
ReverseSequentialColumnAutoresizingStyle	Autoresize each table column sequentially, from right to left. Proceed to the next column when the current column has reached its minimum or maximum size. Available in Mac OS X v10.4 and later.
LastColumnOnlyAutoresizingStyle	Autoresize only the last table colum. When that table column can no longer be resized, stop autoresizing. Normally you should use one of the sequential autoresizing modes instead. Available in Mac OS X v10.4 and later.
FirstColumnOnlyAutoresizingStyle	Autoresize only the first table colum. When that table column can no longer be resized, stop autoresizing. Normally you should use one of the sequential autoresizing modes instead. Available in Mac OS X v10.4 and later.

Delegate Methods

selectionShouldChangeInTableView

Returns `true` to permit *aTableView* to change its selection (typically a row being edited), `false` to deny permission.

```
public abstract boolean selectionShouldChangeInTableView(NSTableView aTableView)
```

Discussion

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.

tableViewColumnDidMove

Informs the delegate that a column was moved by user action in the NSTableView.

```
public abstract void tableViewColumnDidMove(NSNotification aNotification)
```

Discussion

aNotification is a `TableViewColumnDidMoveNotification` (page 1482).

tableViewColumnDidResize

Informs the delegate that a column was resized in the NSTableView.

```
public abstract void tableViewColumnDidResize(NSNotification aNotification)
```

Discussion

aNotification is a `TableViewColumnDidResizeNotification` (page 1482).

tableViewDidClickTableColumn

Sent at the time the mouse button subsequently goes up in *tableView* and *tableColumn* has been "clicked" without having been dragged anywhere.

```
public abstract void tableViewDidClickTableColumn(NSTableView tableView,
    NSTableColumn tableColumn)
```

tableViewDidDragTableColumn

Sent at the time the mouse button goes up in *tableView* and *tableColumn* has been dragged during the time the mouse button was down.

```
public abstract void tableViewDidDragTableColumn(NSTableView tableView, NSTableColumn
     tableColumn)
```

tableViewHeightOfRow

Returns the height of *row* in *tableView*.

```
public abstract float tableViewHeightOfRow(NSTableView tableView, int row)
```

Discussion

You should implement this method if your table supports varying row heights. The height returned should not include intercell spacing and must be greater than zero.

Although NSTableViews may cache the returned values, you should ensure that this method is efficient. When you change a row's height you must invalidate the existing row height by calling `noteHeightOfRowsWithIndexesChanged` (page 1460). NSTableView automatically invalidates its entire row height cache when `reloadData` (page 1462) and `noteNumberOfRowsChanged` (page 1460) are called.

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

tableViewMouseDownInHeaderOfTableColumn

Sent to the delegate whenever the mouse button is clicked in *tableView* while the cursor is in a column header *tableColumn*.

```
public abstract void tableViewMouseDownInHeaderOfTableColumn(NSTableView tableView,
     NSTableColumn tableColumn)
```

tableViewSelectionDidChange

Informs the delegate that the NSTableView's selection has changed.

```
public abstract void tableViewSelectionDidChange(NSNotification aNotification)
```

Discussion
aNotification is a `TableViewSelectionDidChangeNotification` (page 1482).

tableViewSelectionIsChanging

Informs the delegate that the NSTableView's selection is in the process of changing (typically because the user is dragging the cursor across a number of rows).

```
public abstract void tableViewSelectionIsChanging(NSNotification aNotification)
```

Discussion
aNotification is a `TableViewSelectionIsChangingNotification` (page 1482).

tableViewShouldEditLocation

Returns `true` to permit *aTableView* to edit the cell at *rowIndex* in *aTableColumn*, `false` to deny permission.

```
public abstract boolean tableViewShouldEditLocation(NSTableView aTableView,
    NSTableColumn aTableColumn, int rowIndex)
```

Discussion
The delegate can implement this method to disallow editing of specific cells.

tableViewShouldSelectRow

Returns `true` to permit *aTableView* to select the row at *rowIndex*, `false` to deny permission.

```
public abstract boolean tableViewShouldSelectRow(NSTableView aTableView, int
    rowIndex)
```

Discussion
The delegate can implement this method to disallow selection of particular rows.

tableViewShouldSelectTableColumn

Returns `true` to permit *aTableView* to select *aTableColumn*, `false` to deny permission.

```
public abstract boolean tableViewShouldSelectTableColumn(NSTableView aTableView,
    NSTableColumn aTableColumn)
```

Discussion
The delegate can implement this method to disallow selection of particular columns.

tableViewToolTipForCell

Returns a string that is displayed as a tooltip for *aCell* in *aTableColumn* of *aTableView*.

```
public abstract String tableViewToolTipForCell(NSTableView aTableView, NSCell aCell,
     NSMutableRect rect, NSTableColumn aTableColumn, int row, NSPoint mouseLocation)
```

Discussion
The *mouseLocation* is the current mouse location in view coordinates. The *row* is the row of the cell and *aTableColumn* is the NSTableColumn that contains the cell. The *rect* represents the proposed active area of the tooltip. By default, *rect* is computed as `[cell drawingRectForBounds:cellFrame]`. You can modify *rect* to provide an alternative active area. Return `null` or the empty string if no tooltip is desired.

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

tableViewWillDisplayCell

Informs the delegate that *aTableView* will display the cell at *rowIndex* in *aTableColumn* using *aCell*.

```
public abstract void tableViewWillDisplayCell(NSTableView aTableView, Object aCell,
     NSTableColumn aTableColumn, int rowIndex)
```

Discussion
The delegate can modify the display attributes of *aCell* to alter the appearance of the cell. Because *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.

Notifications

TableViewColumnDidMoveNotification

Posted whenever a column is moved by user action in an NSTableView. The notification object is the NSTableView in which a column moved. The `userInfo` dictionary contains the following information:

Key	Value
`"NSOldColumn"`	The integer value of the column's original index.
`"NSNewColumn"`	The integer value of the column's present index.

See Also
`moveColumnToColumn` (page 1459)

TableViewColumnDidResizeNotification

Posted whenever a column is resized in an NSTableView. The notification object is the NSTableView in which a column was resized. The `userInfo` dictionary contains the following information:

Key	Value
`"NSTableColumn"`	The column that was resized.
`"NSOldWidth"`	The integer value of the column's original width.

TableViewSelectionDidChangeNotification

Posted after an NSTableView's selection changes. The notification object is the NSTableView whose selection changed. This notification does not contain a `userInfo` dictionary.

TableViewSelectionIsChangingNotification

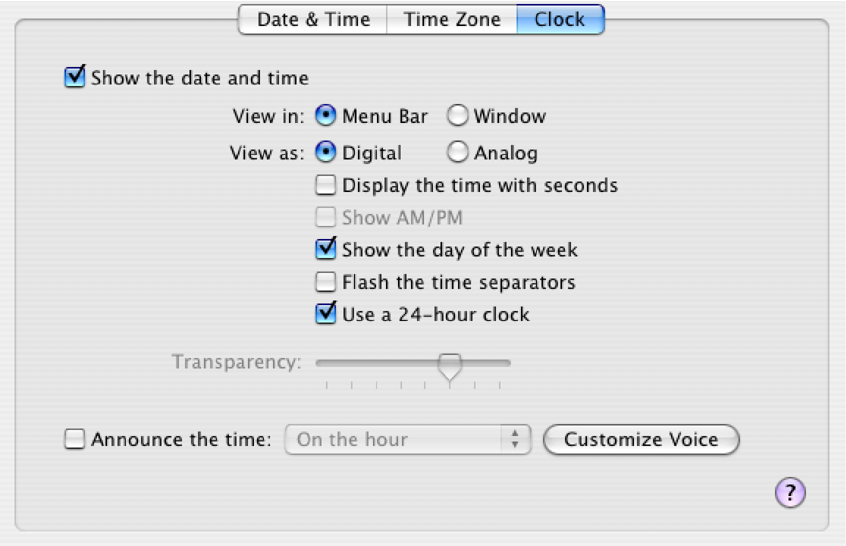
Posted as an NSTableView's selection changes (while the mouse button is still down). The notification object is the NSTableView whose selection is changing. This notification does not contain a `userInfo` dictionary.

NSTabView

Inherits from	NSView : NSResponder : NSObject
Implements	NSCoding (NSResponder)
Package:	com.apple.cocoa.application
Companion guide	Tab Views

Overview

An NSTabView is a convenient way to present information in multiple pages. The view contains a row of tabs that give the appearance of folder tabs, as shown in the following figure. The user selects the desired page by clicking the appropriate tab or using the arrow keys to move between pages. Each page displays a view hierarchy provided by your application.



☑ Show the date and time

View in: ● Menu Bar ○ Window

View as: ● Digital ○ Analog

☐ Display the time with seconds

☐ Show AM/PM

☑ Show the day of the week

☐ Flash the time separators

☑ Use a 24-hour clock

Transparency:

☐ Announce the time: On the hour

Tasks

Constructors

NSTabView (page 1487)
Creates an NSTabView with a zero-sized frame rectangle.

Adding and Removing Tabs

`addTabViewItem` (page 1487)
Adds the tab item specified by *tabViewItem*.

`insertTabViewItem` (page 1489)
Inserts *tabViewItem* into the receiver's array of tab view items at *index*.

`removeTabViewItem` (page 1490)
Removes the item specified by *tabViewItem* from the receiver's array of tab view items.

Accessing Tabs

`indexOfTabViewItem` (page 1489)
Returns the index of the item that matches *tabViewItem*, or `NSArray.NotFound` if the item is not found.

`indexOfTabViewItemWithIdentifier` (page 1489)
Returns the index of the item that matches *identifier*, or `NSArray.NotFound` if the item is not found.

`numberOfTabViewItems` (page 1490)
Returns the number of items in the receiver's array of tab view items.

`tabViewItemAtIndex` (page 1494)
Returns the tab view item at *index* in the tab view's array of items.

`tabViewItems` (page 1495)
Returns the receiver's array of tab view items.

Selecting a Tab

`selectFirstTabViewItem` (page 1491)
This action method selects the first tab view item.

`selectLastTabViewItem` (page 1491)
This action method selects the last tab view item.

`selectNextTabViewItem` (page 1491)
This action method selects the next tab view item in the sequence.

`selectPreviousTabViewItem` (page 1491)
This action method selects the previous tab view item in the sequence.

`selectTabViewItem` (page 1492)
Selects the tab view item specified by *tabViewItem*.

`selectTabViewItemAtIndex` (page 1492)
Selects the tab view item specified by *index*.

`selectTabViewItemWithIdentifier` (page 1492)
Selects the tab view item specified by *identifier*.

`selectedTabViewItem` (page 1491)
Returns the tab view item for the currently selected tab, or `null` if no item is selected.

`takeSelectedTabViewItemFromSender` (page 1495)

Modifying the Font

`font` (page 1489)
Returns the font for tab label text.

`setFont` (page 1494)
Sets the font for tab label text to *`font`*.

Modifying the Tab Type

`setTabViewType` (page 1494)
Sets the tab type to *`tabViewType`*.

`tabViewType` (page 1495)
Returns the tab type for the receiver.

Modifying Controls Tint

`controlTint` (page 1488)
Returns the receiver's control tint.

`setControlTint` (page 1493)
Sets the receiver's control tint to *`controlTint`*.

Manipulating the Background

`drawsBackground` (page 1488)
Returns `true` if the receiver draws a background color when the tab view type is `NoTabsNoBorder`. If the receiver uses bezeled edges or a line border, the appropriate background color for that border is used.

`setDrawsBackground` (page 1493)

Determining the Size

`minimumSize` (page 1490)
Returns the minimum size necessary for the receiver to display tabs in a useful way.

`contentRect` (page 1488)
Returns the rectangle describing the content area of the receiver.

`controlSize` (page 1488)
Returns the size of the receiver.

`setControlSize` (page 1493)
Sets the size of the receiver to *`controlSize`*.

Truncating Tab Labels

allowsTruncatedLabels (page 1487)
Returns true if the receiver allows truncating for labels that don't fit on a tab.

setAllowsTruncatedLabels (page 1492)
Sets whether the receiver allows truncating for names that don't fit on a tab to *allowTruncatedLabels*.

Assigning a Delegate

setDelegate (page 1493)
Sets the receiver's delegate to *anObject*.

delegate (page 1488)
Returns the receiver's delegate.

Event Handling

tabViewItemAtPoint (page 1494)
Returns the tab view item identified by *point*.

View's Window

window (page 1495)

Selecting an item

tabViewShouldSelectTabViewItem (page 1497) *delegate method*
Invoked just before *tabViewItem* in *tabView* is selected.

tabViewWillSelectTabViewItem (page 1497) *delegate method*
Informs the delegate that *tabView* is about to select *tabViewItem*.

tabViewDidSelectTabViewItem (page 1497) *delegate method*
Informs the delegate that *tabView* has selected *tabViewItem*.

Changing number of items in view

tabViewDidChangeNumberOfTabViewItems (page 1497) *delegate method*
Informs the delegate that the number of tab view items in *tabView* has changed.

Constructors

NSTabView

Creates an NSTabView with a zero-sized frame rectangle.

```
public NSTabView()
```

Creates an NSTabView with *frameRect* as its frame rectangle.

```
public NSTabView(NSRect frameRect)
```

Instance Methods

addTabViewItem

Adds the tab item specified by *tabViewItem*.

```
public void addTabViewItem(NSTabViewItem tabViewItem)
```

Discussion
The item is added at the end of the array of tab items, so the new tab appears on the right side of the view. If the delegate supports it, invokes the delegate's `tabViewDidChangeNumberOfTabViewItems` (page 1497) method.

See Also
`insertTabViewItem` (page 1489)
`numberOfTabViewItems` (page 1490)
`removeTabViewItem` (page 1490)
`tabViewItemAtIndex` (page 1494)
`tabViewItems` (page 1495)

allowsTruncatedLabels

Returns `true` if the receiver allows truncating for labels that don't fit on a tab.

```
public boolean allowsTruncatedLabels()
```

Discussion
The default is `false`. When truncating is allowed, the tab view inserts an ellipsis, if necessary, to fit a label in the tab.

See Also
`setAllowsTruncatedLabels` (page 1492)

contentRect

Returns the rectangle describing the content area of the receiver.

```
public NSRect contentRect()
```

Discussion
This area does not include the space required for the receiver's tabs or borders (if any).

controlSize

Returns the size of the receiver.

```
public int controlSize()
```

Discussion
Valid return values are described in "Constants" (page 1495).

See Also
`setControlSize` (page 1493)

controlTint

Returns the receiver's control tint.

```
public int controlTint()
```

Discussion
Valid return values are described in "Constants" (page 1495).

See Also
`setControlTint` (page 1493)

delegate

Returns the receiver's delegate.

```
public Object delegate()
```

See Also
`setDelegate` (page 1493)

drawsBackground

Returns `true` if the receiver draws a background color when the tab view type is `NoTabsNoBorder`. If the receiver uses bezeled edges or a line border, the appropriate background color for that border is used.

```
public boolean drawsBackground()
```

See Also
`setTabViewType` (page 1494)

`setDrawsBackground` (page 1493)

font

Returns the font for tab label text.

```
public NSFont font()
```

See Also
`setFont` (page 1494)

indexOfTabViewItem

Returns the index of the item that matches *tabViewItem*, or `NSArray.NotFound` if the item is not found.

```
public int indexOfTabViewItem(NSTabViewItem tabViewItem)
```

Discussion
A tab view keeps an array containing one tab view item for each tab in the view—this array is the one that is searched. The returned index is base 0.

See Also
`indexOfTabViewItemWithIdentifier` (page 1489)
`insertTabViewItem` (page 1489)
`numberOfTabViewItems` (page 1490)
`tabViewItemAtIndex` (page 1494)

indexOfTabViewItemWithIdentifier

Returns the index of the item that matches *identifier*, or `NSArray.NotFound` if the item is not found.

```
public int indexOfTabViewItemWithIdentifier(Object identifier)
```

Discussion
A tab view keeps an array containing one tab view item for each tab in the view—this array is the one that is searched. The returned index is base 0.

See Also
`indexOfTabViewItem` (page 1489)
`insertTabViewItem` (page 1489)
`numberOfTabViewItems` (page 1490)
`tabViewItemAtIndex` (page 1494)

insertTabViewItem

Inserts *tabViewItem* into the receiver's array of tab view items at *index*.

```
public void insertTabViewItem(NSTabViewItem tabViewItem, int index)
```

Discussion

The *index* parameter is base 0. If there is a delegate and the delegate supports it, sends the delegate the `tabViewDidChangeNumberOfTabViewItems` (page 1497) message.

See Also

`indexOfTabViewItem` (page 1489)

`indexOfTabViewItemWithIdentifier` (page 1489)

`numberOfTabViewItems` (page 1490)

`tabViewItemAtIndex` (page 1494)

minimumSize

Returns the minimum size necessary for the receiver to display tabs in a useful way.

```
public NSSize minimumSize()
```

Discussion

You can use the value returned by this method to limit how much a user can resize a tab view.

See Also

`setTabViewType` (page 1494)

numberOfTabViewItems

Returns the number of items in the receiver's array of tab view items.

```
public int numberOfTabViewItems()
```

Discussion

Because there is one item in the array for each tab in the view, this number is equivalent to the number of tabs in the view.

See Also

`indexOfTabViewItem` (page 1489)

`tabViewItems` (page 1495)

removeTabViewItem

Removes the item specified by *tabViewItem* from the receiver's array of tab view items.

```
public void removeTabViewItem(NSTabViewItem tabViewItem)
```

Discussion

If there is a delegate and the delegate supports it, sends the delegate the `tabViewDidChangeNumberOfTabViewItems` (page 1497) message.

See Also

`addTabViewItem` (page 1487)

`insertTabViewItem` (page 1489)

`tabViewItems` (page 1495)

selectedTabViewItem

Returns the tab view item for the currently selected tab, or `null` if no item is selected.

```
public NSTabViewItem selectedTabViewItem()
```

See Also
`selectTabViewItemAtIndex` (page 1492)

selectFirstTabViewItem

This action method selects the first tab view item.

```
public void selectFirstTabViewItem(Object sender)
```

See Also
`selectTabViewItem` (page 1492)

selectLastTabViewItem

This action method selects the last tab view item.

```
public void selectLastTabViewItem(Object sender)
```

See Also
`selectTabViewItem` (page 1492)

selectNextTabViewItem

This action method selects the next tab view item in the sequence.

```
public void selectNextTabViewItem(Object sender)
```

Discussion
If the currently visible item is the last item in the sequence, this method does nothing, and the last page remains displayed.

See Also
`selectTabViewItem` (page 1492)

selectPreviousTabViewItem

This action method selects the previous tab view item in the sequence.

```
public void selectPreviousTabViewItem(Object sender)
```

Discussion
If the currently visible item is the first item in the sequence, this method does nothing, and the first page remains displayed.

See Also

`selectTabViewItem` (page 1492)

selectTabViewItem

Selects the tab view item specified by *`tabViewItem`*.

```
public void selectTabViewItem(NSTabViewItem tabViewItem)
```

Discussion

If there is a delegate and the delegate supports it, sends the delegate the `tabViewShouldSelectTabViewItem` (page 1497) message.

See Also

`insertTabViewItem` (page 1489)

`selectedTabViewItem` (page 1491)

selectTabViewItemAtIndex

Selects the tab view item specified by *`index`*.

```
public void selectTabViewItemAtIndex(int index)
```

Discussion

The *`index`* parameter is base 0. If there is a delegate and the delegate supports it, sends the delegate the `tabViewShouldSelectTabViewItem` (page 1497) message.

See Also

`insertTabViewItem` (page 1489)

`selectedTabViewItem` (page 1491)

selectTabViewItemWithIdentifier

Selects the tab view item specified by *`identifier`*.

```
public void selectTabViewItemWithIdentifier(Object identifier)
```

See Also

`setIdentifier` (page 1502) (NSTabViewItem)

`identifier` (page 1501) (NSTabViewItem)

`selectTabViewItemAtIndex` (page 1492)

`selectedTabViewItem` (page 1491)

setAllowsTruncatedLabels

Sets whether the receiver allows truncating for names that don’t fit on a tab to *`allowTruncatedLabels`*.

```
public void setAllowsTruncatedLabels(boolean allowTruncatedLabels)
```

See Also

`allowsTruncatedLabels` (page 1487)

setControlSize

Sets the size of the receiver to *controlSize*.

```
public void setControlSize(int controlSize)
```

Discussion

Valid values for *controlSize* are described in "Constants" (page 1495).

See Also

`controlSize` (page 1488)

setControlTint

Sets the receiver's control tint to *controlTint*.

```
public void setControlTint(int controlTint)
```

Discussion

Valid values for *controlTint* are described in "Constants" (page 1495).

See Also

`controlTint` (page 1488)

setDelegate

Sets the receiver's delegate to *anObject*.

```
public void setDelegate(Object anObject)
```

See Also

`delegate` (page 1488)

setDrawsBackground

```
public void setDrawsBackground(boolean flag)
```

Discussion

Sets whether a background is drawn when the view type is `NoTabsNoBorder` to *flag*. If the receiver has a bezeled border or a line border, the appropriate background for that border is used.

See Also

`setTabViewType` (page 1494)

`drawsBackground` (page 1488)

setFont

Sets the font for tab label text to *font*.

```
public void setFont(NSFont font)
```

Discussion
Tab height is adjusted automatically to accommodate a new font size. If the view allows truncating, tab labels are truncated as needed.

See Also
`allowsTruncatedLabels` (page 1487)
`font` (page 1489)
`setAllowsTruncatedLabels` (page 1492)

setTabViewType

Sets the tab type to *tabViewType*.

```
public void setTabViewType(int tabViewType)
```

Discussion
The available types are described in "Constants" (page 1495).

See Also
`tabViewType` (page 1495)

tabViewItemAtIndex

Returns the tab view item at *index* in the tab view's array of items.

```
public NSTabViewItem tabViewItemAtIndex(int index)
```

Discussion
The *index* parameter is base 0.

See Also
`indexOfTabViewItem` (page 1489)
`insertTabViewItem` (page 1489)
`tabViewItems` (page 1495)

tabViewItemAtPoint

Returns the tab view item identified by *point*.

```
public NSTabViewItem tabViewItemAtPoint(NSPoint point)
```

Discussion
You can use this method to find a tab view item based on a user's mouse click.

tabViewItems

Returns the receiver's array of tab view items.

```
public NSArray tabViewItems()
```

Discussion
A tab view keeps an array containing one tab view item for each tab in the view.

See Also
`numberOfTabViewItems` (page 1490)
`tabViewItemAtIndex` (page 1494)

tabViewType

Returns the tab type for the receiver.

```
public int tabViewType()
```

Discussion
The available types are described in "Constants" (page 1495).

takeSelectedTabViewItemFromSender

```
public void takeSelectedTabViewItemFromSender(Object sender)
```

Discussion
If *sender* responds to the `indexOfSelectedItem` method, this method invokes that method and selects the tab view item at the specified index. If *sender* does not respond to `indexOfSelectedItem` but is an instance of NSMatrix, this method uses the index of the matrix's currently selected cell.

The location of the selected cell is a zero-based number, obtained by counting the number of cells up to and including the selected cell. Cells are counted from left to right and from top to bottom. For example in a 5-by-5 matrix, if the selected cell is three rows down in column five (location [2,4] in the matrix), the corresponding index would be 14.

window

```
public NSWindow window()
```

Discussion
Returns the receiver's window object, or `null` if there is none. If the receiver is not the currently visible tab view this method returns `null`.

Constants

These constants specify the tab view's type:

Constant	Description
`TopTabsBezelBorder`	The view includes tabs on the top of the view and has a bezeled border (the default).
`NoTabsBezelBorder`	The view does not include tabs and has a bezeled border.
`NoTabsLineBorder`	The view does not include tabs and has a lined border.
`NoTabsNoBorder`	The view does not include tabs and has no border.
`BottomTabsBezelBorder`	Tabs are on the bottom of the view with a bezeled border.
`LeftTabsBezelBorder`	Tabs are on the left of the view with a bezeled border.
`RightTabsBezelBorder`	Tabs are on the right of the view with a bezeled border.

These constants describe the current display state of a tab:

Constant	Description
`BackgroundTab`	A tab that's not being displayed.
`PressedTab`	A tab that the user is in the process of clicking. That is, the user has pressed the mouse button while the cursor is over the tab, but has not released the mouse button.
`SelectedTab`	The tab that's being displayed.

These constants specify a view's tint. They're used by `controlTint` (page 1488) and `setControlTint` (page 1493).

Constant	Description
`DefaultControlTint`	The current default tint setting.
`ClearControlTint`	Clear control tint.
`BlueControlTint`	Aqua control tint
`GraphiteControlTint`	Graphite control tint

These constants specify a view's size. They're used by `controlSize` (page 1488) and `setControlSize` (page 1493).

Constant	Description
`RegularControlSize`	The control is sized as regular.
`SmallControlSize`	The control has a smaller size. This constant is for controls that cannot be resized in one direction, such as push buttons, radio buttons, checkboxes, sliders, scroll bars, pop-up buttons, tabs, and progress indicators. You should use a small system font when using with a small control.

Constant	Description
`MiniControlSize`	The control has a smaller size than `SmallControlSize`.

Delegate Methods

tabViewDidChangeNumberOfTabViewItems

Informs the delegate that the number of tab view items in *`tabView`* has changed.

```
public abstract void tabViewDidChangeNumberOfTabViewItems(NSTabView tabView)
```

See Also

`numberOfTabViewItems` (page 1490)

tabViewDidSelectTabViewItem

Informs the delegate that *`tabView`* has selected *`tabViewItem`*.

```
public abstract void tabViewDidSelectTabViewItem(NSTabView tabView, NSTabViewItem
     tabViewItem)
```

tabViewShouldSelectTabViewItem

Invoked just before *`tabViewItem`* in *`tabView`* is selected.

```
public abstract boolean tabViewShouldSelectTabViewItem(NSTabView tabView,
    NSTabViewItem tabViewItem)
```

Discussion

The delegate can return `false` to prevent selection of specific tabs.

tabViewWillSelectTabViewItem

Informs the delegate that *`tabView`* is about to select *`tabViewItem`*.

```
public abstract void tabViewWillSelectTabViewItem(NSTabView tabView, NSTabViewItem
     tabViewItem)
```

NSTabViewItem

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.application
Companion guide	Tab Views

Overview

An NSTabViewItem is a convenient way for presenting information in multiple pages. A tab view is usually distinguished by a row of tabs that give the visual appearance of folder tabs. When the user clicks a tab, the tab view displays a view page provided by your application. A tab view keeps a zero-based array of NSTabViewItems, one for each tab in the view.

Tasks

Constructors

`NSTabViewItem` (page 1501)
Creates an empty NSTabViewItem.

Working with Labels

`drawLabel` (page 1501)
Draws the receiver's label in *`tabRect`*, which is the area between the curved end caps.

`label` (page 1502)
Returns the label text for the receiver.

`setLabel` (page 1503)
Sets the label text for the receiver to *`label`*.

`sizeOfLabel` (page 1503)
Calculates the size of the receiver's label.

Checking the Tab Display State

`tabState` (page 1503)
Returns the current display state of the tab associated with the receiver.

Assigning an Identifier Object

`identifier` (page 1501)
Returns the receiver's optional identifier object.

`setIdentifier` (page 1502)
Sets the receiver's optional identifier object to *identifier*.

Setting the Color

`color` (page 1501)
Returns the color for the receiver.

`setColor` (page 1502)
Deprecated. NSTabViewItems use a color supplied by the current theme.

Assigning a View

`view` (page 1504)
Returns the view associated with the receiver.

`setView` (page 1503)
Sets the view associated with the receiver to *view*.

Setting the Initial First Responder

`initialFirstResponder` (page 1502)
Returns the initial first responder for the view associated with the receiver.

`setInitialFirstResponder` (page 1503)
Sets the initial first responder for the view associated with the receiver (the view that is displayed when a user clicks on the tab) to *view*.

Accessing the Parent Tab View

`tabView` (page 1504)
Returns the parent tab view for the receiver.

Constructors

NSTabViewItem

Creates an empty NSTabViewItem.

```
public NSTabViewItem()
```

Creates a new NSTabViewItem. Sets the item's identifier object to *identifier*, if it is not `null`. Use this constructor when creating tab view items programmatically.

```
public NSTabViewItem(Object identifier)
```

Instance Methods

color

Returns the color for the receiver.

```
public NSColor color()
```

Discussion
The color is specified by the current theme.

See Also
`setColor` (page 1502)

drawLabel

Draws the receiver's label in *tabRect*, which is the area between the curved end caps.

```
public void drawLabel(boolean shouldTruncateLabel, NSRect tabRect)
```

Discussion
If *shouldTruncateLabel* is `false`, draws the full label in the rectangle specified by *tabRect*. If *shouldTruncateLabel* is `true`, draws the truncated label. You can override this method to perform customized label drawing. For example, you might want to add an icon to each tab in the view.

See Also
`sizeOfLabel` (page 1503)

identifier

Returns the receiver's optional identifier object.

```
public Object identifier()
```

Discussion

To customize how your application works with tabs, you can initialize each tab view item with an identifier object.

See Also

`setIdentifier` (page 1502)

initialFirstResponder

Returns the initial first responder for the view associated with the receiver.

```
public Object initialFirstResponder()
```

See Also

`setInitialFirstResponder` (page 1503)

label

Returns the label text for the receiver.

```
public String label()
```

See Also

`setLabel` (page 1503)

setColor

Deprecated. NSTabViewItems use a color supplied by the current theme.

```
public void setColor(NSColor color)
```

See Also

`color` (page 1501)

setIdentifier

Sets the receiver's optional identifier object to *identifier*.

```
public void setIdentifier(Object identifier)
```

Discussion

To customize how your application works with tabs, you can specify an identifier object for each tab view item.

See Also

`identifier` (page 1501)

setInitialFirstResponder

Sets the initial first responder for the view associated with the receiver (the view that is displayed when a user clicks on the tab) to *view*.

```
public void setInitialFirstResponder(NSView view)
```

See Also
initialFirstResponder (page 1502)

setLabel

Sets the label text for the receiver to *label*.

```
public void setLabel(String label)
```

See Also
label (page 1502)

setView

Sets the view associated with the receiver to *view*.

```
public void setView(NSView view)
```

Discussion
This is the view displayed when a user clicks the tab.

See Also
view (page 1504)

sizeOfLabel

Calculates the size of the receiver's label.

```
public NSSize sizeOfLabel(boolean shouldTruncateLabel)
```

Discussion
If *shouldTruncateLabel* is false, returns the size of the receiver's full label. If *shouldTruncateLabel* is true, returns the truncated size. If your application does anything to change the size of tab labels, such as overriding the drawLabel (page 1501) method to add an icon to each tab, you should override sizeOfLabel (page 1503) too so the NSTabView knows the correct size for the tab label.

See Also
drawLabel (page 1501)

setFont (page 1494) (NSTabView)

tabState

Returns the current display state of the tab associated with the receiver.

```
public int tabState()
```

Discussion
The possible values are `NSTabView.SelectedTab`, `NSTabView.BackgroundTab`, or `NSTabView.PressedTab`. Your application does not directly set the tab state.

tabView

Returns the parent tab view for the receiver.

```
public NSTabView tabView()
```

Discussion
Note that this is the tab view itself, not the view displayed when a user clicks the tab.

A tab view item normally learns about its parent tab view when it is inserted into the view's array of items. The NSTabView methods `addTabViewItem` (page 1487) and `insertTabViewItem` (page 1489) set the tab view for the added or inserted item.

See Also
`setView` (page 1503)
`view` (page 1504)

view

Returns the view associated with the receiver.

```
public NSView view()
```

Discussion
This is the view displayed when a user clicks the tab.

See Also
`setView` (page 1503)

NSText

Inherits from	NSView : NSResponder : NSObject
Implements	NSChangeSpelling NSIgnoreMisspelledWords NSCoding (NSResponder)
Package:	com.apple.cocoa.application
Companion guide	Text System Overview

Class at a Glance

NSText declares the most general programmatic interface for objects that manage text. You usually use instances of its subclass, NSTextView.

Commonly Used Methods

`readRTFDFromFile` (page 1519)
    Reads an `.rtf` or `.rtfd` file.

`writeRTFDToFile` (page 1529)
    Writes the receiver's text to a file.

`string` (page 1528)
    Returns the receiver's text without attributes.

`RTFFromRange` (page 1521)
    Returns the receiver's text with attributes.

`RTFDFromRange` (page 1521)
    Returns the receiver's text with attributes and attachments.

Overview

The NSText class declares the most general programmatic interface to objects that manage text. Cocoa offers a subclass of NSText, NSTextView, that extends the interface declared by NSText and provides much more sophisticated functionality than that declared in NSText.

NSText's constructor creates an instance of a concrete subclass, such as NSTextView. Instances of any of these classes are generically called text objects.

Text objects are used by the Application Kit wherever text appears in interface objects: A text 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 can also create text objects for its own purposes.

Interfaces Implemented

NSChangeSpelling
`changeSpelling` (page 1941)

NSIgnoreMisspelledWords
`ignoreSpelling` (page 1974)

Tasks

Constructors

`NSText` (page 1512)
Creates an NSText with a zero-sized frame rectangle.

Getting the Characters

`string` (page 1528)
Returns the characters of the receiver's text.

Setting Graphics Attributes

`setBackgroundColor` (page 1522)
Sets the receiver's background color to *`aColor`*.

`backgroundColor` (page 1513)
Returns the receiver's background color.

`setDrawsBackground` (page 1523)
Controls whether the receiver draws its background.

`drawsBackground` (page 1516)
Returns `true` if the receiver draws its background, `false` if it doesn't.

Setting Behavioral Attributes

`setEditable` (page 1523)
Controls whether the receiver allows the user to edit its text.

`isEditable` (page 1516)
Returns `true` if the receiver allows the user to edit text, `false` if it doesn't.

`setSelectable` (page 1525)
Controls whether the receiver allows the user to select its text.

`isSelectable` (page 1518)
Returns `true` if the receiver allows the user to select text, `false` if it doesn't.

`setFieldEditor` (page 1523)
Controls whether the receiver interprets Tab, Shift-Tab, and Return (Enter) as cues to end editing and possibly to change the first responder.

`isFieldEditor` (page 1517)
Returns `true` if the receiver interprets Tab, Shift-Tab, and Return (Enter) as cues to end editing and possibly to change the first responder; `false` if it accepts them as text input.

`setRichText` (page 1525)
Controls whether the receiver allows the user to apply attributes to specific ranges of the text.

`isRichText` (page 1517)
Returns `true` if the receiver allows the user to apply attributes to specific ranges of the text, `false` if it doesn't.

`setImportsGraphics` (page 1524)
Controls whether the receiver allows the user to import files by dragging.

`importsGraphics` (page 1516)
Returns `true` if the receiver allows the user to import files by dragging, `false` if it doesn't.

Using the Font Panel and Menu

`setUsesFontPanel` (page 1527)
Controls whether the receiver uses the Font panel and Font menu.

`usesFontPanel` (page 1529)
Returns `true` if the receiver uses the Font panel, `false` otherwise.

Using the Ruler

`toggleRuler` (page 1529)
This action method shows or hides the ruler, if the receiver is enclosed in a scroll view.

`isRulerVisible` (page 1518)
Returns `true` if the receiver's enclosing scroll view shows its ruler, `false` otherwise.

Changing the Selection

`setSelectedRange` (page 1526)
Selects the receiver's characters within *`aRange`*.

`selectedRange` (page 1522)
Returns the range of selected characters.

Replacing Text

`replaceCharactersInRangeWithRTF` (page 1520)
: Replaces the characters in *aRange* with RTF text interpreted from *rtfData*.

`replaceCharactersInRangeWithRTFD` (page 1520)
: Replaces the characters in *aRange* with RTFD text interpreted from *rtfdData*.

`replaceCharactersInRange` (page 1520)
: Replaces the characters in *aRange* with *aString*.

`setString` (page 1526)
: Replaces the receiver's entire text with *aString*, applying the formatting attributes of the old first character to its new contents.

Action Methods for Editing

`selectAll` (page 1521)
: This action method selects all of the receiver's text.

`copy` (page 1514)
: This action method copies the selected text onto the general pasteboard, in as many formats as the receiver supports.

`cut` (page 1515)
: This action method deletes the selected text and places it onto the general pasteboard, in as many formats as the receiver supports.

`paste` (page 1519)
: This action method pastes text from the general pasteboard at the insertion point or over the selection.

`copyFont` (page 1514)
: This action method copies the font information for the first character of the selection (or for the insertion point) onto the font pasteboard, as `NSPasteboard.FontPboardType`.

`pasteFont` (page 1519)
: This action method pastes font information from the font pasteboard onto the selected text or insertion point of a rich text object, or over all text of a plain text object.

`copyRuler` (page 1515)
: This action method copies the paragraph style information for first selected paragraph onto the ruler pasteboard, as `NSPasteboard.RulerPboardType`, and expands the selection to paragraph boundaries.

`pasteRuler` (page 1519)
: This action method pastes paragraph style information from the ruler pasteboard onto the selected paragraphs of a rich text object.

`delete` (page 1515)
: This action method deletes the selected text.

Changing the Font

`changeFont` (page 1513)
: This action method changes the font of the selection for a rich text object, or of all text for a plain text object.

`setFont` (page 1524)
: Sets the font of all the receiver's text to *aFont*.

`setFontInRange` (page 1524)
: Sets the font of characters within *aRange* to *aFont*.

`font` (page 1516)
: Returns the font of the first character in the receiver's text, or of the insertion point if there's no text.

Setting Text Alignment

`setAlignment` (page 1522)
: Sets the alignment of all the receiver's text to *mode*.

`alignCenter` (page 1512)
: This action method applies center alignment to selected paragraphs (or all text if the receiver is a plain text object).

`alignLeft` (page 1512)
: This action method applies left alignment to selected paragraphs (or all text if the receiver is a plain text object).

`alignRight` (page 1513)
: This action method applies right alignment to selected paragraphs (or all text if the receiver is a plain text object).

`alignment` (page 1513)
: Returns the alignment of the first selected paragraph, or of all text for a plain text object.

Setting Text Color

`setTextColor` (page 1526)
: Sets the text color of all characters in the receiver to *aColor*.

`setTextColorInRange` (page 1526)
: Sets the text color of characters within *aRange* to *aColor*.

`textColor` (page 1529)
: Returns the color of the receiver's first character, or for the insertion point if there's no text.

Setting Superscripting and Subscripting

`superscript` (page 1528)
: This action method applies a superscript attribute to selected text (or all text if the receiver is a plain text object), raising its baseline offset by a predefined amount.

`subscript` (page 1528)
: This action method applies a subscript attribute to selected text (or all text if the receiver is a plain text object), lowering its baseline offset by a predefined amount.

`unscript` (page 1529)
: This action method removes any superscripting or subscripting from selected text (or all text if the receiver is a plain text object).

Underlining Text

`underline` (page 1529)
: This action method underlines selected text for a rich text object, or all text for a plain text object.

Reading and Writing RTF Files

`readRTFDFromFile` (page 1519)
: Attempts to read the RTFD file at *path*, returning `true` if successful and `false` if not.

`writeRTFDToFile` (page 1529)
: Writes the receiver's text as RTF with attachments to a file or directory at *path*.

`RTFDFromRange` (page 1521)
: Returns an NSData object that contains an RTFD stream corresponding to the characters and attributes within *aRange*.

`RTFFromRange` (page 1521)
: Returns an NSData object that contains an RTF stream corresponding to the characters and attributes within *aRange*, omitting any attachment characters and attributes.

Checking Spelling

`checkSpelling` (page 1514)
: This action method searches for a misspelled word in the receiver's text.

`changeSpelling` (page 1514)
: Replaces the selected word in the receiver with a corrected version from the Spelling panel.

`ignoreSpelling` (page 1516)
: This action method informs the receiver to ignore misspelled words on a document-by-document basis. This method is sent by the NSSpellChecker instance.

`showGuessPanel` (page 1527)
: This action method opens the Spelling panel, allowing the user to make a correction during spell checking.

Constraining Size

`setMaxSize` (page 1525)
: Sets the receiver's maximum size to *aSize*.

`maxSize` (page 1518)
: Returns the receiver's maximum size.

`setMinSize` (page 1525)
: Sets the receiver's minimum size to *aSize*.

`minSize` (page 1518)
: Returns the receiver's minimum size.

`setVerticallyResizable` (page 1527)
: Controls whether the receiver changes its height to fit the height of its text.

`isVerticallyResizable` (page 1518)
: Returns `true` if the receiver automatically changes its height to accommodate the height of its text, `false` if it doesn't.

`setHorizontallyResizable` (page 1524)
: Controls whether the receiver changes its width to fit the width of its text.

`isHorizontallyResizable` (page 1517)
: Returns `true` if the receiver automatically changes its width to accommodate the width of its text, `false` if it doesn't.

`sizeToFit` (page 1528)
: Resizes the receiver to fit its text.

Scrolling

`scrollRangeToVisible` (page 1521)
: Scrolls the receiver in its enclosing scroll view so the first characters of *`aRange`* are visible.

Setting the Delegate

`setDelegate` (page 1523)
: Sets the receiver's delegate to *`anObject`*, without retaining it.

`delegate` (page 1515)
: Returns the receiver's delegate, or `null` if it has none.

Editing text

`textShouldBeginEditing` (page 1532) *delegate method*
: Invoked from a text object's implementation of `becomeFirstResponder` (page 1190), this method requests permission for *`aTextObject`* to begin editing.

`textDidBeginEditing` (page 1531) *delegate method*
: Informs the delegate that the text object has begun editing (that the user has begun changing it).

`textShouldEndEditing` (page 1532) *delegate method*
: Invoked from a text object's implementation of `resignFirstResponder` (page 1196), this method requests permission for *`aTextObject`* to end editing.

`textDidEndEditing` (page 1532) *delegate method*
: Informs the delegate that the text object has finished editing (that it has resigned first responder status).

Changing text formatting

`textDidChange` (page 1531) *delegate method*
: Informs the delegate that the text object has changed its characters or formatting attributes.

Constructors

NSText

Creates an NSText with a zero-sized frame rectangle.

```
public NSText()
```

Creates a text object with *frameRect* as its frame rectangle.

```
public NSText(NSRect frameRect)
```

Discussion
This method actually substitutes an instance of a concrete subclass of NSText, such as NSTextView, and configures that instance to archive itself.

Instance Methods

alignCenter

This action method applies center alignment to selected paragraphs (or all text if the receiver is a plain text object).

```
public void alignCenter(Object sender)
```

See Also
`alignLeft` (page 1512)
`alignRight` (page 1513)
`alignment` (page 1513)
`setAlignment` (page 1522)

alignLeft

This action method applies left alignment to selected paragraphs (or all text if the receiver is a plain text object).

```
public void alignLeft(Object sender)
```

See Also
`alignCenter` (page 1512)
`alignRight` (page 1513)
`alignment` (page 1513)
`setAlignment` (page 1522)

alignment

Returns the alignment of the first selected paragraph, or of all text for a plain text object.

```
public int alignment()
```

Discussion

This value is one of the alignments described in "Constants" (page 1530).

Text using `NaturalTextAlignment` is actually displayed using one of the other alignments, depending on the natural alignment of the text's script.

alignRight

This action method applies right alignment to selected paragraphs (or all text if the receiver is a plain text object).

```
public void alignRight(Object sender)
```

See Also

`alignLeft` (page 1512)
`alignCenter` (page 1512)
`alignment` (page 1513)
`setAlignment` (page 1522)

backgroundColor

Returns the receiver's background color.

```
public NSColor backgroundColor()
```

See Also

`drawsBackground` (page 1516)
`setBackgroundColor` (page 1522)

changeFont

This action method changes the font of the selection for a rich text object, or of all text for a plain text object.

```
public void changeFont(Object sender)
```

Discussion

If the receiver doesn't use the Font panel, this method does nothing.

This method changes the font by sending a `convertFont` (page 674) message to the shared NSFontManager and applying each NSFont returned to the appropriate text. See the NSFontManager (page 667) class specification for more information on font conversion.

See Also

`usesFontPanel` (page 1529)

changeSpelling

Replaces the selected word in the receiver with a corrected version from the Spelling panel.

```
public void changeSpelling(Object sender)
```

Discussion

This message is sent by the NSSpellChecker to the object whose text is being checked. To get the corrected spelling, ask *sender* for the string value of its selected cell (visible to the user as the text field in the Spelling panel). This method should replace the selected portion of the text with the string that it gets from the NSSpellChecker.

checkSpelling

This action method searches for a misspelled word in the receiver's text.

```
public void checkSpelling(Object sender)
```

Discussion

The search starts at the end of the 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, a `showGuessPanel` (page 1527) message then opens the Guess panel and allows the user to make a correction or add the word to the local dictionary.

See Also

`showGuessPanel` (page 1527)

copy

This action method copies the selected text onto the general pasteboard, in as many formats as the receiver supports.

```
public void copy(Object sender)
```

Discussion

A plain text object uses `NSPasteboard.StringPboardType` for plain text, and a rich text object also uses `NSPasteboard.RTFPboardType`.

See Also

`copyFont` (page 1514)

`copyRuler` (page 1515)

`cut` (page 1515)

`paste` (page 1519)

copyFont

This action method copies the font information for the first character of the selection (or for the insertion point) onto the font pasteboard, as `NSPasteboard.FontPboardType`.

```
public void copyFont(Object sender)
```

See Also

`copy` (page 1514)

copyRuler (page 1515)
cut (page 1515)
paste (page 1519)

copyRuler

This action method copies the paragraph style information for first selected paragraph onto the ruler pasteboard, as `NSPasteboard.RulerPboardType`, and expands the selection to paragraph boundaries.

```
public void copyRuler(Object sender)
```

See Also

copy (page 1514)
copyFont (page 1514)
cut (page 1515)
paste (page 1519)

cut

This action method deletes the selected text and places it onto the general pasteboard, in as many formats as the receiver supports.

```
public void cut(Object sender)
```

Discussion

A plain text object uses `NSPasteboard.StringPboardType` for plain text, and a rich text object also uses `NSPasteboard.RTFPboardType`.

See Also

delete (page 1515)
copy (page 1514)
copyFont (page 1514)
copyRuler (page 1515)
paste (page 1519)

delegate

Returns the receiver's delegate, or `null` if it has none.

```
public Object delegate()
```

See Also

setDelegate (page 1523)

delete

This action method deletes the selected text.

```
public void delete(Object sender)
```

See Also

`cut` (page 1515)

drawsBackground

Returns `true` if the receiver draws its background, `false` if it doesn't.

```
public boolean drawsBackground()
```

See Also

`backgroundColor` (page 1513)

`setDrawsBackground` (page 1523)

font

Returns the font of the first character in the receiver's text, or of the insertion point if there's no text.

```
public NSFont font()
```

See Also

`setFont` (page 1524)

`setFontInRange` (page 1524)

ignoreSpelling

This action method informs the receiver to ignore misspelled words on a document-by-document basis. This method is sent by the NSSpellChecker instance.

```
public void ignoreSpelling(Object sender)
```

importsGraphics

Returns `true` if the receiver allows the user to import files by dragging, `false` if it doesn't.

```
public boolean importsGraphics()
```

Discussion

A text object that accepts dragged files is also a rich text object.

See Also

`isRichText` (page 1517)

`setImportsGraphics` (page 1524)

isEditable

Returns `true` if the receiver allows the user to edit text, `false` if it doesn't.

```
public boolean isEditable()
```

Discussion
You can change the receiver's text programmatically regardless of this setting.

If the receiver is editable, it's also selectable.

See Also
`isSelectable` (page 1518)
`setEditable` (page 1523)

isFieldEditor

Returns `true` if the receiver interprets Tab, Shift-Tab, and Return (Enter) as cues to end editing and possibly to change the first responder; `false` if it accepts them as text input.

```
public boolean isFieldEditor()
```

Discussion
See the NSWindow (page 1795) class specification for more information on field editors. By default, NSText objects don't behave as field editors.

See Also
`setFieldEditor` (page 1523)

isHorizontallyResizable

Returns `true` if the receiver automatically changes its width to accommodate the width of its text, `false` if it doesn't.

```
public boolean isHorizontallyResizable()
```

Discussion
By default, an NSText object is not horizontally resizable.

See Also
`isVerticallyResizable` (page 1518)
`setHorizontallyResizable` (page 1524)

isRichText

Returns `true` if the receiver allows the user to apply attributes to specific ranges of the text, `false` if it doesn't.

```
public boolean isRichText()
```

See Also
`importsGraphics` (page 1516)
`setRichText` (page 1525)

isRulerVisible

Returns `true` if the receiver's enclosing scroll view shows its ruler, `false` otherwise.

```
public boolean isRulerVisible()
```

See Also
`toggleRuler` (page 1529)

isSelectable

Returns `true` if the receiver allows the user to select text, `false` if it doesn't.

```
public boolean isSelectable()
```

See Also
`isEditable` (page 1516)
`setSelectable` (page 1525)

isVerticallyResizable

Returns `true` if the receiver automatically changes its height to accommodate the height of its text, `false` if it doesn't.

```
public boolean isVerticallyResizable()
```

Discussion
By default, an NSText object is vertically resizable.

See Also
`isHorizontallyResizable` (page 1517)
`setVerticallyResizable` (page 1527)

maxSize

Returns the receiver's maximum size.

```
public NSSize maxSize()
```

See Also
`minSize` (page 1518)
`setMaxSize` (page 1525)

minSize

Returns the receiver's minimum size.

```
public NSSize minSize()
```

See Also

maxSize (page 1518)

setMinSize (page 1525)

paste

This action method pastes text from the general pasteboard at the insertion point or over the selection.

```
public void paste(Object sender)
```

See Also

copy (page 1514)

cut (page 1515)

pasteFont (page 1519)

pasteRuler (page 1519)

pasteFont

This action method pastes font information from the font pasteboard onto the selected text or insertion point of a rich text object, or over all text of a plain text object.

```
public void pasteFont(Object sender)
```

See Also

copyFont (page 1514)

pasteRuler (page 1519)

pasteRuler

This action method pastes paragraph style information from the ruler pasteboard onto the selected paragraphs of a rich text object.

```
public void pasteRuler(Object sender)
```

Discussion

It doesn’t apply to a plain text object.

See Also

copyFont (page 1514)

pasteRuler (page 1519)

readRTFDFromFile

Attempts to read the RTFD file at `path`, returning `true` if successful and `false` if not.

```
public boolean readRTFDFromFile(String path)
```

Discussion

`path` should be the path for an `.rtf` file or an `.rtfd` file wrapper, not for the RTF file within an `.rtfd` file wrapper.

See Also

`writeRTFDToFile` (page 1529)

replaceCharactersInRange

Replaces the characters in *aRange* with *aString*.

```
public void replaceCharactersInRange(NSRange aRange, String aString)
```

Discussion

For a rich text object, the text of *aString* is assigned the formatting attributes of the first character of the text it replaces, or of the character immediately before *aRange* if the range's length is 0. If the range's location is 0, the formatting attributes of the first character in the receiver are used.

This method does not include undo support by default. Clients must invoke `shouldChangeTextInRanges` (page 1656) or `shouldChangeTextInRange` (page 1655) to include this method in an undoable action.

See Also

`replaceCharactersInRangeWithRTF` (page 1520)
`replaceCharactersInRangeWithRTFD` (page 1520)

replaceCharactersInRangeWithRTF

Replaces the characters in *aRange* with RTF text interpreted from *rtfData*.

```
public void replaceCharactersInRangeWithRTF(NSRange aRange, NSData rtfData)
```

Discussion

This method applies only to rich text objects.

This method does not include undo support by default. Clients must invoke `shouldChangeTextInRanges` (page 1656) or `shouldChangeTextInRange` (page 1655) to include this method in an undoable action.

See Also

`replaceCharactersInRangeWithRTFD` (page 1520)
`replaceCharactersInRange` (page 1520)

replaceCharactersInRangeWithRTFD

Replaces the characters in *aRange* with RTFD text interpreted from *rtfdData*.

```
public void replaceCharactersInRangeWithRTFD(NSRange aRange, NSData rtfdData)
```

Discussion

This method applies only to rich text objects.

This method does not include undo support by default. Clients must invoke shouldChangeTextInRanges (page 1656) or shouldChangeTextInRange (page 1655) to include this method in an undoable action.

See Also

replaceCharactersInRangeWithRTF (page 1520)

replaceCharactersInRange (page 1520)

RTFDFromRange

Returns an NSData object that contains an RTFD stream corresponding to the characters and attributes within *aRange*.

```
public NSData RTFDFromRange(NSRange aRange)
```

Discussion

Throws a RangeException if any part of *aRange* lies beyond the end of the receiver's characters.

When writing data to the pasteboard, you can use the NSData object as the first argument to NSPasteboard's setDataForType (page 1077) method, with a second argument of NSPasteboard.RTFDPboardType.

See Also

RTFFromRange (page 1521)

RTFFromRange

Returns an NSData object that contains an RTF stream corresponding to the characters and attributes within *aRange*, omitting any attachment characters and attributes.

```
public NSData RTFFromRange(NSRange aRange)
```

Discussion

Throws a RangeException if any part of *aRange* lies beyond the end of the receiver's characters.

When writing data to the pasteboard, you can use the NSData object as the first argument to NSPasteboard's setDataForType (page 1077) method, with a second argument of NSPasteboard.RTFPboardType.

See Also

RTFDFromRange (page 1521)

scrollRangeToVisible

Scrolls the receiver in its enclosing scroll view so the first characters of *aRange* are visible.

```
public void scrollRangeToVisible(NSRange aRange)
```

selectAll

This action method selects all of the receiver's text.

```
public void selectAll(Object sender)
```

selectedRange

Returns the range of selected characters.

```
public NSRange selectedRange()
```

See Also

`setSelectedRange` (page 1526)

setAlignment

Sets the alignment of all the receiver's text to *mode*.

```
public void setAlignment(int mode)
```

Discussion

mode may be one of the alignments described in "Constants" (page 1530).

Text using `NaturalTextAlignment` is actually displayed using one of the other alignments, depending on the natural alignment of the text's script.

This method does not include undo support by default. Clients must invoke `shouldChangeTextInRanges` (page 1656) or `shouldChangeTextInRange` (page 1655) to include this method in an undoable action.

See Also

`alignment` (page 1513)

`alignLeft` (page 1512)

`alignCenter` (page 1512)

`alignRight` (page 1513)

setBackgroundColor

Sets the receiver's background color to *aColor*.

```
public void setBackgroundColor(NSColor aColor)
```

Discussion

This method does not include undo support by default. Clients must invoke `shouldChangeTextInRanges` (page 1656) or `shouldChangeTextInRange` (page 1655) to include this method in an undoable action.

See Also

`setDrawsBackground` (page 1523)

`backgroundColor` (page 1513)

setDelegate

Sets the receiver's delegate to *anObject*, without retaining it.

```
public void setDelegate(Object anObject)
```

See Also

delegate (page 1515)

setDrawsBackground

Controls whether the receiver draws its background.

```
public void setDrawsBackground(boolean flag)
```

Discussion

If *flag* is true, the receiver fills its background with the background color; if *flag* is false, it doesn't.

See Also

setBackgroundColor (page 1522)

drawsBackground (page 1516)

setEditable

Controls whether the receiver allows the user to edit its text.

```
public void setEditable(boolean flag)
```

Discussion

If *flag* is true, the receiver allows the user to edit text and attributes; if *flag* is false, it doesn't. You can change the receiver's text programmatically regardless of this setting. If the receiver is made editable, it's also made selectable. NSText objects are by default editable.

See Also

setSelectable (page 1525)

isEditable (page 1516)

setFieldEditor

Controls whether the receiver interprets Tab, Shift-Tab, and Return (Enter) as cues to end editing and possibly to change the first responder.

```
public void setFieldEditor(boolean flag)
```

Discussion

If *flag* is true, it does; if *flag* is false, it doesn't, instead accepting these characters as text input. See the NSWindow (page 1795) class specification for more information on field editors. By default, NSText objects don't behave as field editors.

See Also

isFieldEditor (page 1517)

setFont

Sets the font of all the receiver's text to *aFont*.

```
public void setFont(NSFont aFont)
```

Discussion

This method does not include undo support by default. Clients must invoke `shouldChangeTextInRanges` (page 1656) or `shouldChangeTextInRange` (page 1655) to include this method in an undoable action.

See Also

`setFontInRange` (page 1524)
`font` (page 1516)

setFontInRange

Sets the font of characters within *aRange* to *aFont*.

```
public void setFontInRange(NSFont aFont, NSRange aRange)
```

Discussion

This method applies only to a rich text object.

This method does not include undo support by default. Clients must invoke `shouldChangeTextInRanges` (page 1656) or `shouldChangeTextInRange` (page 1655) to include this method in an undoable action.

See Also

`setFont` (page 1524)
`font` (page 1516)

setHorizontallyResizable

Controls whether the receiver changes its width to fit the width of its text.

```
public void setHorizontallyResizable(boolean flag)
```

Discussion

If *flag* is `true` it does; if *flag* is `false` it doesn't.

See Also

`setVerticallyResizable` (page 1527)
`isHorizontallyResizable` (page 1517)

setImportsGraphics

Controls whether the receiver allows the user to import files by dragging.

```
public void setImportsGraphics(boolean flag)
```

Discussion

If `flag` is `true`, it does; if `flag` is `false`, it doesn't. If the receiver is set to accept dragged files, it's also made a rich text object. Subclasses may or may not accept dragged files by default.

See Also

`setRichText` (page 1525)

`importsGraphics` (page 1516)

setMaxSize

Sets the receiver's maximum size to *aSize*.

```
public void setMaxSize(NSSize aSize)
```

See Also

`setMinSize` (page 1525)

`maxSize` (page 1518)

setMinSize

Sets the receiver's minimum size to *aSize*.

```
public void setMinSize(NSSize aSize)
```

See Also

`setMaxSize` (page 1525)

`minSize` (page 1518)

setRichText

Controls whether the receiver allows the user to apply attributes to specific ranges of the text.

```
public void setRichText(boolean flag)
```

Discussion

If `flag` is `true` it does; if `flag` is `false` it doesn't. If `flag` is `false`, the receiver is also set not to accept dragged files. Subclasses may or may not let the user apply multiple attributes to the text and accept drag files by default.

See Also

`isRichText` (page 1517)

`setImportsGraphics` (page 1524)

setSelectable

Controls whether the receiver allows the user to select its text.

```
public void setSelectable(boolean flag)
```

Discussion
If *flag* is `true`, the receiver allows the user to select text; if *flag* is `false`, it doesn't. You can set selections programmatically regardless of this setting. If the receiver is made not selectable, it's also made not editable. NSText objects are by default editable and selectable.

See Also
`setEditable` (page 1523)
`isSelectable` (page 1518)

setSelectedRange

Selects the receiver's characters within *aRange*.

```
public void setSelectedRange(NSRange aRange)
```

See Also
`selectedRange` (page 1522)

setString

Replaces the receiver's entire text with *aString*, applying the formatting attributes of the old first character to its new contents.

```
public void setString(String aString)
```

Discussion
This method does not include undo support by default. Clients must invoke `shouldChangeTextInRanges` (page 1656) or `shouldChangeTextInRange` (page 1655) to include this method in an undoable action.

setTextColor

Sets the text color of all characters in the receiver to *aColor*.

```
public void setTextColor(NSColor aColor)
```

Discussion
Removes the text color attribute if *aColor* is `null`.

This method does not include undo support by default. Clients must invoke `shouldChangeTextInRanges` (page 1656) or `shouldChangeTextInRange` (page 1655) to include this method in an undoable action.

See Also
`setTextColorInRange` (page 1526)
`textColor` (page 1529)

setTextColorInRange

Sets the text color of characters within *aRange* to *aColor*.

```
public void setTextColorInRange(NSColor aColor, NSRange aRange)
```

Discussion
Removes the text color attribute if `aColor` is `null`. This method applies only to rich text objects.

This method does not include undo support by default. Clients must invoke `shouldChangeTextInRanges` (page 1656) or `shouldChangeTextInRange` (page 1655) to include this method in an undoable action.

See Also
`setTextColor` (page 1526)
`textColor` (page 1529)

setUsesFontPanel

Controls whether the receiver uses the Font panel and Font menu.

```
public void setUsesFontPanel(boolean flag)
```

Discussion
If `flag` is `true`, the receiver responds to messages from the Font panel and from the Font menu and updates the Font panel with the selection font whenever it changes. If `flag` is `false` the receiver doesn't do any of these actions. By default, an NSText object uses the Font panel and menu.

See Also
`usesFontPanel` (page 1529)

setVerticallyResizable

Controls whether the receiver changes its height to fit the height of its text.

```
public void setVerticallyResizable(boolean flag)
```

Discussion
If `flag` is `true` it does; if `flag` is `false` it doesn't.

See Also
`setHorizontallyResizable` (page 1524)
`isVerticallyResizable` (page 1518)

showGuessPanel

This action method opens the Spelling panel, allowing the user to make a correction during spell checking.

```
public void showGuessPanel(Object sender)
```

See Also
`checkSpelling` (page 1514)

sizeToFit

Resizes the receiver to fit its text.

```
public void sizeToFit()
```

Discussion

The text view will not be sized any smaller than its minimum size, however.

See Also

`isHorizontallyResizable` (page 1517)
`isVerticallyResizable` (page 1518)

string

Returns the characters of the receiver's text.

```
public String string()
```

Discussion

For performance reasons, this method returns the current backing store of the text object. If you want to maintain a snapshot of this as you manipulate the text storage, you should make a copy of the appropriate substring.

See Also

`setString` (page 1526)

subscript

This action method applies a subscript attribute to selected text (or all text if the receiver is a plain text object), lowering its baseline offset by a predefined amount.

```
public void subscript(Object sender)
```

See Also

`subscript` (page 1528)
`unscript` (page 1529)
`lowerBaseline` (page 1633) (NSTextView)

superscript

This action method applies a superscript attribute to selected text (or all text if the receiver is a plain text object), raising its baseline offset by a predefined amount.

```
public void superscript(Object sender)
```

See Also

`subscript` (page 1528)
`unscript` (page 1529)
`raiseBaseline` (page 1636) (NSTextView)

textColor

Returns the color of the receiver's first character, or for the insertion point if there's no text.

```
public NSColor textColor()
```

See Also

`setTextColor` (page 1526)

`setTextColorInRange` (page 1526)

toggleRuler

This action method shows or hides the ruler, if the receiver is enclosed in a scroll view.

```
public void toggleRuler(Object sender)
```

underline

This action method underlines selected text for a rich text object, or all text for a plain text object.

```
public void underline(Object sender)
```

unscript

This action method removes any superscripting or subscripting from selected text (or all text if the receiver is a plain text object).

```
public void unscript(Object sender)
```

See Also

`subscript` (page 1528)

`superscript` (page 1528)

`raiseBaseline` (page 1636) (NSTextView)

`lowerBaseline` (page 1633) (NSTextView)

usesFontPanel

Returns `true` if the receiver uses the Font panel, `false` otherwise.

```
public boolean usesFontPanel()
```

See Also

`setUsesFontPanel` (page 1527)

writeRTFDToFile

Writes the receiver's text as RTF with attachments to a file or directory at *path*.

```
public boolean writeRTFDToFile(String path, boolean atomicFlag)
```

Discussion

Returns true on success and false on failure. If *atomicFlag* is true, attempts to write the file safely so that an existing file at *path* is not overwritten, nor does a new file at *path* actually get created, unless the write is successful.

See Also

RTFFromRange (page 1521)
RTFDFromRange (page 1521)
readRTFDFromFile (page 1519)

Constants

These constants specify text alignment:

Constant	Description
LeftTextAlignment	Text is visually left aligned.
RightTextAlignment	Text is visually right aligned.
CenterTextAlignment	Text is visually center aligned.
JustifiedTextAlignment	Text is justified.
NaturalTextAlignment	Use the natural alignment of the text's script.

These constants specify the reason for a change of editing focus among text fields, in essence answering the question "why am I leaving the field?" They are the possible values for the TextMovement key of the TextDidEndEditingNotification (page 1533) userInfo dictionary. The field editor makes sure that these are the values sent when the user presses the Tab, Backtab, or Return key while editing. The control then uses this information to decide where to send focus next.

Constant	Description
IllegalTextMovement	Currently unused.
ReturnTextMovement	The Return key was pressed.
TabTextMovement	The Tab key was pressed.
BacktabTextMovement	The Backtab (Shift-Tab) key was pressed.
LeftTextMovement	The left arrow key was pressed.
RightTextMovement	The right arrow key was pressed.
UpTextMovement	The up arrow key was pressed.
DownTextMovement	The down arrow key was pressed.
CancelTextMovement	The user cancelled the completion.

Constant	Description
`OtherTextMovement`	The user performed some undefined action.

These constants specify several commonly used Unicode characters.

Constant	Description
`ParagraphSeparatorCharacter`	The paragraph separator character: `0x2029`
`LineSeparatorCharacter`	The line separator character: `0x2028`
`TabCharacter`	The tab character: `0x0009`
`BackTabCharacter`	The back tab character: `0x0019`
`FormFeedCharacter`	The form feed character: `0x000c`
`NewlineCharacter`	The newline character: `0x000a`
`CarriageReturnCharacter`	The carriage return character: `0x000d`
`EnterCharacter`	The enter character: `0x0003`
`BackspaceCharacter`	The backspace character: `0x0008`
`DeleteCharacter`	The delete character: `0x007f`

Delegate Methods

textDidBeginEditing

Informs the delegate that the text object has begun editing (that the user has begun changing it).

```
public abstract void textDidBeginEditing(NSNotification aNotification)
```

Discussion
The name of *aNotification* is `TextDidBeginEditingNotification` (page 1532).

textDidChange

Informs the delegate that the text object has changed its characters or formatting attributes.

```
public abstract void textDidChange(NSNotification aNotification)
```

Discussion
The name of *aNotification* is `TextDidChangeNotification` (page 1532).

textDidEndEditing

Informs the delegate that the text object has finished editing (that it has resigned first responder status).

```
public abstract void textDidEndEditing(NSNotification aNotification)
```

Discussion
The name of *aNotification* is TextDidEndEditingNotification (page 1533).

textShouldBeginEditing

Invoked from a text object's implementation of becomeFirstResponder (page 1190), this method requests permission for *aTextObject* to begin editing.

```
public abstract boolean textShouldBeginEditing(NSText aTextObject)
```

Discussion
If the delegate returns true, the text object proceeds to make changes. If the delegate returns false, the text object abandons the editing operation. This method is invoked whenever *aTextObject* attempts to become first responder. It is also invoked when the user drags and drops a file onto the text object.

See Also
makeFirstResponder (page 1841) (NSWindow)
becomeFirstResponder (page 1190) (NSResponder)

textShouldEndEditing

Invoked from a text object's implementation of resignFirstResponder (page 1196), this method requests permission for *aTextObject* to end editing.

```
public abstract boolean textShouldEndEditing(NSText aTextObject)
```

Discussion
If the delegate returns true, the text object proceeds to finish editing and resign first responder status. If the delegate returns false, the text object selects all of its text and remains the first responder.

See Also
resignFirstResponder (page 1196) (NSResponder)

Notifications

TextDidBeginEditingNotification

Posted when an NSText object begins any operation that changes characters or formatting attributes.

The notification object is the notifying NSText object. This notification does not contain a *userInfo* dictionary.

TextDidChangeNotification

Posted after an NSText object performs any operation that changes characters or formatting attributes.

The notification object is the notifying NSText object. This notification does not contain a `userInfo` dictionary.

TextDidEndEditingNotification

Posted when an NSText object completes any operation that changes characters or formatting attributes.

The notification object is the notifying NSText object. The `userInfo` dictionary contains the following information:

Key	Value
`"NSTextMovement"`	Possible movement code values are described in "Constants" (page 1530).

NSTextAttachment

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.application
Companion guides	Text System Overview Text Attachment Programming Topics for Cocoa

Overview

NSTextAttachment objects are used by the NSAttributedString class as the values for attachment attributes (stored in the attributed string under the key named `AttachmentAttributeName`). The objects you create with this class are referred to as text attachment objects, or when no confusion will result, as text attachments or merely attachments. See the NSAttributedString and NSTextView (page 1609) class specifications for general information on text attachments.

A text attachment object contains an NSFileWrapper (page 631), which in turn holds the contents of the attached file. It also uses a cell object conforming to the NSCellForTextAttachment (page 1935) interface to draw and handle mouse events. Most of the behavior of a text attachment is relegated to the file wrapper and the attachment cell. See the corresponding class and interface specifications for more information.

Tasks

Constructors

`NSTextAttachment` (page 1536)
Creates an empty NSTextAttachment.

Setting the File Wrapper

`setFileWrapper` (page 1537)
Sets the receiver's file wrapper, which holds the contents of the attached file, to *`aWrapper`*.

`fileWrapper` (page 1537)
Returns the receiver's file wrapper, which holds the contents of the attached file.

Setting the Attachment Cell

`setAttachmentCell` (page 1537)
Sets the object used to draw the icon for the receiver and to handle mouse events to *aCell*.

`attachmentCell` (page 1536)
Returns the object used to draw the icon for the receiver and to handle mouse events.

Constructors

NSTextAttachment

Creates an empty NSTextAttachment.

```
public NSTextAttachment()
```

Creates an NSTextAttachment to contain *aWrapper* and use an NSTextAttachmentCell as its attachment cell.

```
public NSTextAttachment(NSFileWrapper aWrapper)
```

Discussion
If *aWrapper* contains an image file that the receiver can interpret as an NSImage object, sets the attachment cell's image to the NSImage rather than to the icon of *aWrapper*.

See Also
`setFileWrapper` (page 1537)
`setAttachmentCell` (page 1537)

Instance Methods

attachmentCell

Returns the object used to draw the icon for the receiver and to handle mouse events.

```
public NSCellForTextAttachment attachmentCell()
```

Discussion
An NSTextAttachment by default uses an NSTextAttachmentCell that displays the attached file's icon, or its contents if the file contains an image.

See Also
`fileWrapper` (page 1537)
`image` (page 313) (NSCell)
`icon` (page 636) (NSFileWrapper)
`setAttachmentCell` (page 1537)

fileWrapper

Returns the receiver's file wrapper, which holds the contents of the attached file.

```
public NSFileWrapper fileWrapper()
```

See Also

`setFileWrapper` (page 1537)

setAttachmentCell

Sets the object used to draw the icon for the receiver and to handle mouse events to *`aCell`*.

```
public void setAttachmentCell(NSCellForTextAttachment aCell)
```

See Also

`setFileWrapper` (page 1537)
`setImage` (page 327) (NSCell)
`icon` (page 636) (NSFileWrapper)
`attachmentCell` (page 1536)

setFileWrapper

Sets the receiver's file wrapper, which holds the contents of the attached file, to *`aWrapper`*.

```
public void setFileWrapper(NSFileWrapper aWrapper)
```

See Also

`fileWrapper` (page 1537)

Constants

The following constant is provided by NSTextAttachment:

Constant	Description
`AttachmentCharacter`	Denotes attachments.

NSTextAttachmentCell

Inherits from	NSCell : NSObject
Implements	NSCellForTextAttachment NSCoding (NSCell)
Package:	com.apple.cocoa.application
Companion guides	Text System Overview Text Attachment Programming Topics for Cocoa

Overview

NSTextAttachmentCell implements the functionality of the NSCellForTextAttachment interface.

Interfaces Implemented

NSCellForTextAttachment

- attachment (page 1936)
- cellBaselineOffset (page 1936)
- cellFrame (page 1936)
- cellSize (page 1937)
- drawWithFrameInView (page 1937)
- highlightWithFrameInView (page 1937)
- setAttachment (page 1938)
- trackMouse (page 1938)
- wantsToTrackMouse (page 1939)
- wantsToTrackMouseForEvent (page 1939)

Tasks

Constructors

NSTextAttachmentCell (page 1540)
Creates an empty NSTextAttachmentCell.

Drawing

`drawWithFrameInView` (page 1542)
Draws the receiver's image within `cellFrame` in `aView`, which should be the focus view.

`highlightWithFrameInView` (page 1542)
Draws the receiver's image—with highlighting if `flag` is `true`—within `cellFrame` in `aView`, which should be the focus view.

Cell Size and Position

`cellSize` (page 1541)
Returns the size of the attachment's icon.

`cellBaselineOffset` (page 1541)
Returns the position where the attachment cell's image should be drawn in text, relative to the current point established in the glyph layout.

`cellFrame` (page 1541)
Returns the frame of the cell as it would be drawn at `position` as the character at `charIndex` in the specified `textContainer` within the region of `lineFrag`.

Event Handling

`trackMouse` (page 1543)
Handles a mouse-down event on the receiver's image.

`wantsToTrackMouse` (page 1543)
Returns `true`.

`wantsToTrackMouseForEvent` (page 1544)
Allows an attachment to specify what events it would want to track the cursor for.

Setting the Attachment

`attachment` (page 1541)
Returns the text attachment object that owns the receiver.

`setAttachment` (page 1542)
Sets the text attachment object that owns the receiver to `anAttachment`.

Constructors

NSTextAttachmentCell

Creates an empty NSTextAttachmentCell.

```
public NSTextAttachmentCell()
```

Creates an NSAttachmentCell initialized with *aString* and set to have the cell's default menu.

```
public NSTextAttachmentCell(String aString)
```

Discussion
If no field editor has been created for all NSTextAttachmentCells, one is created.

Creates an NSTextAttachmentCell initialized with *anImage* and set to have the cell's default menu.

```
public NSTextAttachmentCell(NSImage anImage)
```

Discussion
If *anImage* is `null`, no image is set.

Instance Methods

attachment

Returns the text attachment object that owns the receiver.

```
public NSTextAttachment attachment()
```

See Also
`setAttachment` (page 1542)

cellBaselineOffset

Returns the position where the attachment cell's image should be drawn in text, relative to the current point established in the glyph layout.

```
public NSPoint cellBaselineOffset()
```

Discussion
The image should be drawn so its lower-left corner lies on this point.

See Also
`icon` (page 636) (NSFileWrapper)

cellFrame

Returns the frame of the cell as it would be drawn at *position* as the character at *charIndex* in the specified *textContainer* within the region of *lineFrag*.

```
public NSRect cellFrame(NSTextContainer textContainer, NSRect lineFrag, NSPoint
    position, int charIndex)
```

cellSize

Returns the size of the attachment's icon.

```
public NSSize cellSize()
```

See Also

icon (page 636) (NSFileWrapper)

fileWrapper (page 1537) (NSTextAttachment)

drawWithFrameInView

Draws the receiver's image within *cellFrame* in *aView*, which should be the focus view.

```
public void drawWithFrameInView(NSRect cellFrame, NSView aView)
```

See Also

drawWithFrameInView (page 310) (NSCell)

lockFocus (page 1759) (NSView)

Draws the receiver's image within *cellFrame* in *aView*, which is the view currently focused.

```
public void drawWithFrameInView(NSRect cellFrame, NSView aView, int charIndex)
```

Discussion

charIndex is the index of the attachment character within the text. The default implementation simply calls drawWithFrameInView(*cellFrame*, *aView*).

Draws the receiver's image within *cellFrame* in *aView*, which is the view currently focused.

```
public void drawWithFrameInView(NSRect cellFrame, NSView aView, int charIndex,
    NSLayoutManager layoutManager)
```

Discussion

charIndex is the index of the attachment character within the text. *layoutManager* is the layout manager for the text. The default implementation simply calls drawWithFrameInView(*cellFrame*, *aView*, *charIndex*).

highlightWithFrameInView

Draws the receiver's image—with highlighting if *flag* is true—within *cellFrame* in *aView*, which should be the focus view.

```
public void highlightWithFrameInView(boolean flag, NSRect cellFrame, NSView aView)
```

See Also

highlightWithFrameInView (page 312) (NSCell)

lockFocus (page 1759) (NSView)

setAttachment

Sets the text attachment object that owns the receiver to *anAttachment*.

```
public void setAttachment(NSTextAttachment anAttachment)
```

See Also

attachment (page 1541)

setAttachmentCell (page 1537) (NSTextAttachment)

trackMouse

Handles a mouse-down event on the receiver's image.

```
public boolean trackMouse(NSEvent theEvent, NSRect cellFrame, NSView aTextView,
    int charIndex, boolean flag)
```

Discussion

theEvent is the mouse-down event. *cellFrame* is the region of *aTextView* in which further mouse events should be tracked. *charIndex* is the position in the text at which this attachment appears. *aTextView* is the view that received the event. It's assumed to be an NSTextView and should be the focus view. If *flag* is true, the receiver tracks the cursor until a mouse-up event occurs; if *flag* is false, it stops tracking when a mouse-dragged event occurs outside of *cellFrame*. Returns true if the receiver successfully finished tracking the cursor (typically through a mouse-up event), false otherwise (such as when the cursor is dragged outside *cellFrame*).

NSTextAttachmentCell's implementation of this method calls upon the delegate of *aTextView* to handle the event. If *theEvent* is a mouse-up event for a double click, the text attachment cell sends the delegate a textViewDoubleClickedCell (page 1669) message and returns true. Otherwise, depending on whether the user clicks or drags the cell, it sends the delegate a textViewClickedCell (page 1667) or a textViewDraggedCell (page 1669) message and returns true. NSTextAttachmentCell's implementation returns false only if *flag* is false and the cursor is dragged outside of *cellFrame*. The delegate methods are invoked only if the delegate responds.

Handles a mouse-down event on the receiver's image.

```
public boolean trackMouse(NSEvent theEvent, NSRect cellFrame, NSView aTextView,
    boolean flag)
```

Discussion

NSTextAttachmentCell's implementation of this method calls upon the delegate of *aTextView* to handle the event. If *theEvent* concludes as a double click, the text attachment cell sends the delegate a textViewDoubleClickedCell (page 1669) message and returns true. Otherwise, depending on whether the user clicks or drags the cell, it sends the delegate a textViewClickedCell (page 1667) or a textViewDraggedCell (page 1669) message and returns true. NSTextAttachmentCell's implementation returns false only if *flag* is false and the cursor is dragged outside of *cellFrame*. The delegate methods are invoked only if the delegate can respond to them.

See Also

wantsToTrackMouse (page 1543)

trackMouse (page 336) (NSCell)

lockFocus (page 1759) (NSView)

wantsToTrackMouse

Returns true.

```
public boolean wantsToTrackMouse()
```

Discussion

NSTextAttachmentCell objects support dragging. An NSTextView invokes this method before sending `trackMouse` (page 336) to the text attachment cell.

A more static subclass might override this method to return `false`, which results in the attachment image behaving as any other glyph in the text and not allowing itself to be dragged or to perform a method on being clicked.

wantsToTrackMouseForEvent

Allows an attachment to specify what events it would want to track the cursor for.

```
public boolean wantsToTrackMouseForEvent(NSEvent theEvent, NSRect cellFrame, NSView
     aTextView, int flag)
```

Discussion

If `wantsToTrackMouse` (page 1543) returns `true`, this method allows the attachment to decide whether it wishes to do so for particular events. See `trackMouse` (page 1543) for a description of the arguments to this method.

NSTextBlock

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.application
Availability	Available in Mac OS X v10.4 and later.
Companion guides	Text System Overview Text Layout Programming Guide for Cocoa

Overview

NSTextBlock represents a block of text laid out in a subregion of the text container. Text blocks appear as attributes on paragraphs, as part of the paragraph style.

The most important subclass is NSTextTableBlock, which represents a block of text that appears as a cell in a table. The table itself is represented by a separate class, NSTextTable, which is referenced by all of its NSTextTableBlocks and which controls their sizing and positioning.

Tasks

Constructors

`NSTextBlock` (page 1547)

Working with Dimensions of Content

`setValue` (page 1550)
Sets the value type for a dimension of the text block.

`valueForDimension` (page 1551)
Returns the value of the text block dimension specified by *`dimension`*.

`valueTypeForDimension` (page 1551)
Returns the value type of the text block dimension specified by *`dimension`*.

`setContentWidth` (page 1550)
Sets the width of the text block.

`contentWidth` (page 1548)
Returns the width of the text block in points.

`contentWidthValueType` (page 1548)
Returns the type of value stored for the text block width.

Getting and Setting Margins, Borders, and Padding

`setWidthForLayer` (page 1551)
Sets the width of one or more edges of a specified layer of the text block.

`widthForLayer` (page 1552)
Returns the width of a *layer* for a given *edge* of the text block in points.

`widthValueTypeForLayer` (page 1552)
Returns the value type of a *layer* for a given *edge* of the text block.

Getting and Setting Alignment

`setVerticalAlignment` (page 1550)
Sets the vertical alignment of the text block to *alignment*.

`verticalAlignment` (page 1552)
Returns the vertical alignment of the text block.

Working with Color

`setBackgroundColor` (page 1549)
Sets the background color of the text block.

`backgroundColor` (page 1547)
Returns the background color of the text block.

`setBorderColor` (page 1549)
Sets the color of one or more borders of the text block.

`borderColorForEdge` (page 1547)
Returns the border color of the text block edge specified by *edge*.

Determining Size and Position of Text Block

`rectForLayoutAtPoint` (page 1549)
Called by the typesetter before the text block is laid out to return the rectangle within which glyphs should be laid out.

`boundsRectForContentRect` (page 1547)
Called by the typesetter after the text block is laid out to return the rectangle the text in the block actually occupies, including padding, borders, and margins.

Drawing Colors and Decorations

`drawBackgroundWithFrame` (page 1548)
Called by the layout manager to draw any colors and other decorations before the text is drawn.

Constructors

NSTextBlock

```
public NSTextBlock()
```

Discussion
The constructor for the NSTextBlock object.

Instance Methods

backgroundColor

Returns the background color of the text block.

```
public NSColor backgroundColor()
```

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

See Also
`setBackgroundColor` (page 1549)

borderColorForEdge

Returns the border color of the text block edge specified by *edge*.

```
public NSColor borderColorForEdge(int edge)
```

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

See Also
`setBorderColor` (page 1549)

boundsRectForContentRect

Called by the typesetter after the text block is laid out to return the rectangle the text in the block actually occupies, including padding, borders, and margins.

```
public NSRect boundsRectForContentRect(NSRect contentRect, NSRect rect,
    NSTextContainer textContainer, NSRange charRange)
```

Discussion

The *contentRect* is the actual rectangle in which the text was laid out, as determined by `rectForLayoutAtPoint` (page 1549). The *rect* is the initial rectangle in *textContainer* proposed by the typesetter in which to lay out the glyphs for the characters in *charRange*.

Availability

Available in Mac OS X v10.4 and later.

See Also

`rectForLayoutAtPoint` (page 1549)

contentWidth

Returns the width of the text block in points.

```
public float contentWidth()
```

Discussion

This is a convenience method that invokes `valueForDimension:Width`.

Availability

Available in Mac OS X v10.4 and later.

See Also

`setContentWidth` (page 1550)

`contentWidthValueType` (page 1548)

contentWidthValueType

Returns the type of value stored for the text block width.

```
public int contentWidthValueType()
```

Discussion

The value type is either an absolute value in points or a percentage.

Availability

Available in Mac OS X v10.4 and later.

See Also

`setContentWidth` (page 1550)

`contentWidth` (page 1548)

drawBackgroundWithFrame

Called by the layout manager to draw any colors and other decorations before the text is drawn.

```
public void drawBackgroundWithFrame(NSRect frameRect, NSView controlView, NSRange
     charRange, NSLayoutManager layoutManager)
```

Discussion

The *frameRect* is the bounds rectangle in view coordinates. The *controlView* is the view in which drawing occurs. The *charRange* describes the characters whose glyphs are to be drawn, and the *layoutManager* is the layout manager controlling the typesetting.

Availability

Available in Mac OS X v10.4 and later.

rectForLayoutAtPoint

Called by the typesetter before the text block is laid out to return the rectangle within which glyphs should be laid out.

```
public NSRect rectForLayoutAtPoint(NSPoint startingPoint, NSRect rect,
    NSTextContainer textContainer, NSRange charRange)
```

Discussion

The *startingPoint* argument specifies the location, in container coordinates, where layout begins. The *rect* is the rectangle in which the block is constrained to lie: for top-level blocks, the container rectangle of *textContainer*; for nested blocks, the layout rectangle of the enclosing block. The *charRange* argument is the range of the characters to be laid out.

Availability

Available in Mac OS X v10.4 and later.

See Also

boundsRectForContentRect (page 1547)

setBackgroundColor

Sets the background color of the text block.

```
public void setBackgroundColor(NSColor color)
```

Availability

Available in Mac OS X v10.4 and later.

See Also

backgroundColor (page 1547)

setBorderColor

Sets the color of one or more borders of the text block.

```
public void setBorderColor(NSColor color)
```

Sets the color of one or more borders of the text block.

```
public void setBorderColor(NSColor color, int edge)
```

Discussion

If you specify an *edge* parameter, the color is applied to the specified edge; otherwise, it is applied to all edges. This setting has no visible effect unless the border width is larger than the default, which is 0.

Availability

Available in Mac OS X v10.4 and later.

See Also

`borderColorForEdge` (page 1547)
`setWidthForLayer` (page 1551)

setContentWidth

Sets the width of the text block.

```
public void setContentWidth(float val, int type)
```

Discussion

The value *val* is either an absolute value in points or a percentage of the enclosing block, as specified by *type*.

Availability

Available in Mac OS X v10.4 and later.

See Also

`contentWidth` (page 1548)
`contentWidthValueType` (page 1548)

setValue

Sets the value type for a dimension of the text block.

```
public void setValue(float val, int type, int dimension)
```

Discussion

The value *val* is either an absolute value in points or a percentage of the enclosing block, as specified by *type*. The *dimension* argument specifies which dimension's value to set.

Availability

Available in Mac OS X v10.4 and later.

See Also

`valueForDimension` (page 1551)
`valueTypeForDimension` (page 1551)

setVerticalAlignment

Sets the vertical alignment of the text block to *alignment*.

```
public void setVerticalAlignment(int alignment)
```

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

See Also
`verticalAlignment` (page 1552)

setWidthForLayer

Sets the width of one or more edges of a specified layer of the text block.

```
public void setWidthForLayer(float val, int type, int layer)
```

Discussion
The width is specified by *val*, which is interpreted as points or a percentage according to *type*, and the layer—border, padding, or margin—is specified by *layer*.

Sets the width of one or more edges of a specified layer of the text block.

```
public void setWidthForLayer(float val, int type, int layer, int edge)
```

Discussion
The edge is specified by *edge*. If you do not specify an *edge* parameter, the width is applied to all edges.

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

See Also
`widthForLayer` (page 1552)

valueForDimension

Returns the value of the text block dimension specified by *dimension*.

```
public float valueForDimension(int dimension)
```

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

See Also
`setValue` (page 1550)

valueTypeForDimension

Returns the value type of the text block dimension specified by *dimension*.

```
public int valueTypeForDimension(int dimension)
```

Discussion
The value type determines whether the value is interpreted as an absolute value in points or a percentage of the enclosing block.

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

See Also
`setValue` (page 1550)

verticalAlignment

Returns the vertical alignment of the text block.

```
public int verticalAlignment()
```

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

See Also
`setVerticalAlignment` (page 1550)

widthForLayer

Returns the width of a *layer* for a given *edge* of the text block in points.

```
public float widthForLayer(int layer, int edge)
```

Discussion
The text block layer can be its border, padding, or margin.

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

See Also
`setWidthForLayer` (page 1551)

widthValueTypeForLayer

Returns the value type of a *layer* for a given *edge* of the text block.

```
public int widthValueTypeForLayer(int layer, int edge)
```

Discussion
The text block layer can be its border, padding, or margin.

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

See Also
`setWidthForLayer` (page 1551)

Constants

The following constants specify values used with method parameters used for specifying width types.

Constant	Description
`AbsoluteValueType`	Absolute value in points
`PercentageValueType`	Percentage value (out of 100)

The following constants specify values used with method parameters used for specifying dimensions.

Constant	Description
`Width`	Width of the text block
`MinimumWidth`	Minimum width of the text block
`MaximumWidth`	Maximum width of the text block
`Height`	Height of the text block
`MinimumHeight`	Minimum height of the text block
`MaximumHeight`	Maximum height of the text block

The following constants specify values used with method parameters used for specifying layer information.

Constant	Description
`Padding`	Padding of the text block: space surrounding the content area extending to the border
`Border`	Border of the text block: space between padding and margin, typically colored to present a visible boundary
`Margin`	Margin of the text block: space surrounding the border

The following constants specify values used with method parameters used for specifying alignment.

Constant	Description
`TopAlignment`	Aligns adjacent blocks at their top
`MiddleAlignment`	Aligns adjacent blocks at their middle
`BottomAlignment`	Aligns adjacent blocks at their bottom
`BaselineAlignment`	Aligns adjacent blocks at the baseline of the first line of text in the block

NSTextContainer

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.application
Companion guide	Text System Overview

Overview

An NSTextContainer defines a region where text is laid out. An NSLayoutManager uses NSTextContainers to determine where to break lines, lay out portions of text, and so on. NSTextContainer defines rectangular regions, but you can create subclasses that define regions of other shapes, such as circular regions, regions with holes in them, or regions that flow alongside graphics.

Tasks

Constructors

`NSTextContainer` (page 1557)
: Creates an NSTextContainer with a zero-sized bounding rectangle.

Managing Text Components

`setLayoutManager` (page 1560)
: Sets the receiver's NSLayoutManager to *`aLayoutManager`*.

`layoutManager` (page 1558)
: Returns the receiver's NSLayoutManager.

`replaceLayoutManager` (page 1559)
: Replaces the NSLayoutManager for the group of text system objects containing the receiver with *`aLayoutManager`*.

`setTextView` (page 1561)
: Sets the receiver's NSTextView to *`aTextView`* and sends `setTextContainer` (page 1653) to *`aTextView`* to complete the association of the text container and text view.

`textView` (page 1561)
: Returns the receiver's NSTextView, or `null` if it has none.

Controlling Size

`setContainerSize` (page 1559)
Sets the size of the receiver's bounding rectangle to *aSize* and sends `textContainerChangedGeometry` (page 852) to the NSLayoutManager.

`containerSize` (page 1557)
Returns the size of the receiver's bounding rectangle, regardless of the size of its region.

`setWidthTracksTextView` (page 1561)
Controls whether the receiver adjusts the width of its bounding rectangle when its NSTextView is resized.

`widthTracksTextView` (page 1561)
Returns `true` if the receiver adjusts the width of its bounding rectangle when its NSTextView is resized, `false` otherwise.

`setHeightTracksTextView` (page 1560)
Controls whether the receiver adjusts the height of its bounding rectangle when its NSTextView is resized.

`heightTracksTextView` (page 1558)
Returns `true` if the receiver adjusts the height of its bounding rectangle when its NSTextView is resized, `false` otherwise.

Setting Line Fragment Padding

`setLineFragmentPadding` (page 1560)
Sets the amount (in points) by which text is inset within line fragment rectangles to *aFloat*.

`lineFragmentPadding` (page 1558)
Returns the amount (in points) by which text is inset within line fragment rectangles.

Calculating Text Layout

`lineFragmentRectForProposedRect` (page 1559)
Overridden by subclasses to calculate and return the longest rectangle available for *proposedRect* for displaying text, or `NSRect.ZeroRect` if there is none according to the receiver's region definition.

`isSimpleRectangularTextContainer` (page 1558)
Overridden by subclasses to return `true` if the receiver's region is a rectangle with no holes or gaps and whose edges are parallel to the NSTextView's coordinate system axes; returns `false` otherwise.

Mouse Hit Testing

`containsPoint` (page 1557)
Overridden by subclasses to return `true` if *aPoint* lies within the receiver's region or on the region's edge—not simply within its bounding rectangle—`false` otherwise.

Constructors

NSTextContainer

Creates an NSTextContainer with a zero-sized bounding rectangle.

```
public NSTextContainer()
```

Creates an NSTextContainer, with *aSize* as the size of its bounding rectangle.

```
public NSTextContainer(NSSize aSize)
```

Discussion
For both constructors, the new NSTextContainer must be added to an NSLayoutManager before it can be used; it must also have an NSTextView set for text to be displayed.

See Also
`addTextContainer` (page 827) (NSLayoutManager)
`setTextView` (page 1561)

Instance Methods

containerSize

Returns the size of the receiver's bounding rectangle, regardless of the size of its region.

```
public NSSize containerSize()
```

See Also
`textContainerInset` (page 1659) (NSTextView)
`setContainerSize` (page 1559)

containsPoint

Overridden by subclasses to return `true` if *aPoint* lies within the receiver's region or on the region's edge—not simply within its bounding rectangle—`false` otherwise.

```
public boolean containsPoint(NSPoint aPoint)
```

Discussion
For example, if the receiver defines a donut shape and *aPoint* lies in the hole, this method returns `false`. This method can be used for hit testing of mouse events.

NSTextContainer's implementation merely checks that *aPoint* lies within its bounding rectangle.

heightTracksTextView

Returns `true` if the receiver adjusts the height of its bounding rectangle when its NSTextView is resized, `false` otherwise.

```
public boolean heightTracksTextView()
```

Discussion

The height is adjusted to the height of the NSTextView minus twice the inset height (as given by NSTextView's `textContainerInset` (page 1659) method).

See *Text System Storage Layer Overview* for more information on size tracking.

See Also

`widthTracksTextView` (page 1561)

`setHeightTracksTextView` (page 1560)

isSimpleRectangularTextContainer

Overridden by subclasses to return `true` if the receiver's region is a rectangle with no holes or gaps and whose edges are parallel to the NSTextView's coordinate system axes; returns `false` otherwise.

```
public boolean isSimpleRectangularTextContainer()
```

Discussion

An NSTextContainer whose shape changes can return `true` if its region is currently a simple rectangle, but when its shape does change it must send `textContainerChangedGeometry` (page 852) to its NSLayoutManager so the layout can be recalculated.

NSTextContainer's implementation of this method returns `true`.

layoutManager

Returns the receiver's NSLayoutManager.

```
public NSLayoutManager layoutManager()
```

See Also

`setLayoutManager` (page 1560)

`replaceLayoutManager` (page 1559)

lineFragmentPadding

Returns the amount (in points) by which text is inset within line fragment rectangles.

```
public float lineFragmentPadding()
```

See Also

`lineFragmentRectForProposedRect` (page 1559)

`setLineFragmentPadding` (page 1560)

lineFragmentRectForProposedRect

Overridden by subclasses to calculate and return the longest rectangle available for `proposedRect` for displaying text, or `NSRect.ZeroRect` if there is none according to the receiver's region definition.

```
public NSRect lineFragmentRectForProposedRect(NSRect proposedRect, int
    sweepDirection, int movementDirection, NSMutableRect remainingRect)
```

Discussion

There is no guarantee as to the width of the proposed rectangle or to its location. For example, the proposed rectangle is likely to be much wider than the width of the receiver. The receiver should examine *proposedRect* to see that it intersects its bounding rectangle and should return a modified rectangle based on *sweepDirection* and *movementDirection*, whose possible values are listed in the class description. If *sweepDirection* is `LineSweepRight`, for example, the receiver uses this information to trim the right end of *proposedRect* as needed rather than the left end.

If *proposedRect* doesn't completely overlap the region along the axis of *movementDirection* and *movementDirection* isn't `LineDoesntMove`, this method can either shift the rectangle in that direction as much as needed so that it does completely overlap, or return `NSRect.ZeroRect` to indicate that the proposed rectangle simply doesn't fit.

Upon returning, *remainingRect* contains the unused, possibly shifted, portion of *proposedRect* that's available for further text, or `NSRect.ZeroRect` if there is no remainder.

See the class description for more information on overriding this method.

replaceLayoutManager

Replaces the NSLayoutManager for the group of text system objects containing the receiver with *aLayoutManager*.

```
public void replaceLayoutManager(NSLayoutManager aLayoutManager)
```

Discussion

All NSTextContainers and NSTextViews sharing the original NSLayoutManager then share the new one. This method makes all the adjustments necessary to keep these relationships intact, unlike `setLayoutManager` (page 1560).

See Also

`layoutManager` (page 1558)

setContainerSize

Sets the size of the receiver's bounding rectangle to *aSize* and sends `textContainerChangedGeometry` (page 852) to the NSLayoutManager.

```
public void setContainerSize(NSSize aSize)
```

See Also

`setTextContainerInset` (page 1654) (NSTextView)

`containerSize` (page 1557)

setHeightTracksTextView

Controls whether the receiver adjusts the height of its bounding rectangle when its NSTextView is resized.

```
public void setHeightTracksTextView(boolean flag)
```

Discussion
If *flag* is true, the receiver follows changes to the height of its text view; if *flag* is false, it doesn't.

See *Text System Storage Layer Overview* for more information on size tracking.

See Also
setContainerSize (page 1559)
setWidthTracksTextView (page 1561)
heightTracksTextView (page 1558)

setLayoutManager

Sets the receiver's NSLayoutManager to *aLayoutManager*.

```
public void setLayoutManager(NSLayoutManager aLayoutManager)
```

Discussion
This method is invoked automatically when you add an NSTextContainer to an NSLayoutManager; you should never need to invoke it directly, but might want to override it. If you want to replace the NSLayoutManager for an established group of text system objects, use replaceLayoutManager (page 1559).

See Also
addTextContainer (page 827) (NSLayoutManager)
layoutManager (page 1558)

setLineFragmentPadding

Sets the amount (in points) by which text is inset within line fragment rectangles to *aFloat*.

```
public void setLineFragmentPadding(float aFloat)
```

Discussion
Also sends textContainerChangedGeometry (page 852) to the receiver's NSLayoutManager to inform it of the change.

Line fragment padding is not designed to express text margins. Instead, use the NSTextView method setTextContainerInset (page 1654), paragraph margin attributes, or the position of the text view within a superview.

See Also
lineFragmentRectForProposedRect (page 1559)
lineFragmentPadding (page 1558)

setTextView

Sets the receiver's NSTextView to *aTextView* and sends setTextContainer (page 1653) to *aTextView* to complete the association of the text container and text view.

```
public void setTextView(NSTextView aTextView)
```

Discussion

Because you usually specify an NSTextContainer when you create an NSTextView, you should rarely need to invoke this method. An NSTextContainer doesn't need an NSTextView to calculate line fragment rectangles, but must have one to display text.

You can use this method to disconnect an NSTextView from a group of text system objects by sending this message to its text container and passing null as *aTextView*.

See Also

replaceTextContainer (page 1640) (NSTextView)

setWidthTracksTextView

Controls whether the receiver adjusts the width of its bounding rectangle when its NSTextView is resized.

```
public void setWidthTracksTextView(boolean flag)
```

Discussion

If *flag* is true, the receiver follows changes to the width of its text view; if *flag* is false, it doesn't.

See *Text System Storage Layer Overview* for more information on size tracking.

See Also

setContainerSize (page 1559)

setHeightTracksTextView (page 1560)

widthTracksTextView (page 1561)

textView

Returns the receiver's NSTextView, or null if it has none.

```
public NSTextView textView()
```

See Also

setTextView (page 1561)

widthTracksTextView

Returns true if the receiver adjusts the width of its bounding rectangle when its NSTextView is resized, false otherwise.

```
public boolean widthTracksTextView()
```

Discussion
The width is adjusted to the width of the NSTextView minus twice the inset width (as given by NSTextView's `textContainerInset` (page 1659) method).

See *Text System Storage Layer Overview* for more information on size tracking.

See Also
`heightTracksTextView` (page 1558)
`setWidthTracksTextView` (page 1561)

Constants

These constants describe the progression of text on a page. The typesetter decides which way text is supposed to flow and passes these values as arguments to the text container, which uses them to calculate the next line rectangle.

The only values currently used by the supplied typesetters are `LineSweepRight` and `LineMovesDown`. An NSTextContainer subclass should be prepared to deal with any value, and an NSTypesetter subclass should be able to use any of them.

Line sweep is the direction text progresses within a line. See *Text System Storage Layer Overview*.

Constant	Description
`LineSweepLeft`	Characters move from right to left.
`LineSweepRight`	Characters move from left to right.
`LineSweepDown`	Characters move from top to bottom.
`LineSweepUp`	Characters move from bottom to top.

Line movement is the direction in which lines move. See *Text System Storage Layer Overview*.

Constant	Description
`LineMovesLeft`	Lines move from right to left.
`LineMovesRight`	Lines move from left to right.
`LineMovesDown`	Lines move from top to bottom.
`LineMovesUp`	Lines move from bottom to top.
`LineDoesntMove`	Line has no movement.

NSTextField

Inherits from	NSControl : NSView : NSResponder : NSObject
Implements	NSCoding (NSResponder)
Package:	com.apple.cocoa.application
Companion guide	Text Fields

Overview

An NSTextField is a kind of NSControl that displays text that the user can select or edit and that sends its action message to its target when the user presses the Return key while editing.

NSTextField uses NSTextFieldCell (page 1575) to implement its user interface.

Tasks

Constructors

`NSTextField` (page 1566)
Creates an NSTextField with a zero-sized frame rectangle.

Controlling Editability and Selectability

`setEditable` (page 1571)
Controls whether the user can edit the receiver's text.

`isEditable` (page 1568)
Returns `true` if the user is allowed to select and edit the receiver's text, `false` if the user isn't allowed to edit it (though the user may be able to select it).

`setSelectable` (page 1571)

`isSelectable` (page 1568)
Returns `true` if the user is allowed to select the receiver's text, `false` if the user isn't allowed to select it.

Controlling Rich Text Behavior

`setAllowsEditingTextAttributes` (page 1569)
    Controls whether the receiver allows the user to change font attributes of the receiver's text.

`allowsEditingTextAttributes` (page 1566)
    Returns `true` if the receiver allows the user to change font attributes of the receiver's text, `false` if the user isn't permitted to do so.

`setImportsGraphics` (page 1571)
    Controls whether the receiver allows the user to drag image files into it.

`importsGraphics` (page 1567)
    Returns `true` if the receiver allows the user to drag image files into it, `false` if it doesn't accept dragged images.

Setting the Text Color

`setTextColor` (page 1572)
    Sets the color used to draw the receiver's text to *aColor*.

`textColor` (page 1572)
    Returns the color used to draw the receiver's text.

Controlling the Background

`setBackgroundColor` (page 1569)
    Sets the color of the background that the receiver draws behind the text to *aColor*.

`backgroundColor` (page 1566)
    Returns the color of the background that the receiver draws behind the text.

`setDrawsBackground` (page 1570)
    Controls whether the receiver draws its background color behind its text.

`drawsBackground` (page 1567)
    Returns `true` if the receiver's cell draws its background color behind its text, `false` if it draws no background.

Setting a Border

`setBezeled` (page 1569)
    Controls whether the receiver draws a bezeled border around its contents.

`isBezeled` (page 1568)
    Returns `true` if the receiver draws a bezeled frame around its contents, `false` if it doesn't.

`setBezelStyle` (page 1570)
    Sets the receiver's bezel style to be *style*.

`bezelStyle` (page 1567)
    Returns the receiver's bezel style.

`setBordered` (page 1570)
: Controls whether the receiver draws a solid black border around its contents.

`isBordered` (page 1568)
: Returns `true` if the receiver draws a solid black border around its contents, `false` if it doesn't.

Selecting the Text

`selectText` (page 1569)
: This action method ends editing and selects the entire contents of the receiver if it's selectable.

Working with the Responder Chain

`acceptsFirstResponder` (page 1566)
: Returns `true` if the receiver is editable, `false` otherwise.

Using Keyboard Interface Control

`setTitleWithMnemonic` (page 1572)
: Sets the receiver's string value to *`aString`*, using the first character preceded by an ampersand ('&') as the mnemonic and stripping out that first ampersand character.

Setting the Delegate

`setDelegate` (page 1570)
: Sets the receiver's delegate to *`anObject`*.

`delegate` (page 1567)
: Returns the receiver's delegate.

Text Delegate Methods

`textShouldBeginEditing` (page 1573)

`textDidBeginEditing` (page 1572)
: Posts a `ControlTextDidBeginEditingNotification` (page 466) to the default notification center.

`textDidChange` (page 1573)
: Forwards this message to the receiver's cell if it responds and posts a `ControlTextDidChangeNotification` (page 467) to the default notification center.

`textShouldEndEditing` (page 1574)
: Performs validation on the receiver's new value using NSCell's `isEntryAcceptable` (page 314), sending the receiver's error action to its target if validation fails.

`textDidEndEditing` (page 1573)
: Handles an end of editing NSNotification *`aNotification`*.

Constructors

NSTextField

Creates an NSTextField with a zero-sized frame rectangle.

```
public NSTextField()
```

Creates an NSTextField with *frameRect* as its frame rectangle.

```
public NSTextField(NSRect frameRect)
```

Instance Methods

acceptsFirstResponder

Returns `true` if the receiver is editable, `false` otherwise.

```
public boolean acceptsFirstResponder()
```

allowsEditingTextAttributes

Returns `true` if the receiver allows the user to change font attributes of the receiver's text, `false` if the user isn't permitted to do so.

```
public boolean allowsEditingTextAttributes()
```

Discussion
You can change text attributes programmatically regardless of this setting.

See Also
`importsGraphics` (page 1567)
`setAllowsEditingTextAttributes` (page 1569)

backgroundColor

Returns the color of the background that the receiver draws behind the text.

```
public NSColor backgroundColor()
```

See Also
`drawsBackground` (page 1567)
`setBackgroundColor` (page 1569)

bezelStyle

Returns the receiver's bezel style.

```
public int bezelStyle()
```

Discussion
Possible return values are described in NSTextFieldCell's "Constants" (page 1580).

Availability
Available in Mac OS X v10.2 and later.

See Also
`setBezelStyle` (page 1570)

delegate

Returns the receiver's delegate.

```
public Object delegate()
```

See Also
`textShouldBeginEditing` (page 1573)
`textShouldEndEditing` (page 1574)
`textDidBeginEditing` (page 1572)
`textDidEndEditing` (page 1573)
`textDidChange` (page 1573)
`setDelegate` (page 1570)

drawsBackground

Returns `true` if the receiver's cell draws its background color behind its text, `false` if it draws no background.

```
public boolean drawsBackground()
```

See Also
`backgroundColor` (page 1566)
`drawsBackground` (page 1577) (NSTextFieldCell)
`setDrawsBackground` (page 1570)

importsGraphics

Returns `true` if the receiver allows the user to drag image files into it, `false` if it doesn't accept dragged images.

```
public boolean importsGraphics()
```

Discussion
You can add images programmatically regardless of this setting.

See Also

`allowsEditingTextAttributes` (page 1566)
`importsGraphics` (page 1629) (NSTextView)
`setImportsGraphics` (page 1571)

isBezeled

Returns `true` if the receiver draws a bezeled frame around its contents, `false` if it doesn't.

```
public boolean isBezeled()
```

See Also

`isBordered` (page 1568)
`setBezeled` (page 1569)

isBordered

Returns `true` if the receiver draws a solid black border around its contents, `false` if it doesn't.

```
public boolean isBordered()
```

See Also

`isBezeled` (page 1568)
`setBordered` (page 1570)

isEditable

Returns `true` if the user is allowed to select and edit the receiver's text, `false` if the user isn't allowed to edit it (though the user may be able to select it).

```
public boolean isEditable()
```

See Also

`isSelectable` (page 1568)
`setEditable` (page 1571)

isSelectable

Returns `true` if the user is allowed to select the receiver's text, `false` if the user isn't allowed to select it.

```
public boolean isSelectable()
```

Discussion

Selectable text isn't necessarily editable; use `isEditable` (page 1568) to check for editability.

See Also

`setSelectable` (page 1571)

selectText

This action method ends editing and selects the entire contents of the receiver if it's selectable.

```
public void selectText(Object sender)
```

Discussion

However, if the receiver isn't in some window's view hierarchy, this method has no effect.

See Also

`isSelectable` (page 1568)

setAllowsEditingTextAttributes

Controls whether the receiver allows the user to change font attributes of the receiver's text.

```
public void setAllowsEditingTextAttributes(boolean flag)
```

Discussion

If *`flag`* is `true`, the user is permitted to make such changes; if *`flag`* is `false`, the user isn't so permitted. You can change text attributes programmatically regardless of this setting.

See Also

`setImportsGraphics` (page 1571)

`allowsEditingTextAttributes` (page 1566)

setBackgroundColor

Sets the color of the background that the receiver draws behind the text to *`aColor`*.

```
public void setBackgroundColor(NSColor aColor)
```

See Also

`setDrawsBackground` (page 1570)

`backgroundColor` (page 1566)

setBezeled

Controls whether the receiver draws a bezeled border around its contents.

```
public void setBezeled(boolean flag)
```

Discussion

If *`flag`* is `false`, it draws no border; if *`flag`* is `true`, it draws a bezeled border and invokes `setDrawsBackground` (page 1570) with an argument of `false`.

See Also

`isBezeled` (page 1568)

`setBordered` (page 1570)

setBezelStyle

Sets the receiver's bezel style to be *style*.

```
public void setBezelStyle(int style)
```

Discussion
Possible values for *style* are described in NSTextFieldCell's "Constants" (page 1580). You must have already sent the receiver `setBezeled` (page 1569) with an argument of `true`.

Availability
Available in Mac OS X v10.2 and later.

See Also
`bezelStyle` (page 1567)

setBordered

Controls whether the receiver draws a solid black border around its contents.

```
public void setBordered(boolean flag)
```

Discussion
If *flag* is `true`, then it draws a border; if *flag* is `false`, it draws no border.

See Also
`isBordered` (page 1568)
`setBezeled` (page 1569)

setDelegate

Sets the receiver's delegate to *anObject*.

```
public void setDelegate(Object anObject)
```

See Also
`textShouldBeginEditing` (page 1573)
`textShouldEndEditing` (page 1574)
`textDidBeginEditing` (page 1572)
`textDidEndEditing` (page 1573)
`textDidChange` (page 1573)
`delegate` (page 1567)

setDrawsBackground

Controls whether the receiver draws its background color behind its text.

```
public void setDrawsBackground(boolean flag)
```

Discussion
If *flag* is `true`, then it does; if *flag* is `false`, then it draws nothing behind its text.

See Also

`setBackgroundColor` (page 1569)

`setDrawsBackground` (page 1579) (NSTextFieldCell)

`drawsBackground` (page 1567)

setEditable

Controls whether the user can edit the receiver's text.

```
public void setEditable(boolean flag)
```

Discussion

If *flag* is `true`, then the user is allowed to both select and edit text. If *flag* is `false`, then the user isn't permitted to edit text, and the receiver's selectability is restored to its previous value. For example, if an NSTextField is selectable but not editable, then made editable for a time, then made not editable, it remains selectable. To guarantee that text is neither editable nor selectable, simply use `setSelectable` (page 1571) to turn off selectability.

See Also

`isEditable` (page 1568)

setImportsGraphics

Controls whether the receiver allows the user to drag image files into it.

```
public void setImportsGraphics(boolean flag)
```

Discussion

If *flag* is `true`, the receiver accepts dragged images; if *flag* is `false`, it doesn't. You can add images programmatically regardless of this setting.

See Also

`setAllowsEditingTextAttributes` (page 1569)

`setImportsGraphics` (page 1649) (NSTextView)

`importsGraphics` (page 1567)

setSelectable

```
public void setSelectable(boolean flag)
```

Discussion

If *flag* is `true`, the receiver is made selectable but not editable (use `setEditable` (page 1571) to make text both selectable and editable). If `false`, then the text is made neither editable nor selectable.

See Also

`setEditable` (page 1571)

setTextColor

Sets the color used to draw the receiver's text to *aColor*.

```
public void setTextColor(NSColor aColor)
```

See Also

setBackgroundColor (page 1569)
setTextColor (page 1580) (NSTextFieldCell)
textColor (page 1572)

setTitleWithMnemonic

Sets the receiver's string value to *aString*, using the first character preceded by an ampersand ('&') as the mnemonic and stripping out that first ampersand character.

```
public void setTitleWithMnemonic(String aString)
```

Discussion

Mnemonics are not supported in Mac OS X.

textColor

Returns the color used to draw the receiver's text.

```
public NSColor textColor()
```

See Also

backgroundColor (page 1566)
textColor (page 1580) (NSTextFieldCell)
setTextColor (page 1572)

textDidBeginEditing

Posts a ControlTextDidBeginEditingNotification (page 466) to the default notification center.

```
public void textDidBeginEditing(NSNotification aNotification)
```

Discussion

This action causes the receiver's delegate to receive a controlTextDidBeginEditing (page 464) message. See the NSControl (page 441) class specification for more information on the text delegate method.

See Also

textDidBeginEditing (page 1572)
textDidChange (page 1573)
textShouldEndEditing (page 1574)
textDidEndEditing (page 1573)

textDidChange

Forwards this message to the receiver's cell if it responds and posts a `ControlTextDidChangeNotification` (page 467) to the default notification center.

```
public void textDidChange(NSNotification aNotification)
```

Discussion

This method causes the receiver's delegate to receive a `controlTextDidChange` (page 465) message. See the NSControl (page 441) class specification for more information on the text delegate method.

See Also

`textShouldBeginEditing` (page 1573)

`textDidBeginEditing` (page 1572)

`textShouldEndEditing` (page 1574)

`textDidEndEditing` (page 1573)

textDidEndEditing

Handles an end of editing NSNotification *aNotification*.

```
public void textDidEndEditing(NSNotification aNotification)
```

Discussion

After validating the new value, posts a `ControlTextDidEndEditingNotification` (page 467) to the default notification center. This posting causes the receiver's delegate to receive a `controlTextDidEndEditing` (page 465) message. After this message, sends `endEditing` (page 310) to the receiver' cell and handles the key that caused editing to end:

- If the user ended editing by pressing Return, this method tries to send the receiver's action to its target; if unsuccessful, it sends `performKeyEquivalent` (page 1763) to its NSView (for example, to handle the default button on a panel); if that also fails, the receiver simply selects its text.
- If the user ended editing by pressing Tab or Shift-Tab, the receiver tries to have its NSWindow select its next or previous key view, using the NSWindow method `selectKeyViewFollowingView` (page 1852) or `selectKeyViewPrecedingView` (page 1852). If unsuccessful in doing this, the receiver simply selects its text.

See the NSControl (page 441) class specification for more information on the text delegate method.

See Also

`textShouldBeginEditing` (page 1573)

`textDidBeginEditing` (page 1572)

`textDidChange` (page 1573)

`textShouldEndEditing` (page 1574)

textShouldBeginEditing

```
public boolean textShouldBeginEditing(NSText textObject)
```

Discussion

If the receiver isn't editable, returns `false` immediately. If it is editable and its delegate responds to `controlTextShouldBeginEditing` (page 465), invokes that method and returns the result. Otherwise simply returns `true` to allow editing to occur. The *`textObject`* argument is the NSText object that invoked this method. See the NSControl (page 441) class specification for more information on the text delegate method.

See Also

`textDidBeginEditing` (page 1572)
`textDidChange` (page 1573)
`textShouldEndEditing` (page 1574)
`textDidEndEditing` (page 1573)

textShouldEndEditing

Performs validation on the receiver's new value using NSCell's `isEntryAcceptable` (page 314), sending the receiver's error action to its target if validation fails.

```
public boolean textShouldEndEditing(NSText textObject)
```

Discussion

If the new value is valid and the delegate responds to `controlTextShouldEndEditing` (page 465), invokes that method and returns the result, in addition beeping if the delegate returns `false`. The *`textObject`* argument is the NSText object that invoked this method. See the NSControl (page 441) class specification for more information on the text delegate method.

See Also

`textShouldBeginEditing` (page 1573)
`textDidBeginEditing` (page 1572)
`textDidChange` (page 1573)
`textDidEndEditing` (page 1573)

NSTextFieldCell

Inherits from	NSActionCell : NSCell : NSObject
Implements	NSCoding (NSCell)
Package:	com.apple.cocoa.application
Companion guide	Text Fields

Overview

NSTextFieldCell adds to NSCell's text display capabilities by allowing you to set the color of both the text and its background. You can also specify whether the cell draws its background at all. All of the methods declared by this class are also declared by NSTextField (page 1563), which uses NSTextFieldCells to draw and edit text.

Placeholder strings, set using `setPlaceholderString` (page 1579) or `setPlaceholderAttributedString` (page 1579), now appear in the text field cell if the actual string is `null` or " ". They are drawn in grey on the cell and are not archived in the "pre-10.2" nib format.

Tasks

Constructors

`NSTextFieldCell` (page 1576)
Creates an empty NSTextFieldCell.

Setting the Text Color

`setTextColor` (page 1580)
Sets the color used to draw the receiver's text to *`aColor`*.

`textColor` (page 1580)
Returns the color used to draw the receiver's text.

Setting the Bezel Style

`setBezelStyle` (page 1578)
: Sets the receiver's bezel style to be *`style`*.

`bezelStyle` (page 1577)
: Returns the receiver's bezel style.

Controlling the Background

`setBackgroundColor` (page 1578)
: Sets the color of the background that the receiver draws behind the text to *`aColor`*.

`backgroundColor` (page 1577)
: Returns the color of the background the receiver draws behind the text.

`setDrawsBackground` (page 1579)
: Controls whether the receiver draws its background color behind its text.

`drawsBackground` (page 1577)
: Returns `true` if the receiver's cell draws its background color behind its text, `false` if it draws no background.

Changing the Field Editor

`setUpFieldEditorAttributes` (page 1580)

Managing Placeholder Strings

`setPlaceholderString` (page 1579)
: Sets the placeholder of the cell as a plain text string.

`placeholderString` (page 1578)
: Returns the cell's plain text placeholder string.

`setPlaceholderAttributedString` (page 1579)
: Sets the placeholder of the cell as an attributed string.

`placeholderAttributedString` (page 1578)
: Returns the cell's attributed placeholder string.

Constructors

NSTextFieldCell

Creates an empty NSTextFieldCell.

```
public NSTextFieldCell()
```

Creates an NSTextFieldCell initialized with *aString* and set to have the cell's default menu.

```
public NSTextFieldCell(String aString)
```

Discussion
If no field editor has been created, one is created.

Creates an NSTextFieldCell initialized with *anImage* and set to have the cell's default menu.

```
public NSTextFieldCell(NSImage anImage)
```

Discussion
If *anImage* is `null`, no image is set.

Instance Methods

backgroundColor

Returns the color of the background the receiver draws behind the text.

```
public NSColor backgroundColor()
```

See Also
`drawsBackground` (page 1577)
`backgroundColor` (page 1566) (NSTextField)
`setBackgroundColor` (page 1578)

bezelStyle

Returns the receiver's bezel style.

```
public int bezelStyle()
```

Discussion
Possible return values are described in "Constants" (page 1580).

Availability
Available in Mac OS X v10.2 and later.

See Also
`setBezelStyle` (page 1578)

drawsBackground

Returns `true` if the receiver's cell draws its background color behind its text, `false` if it draws no background.

```
public boolean drawsBackground()
```

See Also
`backgroundColor` (page 1577)

drawsBackground (page 1567) (NSTextField)
setDrawsBackground (page 1579)

placeholderAttributedString

Returns the cell's attributed placeholder string.

```
public NSAttributedString placeholderAttributedString()
```

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

See Also
setPlaceholderAttributedString (page 1579)
placeholderString (page 1578)

placeholderString

Returns the cell's plain text placeholder string.

```
public String placeholderString()
```

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

See Also
setPlaceholderString (page 1579)
placeholderAttributedString (page 1578)

setBackgroundColor

Sets the color of the background that the receiver draws behind the text to *aColor*.

```
public void setBackgroundColor(NSColor aColor)
```

See Also
setDrawsBackground (page 1579)
setBackgroundColor (page 1569) (NSTextField)
backgroundColor (page 1577)

setBezelStyle

Sets the receiver's bezel style to be *style*.

```
public void setBezelStyle(int style)
```

Discussion
Possible values for *style* are described in "Constants" (page 1580). You must have already sent setBezeled (page 1569) with an argument of true.

Availability
Available in Mac OS X v10.2 and later.

See Also
`bezelStyle` (page 1577)

setDrawsBackground

Controls whether the receiver draws its background color behind its text.

```
public void setDrawsBackground(boolean flag)
```

Discussion
If *flag* is `true`, then it does; if *flag* is `false`, then it draws nothing behind its text.

See Also
`setBackgroundColor` (page 1578)
`setDrawsBackground` (page 1570) (NSTextField)
`drawsBackground` (page 1577)

setPlaceholderAttributedString

Sets the placeholder of the cell as an attributed string.

```
public void setPlaceholderAttributedString(NSAttributedString string)
```

Discussion
Note that invoking this successfully will clear out the plain text string set by `setPlaceholderString` (page 1579).

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

See Also
`placeholderAttributedString` (page 1578)
`setPlaceholderString` (page 1579)

setPlaceholderString

Sets the placeholder of the cell as a plain text string.

```
public void setPlaceholderString(String string)
```

Discussion
Note that invoking this successfully will clear out the attributed string set by `setPlaceholderAttributedString` (page 1579).

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

See Also

`placeholderString` (page 1578)

`setPlaceholderAttributedString` (page 1579)

setTextColor

Sets the color used to draw the receiver's text to *aColor*.

```
public void setTextColor(NSColor aColor)
```

See Also

`setBackgroundColor` (page 1578)

`setTextColor` (page 1572) (NSTextField)

`textColor` (page 1580)

setUpFieldEditorAttributes

```
public NSText setUpFieldEditorAttributes(NSText textObj)
```

Discussion

You never invoke this method directly; by overriding it, however, you can customize or replace the field editor. When you override this method, you should generally invoke the implementation of `super` and return the *textObj* argument. For information on field editors, see "Using the Window's Field Editor".

textColor

Returns the color used to draw the receiver's text.

```
public NSColor textColor()
```

See Also

`backgroundColor` (page 1577)

`textColor` (page 1572) (NSTextField)

`setTextColor` (page 1580)

Constants

The following constants are specify the bezel style of the text field, and are set using `bezelStyle` (page 1577) and `setBezelStyle` (page 1578).

Constant	Description
`TextFieldSquareBezel`	Corners are square.
`TextFieldRoundedBezel`	Corners are rounded.

NSTextList

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.application
Availability	Availabile in Mac OS X v10.4 and later.

Overview

NSTextList represents a section of text that forms a single list. The visible elements of the list, including list markers, appear in the text as they do for lists created by hand. The list object, however, allows the list to be recognized as such by the text system. This enables automatic creation of markers and spacing. Text lists are used in HTML import and export.

Text lists appear as attributes on paragraphs, as part of the paragraph style. An NSParagraphStyle may have an array of text lists, representing the nested lists containing the paragraph, in order from outermost to innermost. For example, if list1 contains four paragraphs, the middle two of which are also in the inner list2, then the text lists array for the first and fourth paragraphs is (list1), while the text lists array for the second and third paragraphs is (list1, list2).

The methods implementing this are `textLists` (page 1065) on NSParagraphStyle, and `setTextLists` (page 1001) on NSMutableParagraphStyle.

Tasks

Constructors

`NSTextList` (page 1582)

Working with Markers

`markerFormat` (page 1582)
    Returns the marker format string used by the receiver.

`markerForItemNumber` (page 1582)
    Returns the computed value for a specific ordinal position in the list specified by *`itemNum`*.

Getting List Options

`listOptions` (page 1582)
Returns the list options mask value of the receiver.

Constructors

NSTextList

```
public NSTextList()
```

Discussion
Creates an NSTextList object.

```
public NSTextList(String format, int mask)
```

Discussion
Creates an NSTextList object with a marker format specified by *format* and the marker options specified in *mask*. Values for *mask* are listed in "Constants" (page 1583).

Instance Methods

listOptions

Returns the list options mask value of the receiver.

```
public int listOptions()
```

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

markerForItemNumber

Returns the computed value for a specific ordinal position in the list specified by *itemNum.*

```
public String markerForItemNumber(int itemNum)
```

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

markerFormat

Returns the marker format string used by the receiver.

```
public String markerFormat()
```

Availability

Available in Mac OS X v10.4 and later.

See Also

`NSTextList` (page 1582)

Constants

The following constant specifies an option mask used with `NSTextList` (page 1582).

Constant	Description
`PrependEnclosingMarker`	Specifies that a nested list should include the marker for its enclosing superlist before its own marker.

NSTextStorage

Inherits from	NSMutableAttributedString : NSAttributedString : NSObject
Package:	com.apple.cocoa.application
Companion guides	Text System Overview Text System Storage Layer Overview

Overview

NSTextStorage is a semiconcrete subclass of NSMutableAttributedString that manages a set of client NSLayoutManagers, notifying them of any changes to its characters or attributes so that they can relay and redisplay the text as needed. NSTextStorage defines the fundamental storage mechanism of the Application Kit's extended text-handling system.

Tasks

Constructors

`NSTextStorage` (page 1587)
Creates an empty NSTextStorage.

Managing NSLayoutManagers

`addLayoutManager` (page 1588)
Adds *aLayoutManager* to the receiver's set of NSLayoutManagers.

`removeLayoutManager` (page 1591)
Removes *aLayoutManager* from the receiver's set of NSLayoutManagers.

`layoutManagers` (page 1591)
Returns the receiver's NSLayoutManagers.

Handling Text Edited Messages

`editedInRange` (page 1589)
Tracks changes made to the receiver, allowing the NSTextStorage to record the full extent of changes made.

`ensureAttributesAreFixedInRange` (page 1590)
: An NSTextStorage using lazy attribute fixing is required to call this method before accessing any attributes within *range*.

`fixesAttributesLazily` (page 1590)
: Returns whether the receiver fixes attributes lazily.

`invalidateAttributesInRange` (page 1590)
: Invalidates attributes in *range*.

`processEditing` (page 1591)
: Cleans up changes made to the receiver and notifies its delegate and layout managers of changes.

`replaceCharactersInRange` (page 1591)
: Replaces the characters in *aRange* with the characters of *aString*.

`setAttributesInRange` (page 1592)
: Sets the attributes for the characters in *aRange* to the attributes listed in *attributes*.

Determining the Nature of Changes

`editedMask` (page 1589)
: Returns the kinds of edits pending for the receiver, as a mask containing either or both of `TextStorageEditedAttributes` and `TextStorageEditedCharacters`.

Determining the Extent of Changes

`editedRange` (page 1589)
: Returns the range of the receiver to which pending changes have been made, whether of characters or of attributes.

`changeInLength` (page 1588)
: Returns the difference between the current length of the edited range and its length before editing began (that is, before the receiver was sent the first `beginEditing` message or a single `editedInRange` (page 1589) message).

Setting the Delegate

`setDelegate` (page 1592)
: Sets the receiver's delegate to *anObject*.

`delegate` (page 1588)
: Returns the receiver's delegate.

Processing edit

`textStorageWillProcessEditing` (page 1592) *delegate method*
: Informs the delegate that an NSTextStorage object is about to process edits.

`textStorageDidProcessEditing` (page 1592) *delegate method*
: Informs the delegate that an NSTextStorage object has finished processing edits.

Constructors

NSTextStorage

Creates an empty NSTextStorage.

```
public NSTextStorage()
```

Creates an NSTextStorage with the characters of *string*

```
public NSTextStorage(String aString)
```

Discussion
and no attribute information.

Creates an NSTextStorage with the characters of *aString* and the attributes of *attributes*.

```
public NSTextStorage(String aString, NSDictionary attributes)
```

Creates an NSTextStorage with the characters and attributes of *attributedString*.

```
public NSTextStorage(NSAttributedString attributedString)
```

Creates an NSTextStorage with the contents of *aURL*, returning document properties, which are described in NSAttributedString, in *attributes*.

```
public NSTextStorage(java.net.URL aURL, NSMutableDictionary attributes)
```

Creates an NSTextStorage with the contents of *data*, returning document properties, which are described in NSAttributedString, in *attributes*.

```
public NSTextStorage(NSData data, NSMutableDictionary attributes)
```

Creates an NSTextStorage from *wrapper*, an NSFileWrapper object containing an RTFD document.

```
public NSTextStorage(NSFileWrapper wrapper, NSMutableDictionary attributes)
```

Discussion
Also returns in *attributes* a dictionary containing document-level attributes described in NSAttributedString. Returns `null` if *wrapper* can't be interpreted as an RTFD document.

Creates an NSTextStorage from the HTML contained in *data* and base URL *aURL*.

```
public NSTextStorage(NSData data, java.net.URL aURL, NSMutableDictionary attributes)
```

Discussion
Also returns *attributes*, a dictionary containing document-level attributes described in NSAttributedString. Returns `null` if the file at *aURL* can't be decoded.

Creates an NSTextStorage with the contents of *aString*, returning document properties, which are described in NSAttributedString, in *attributes*.

```
public NSTextStorage(String aString, NSMutableDictionary attributes)
```

Creates an NSTextStorage with the contents of *aURL*, returning document properties, which are described in NSAttributedString, in *attributes*.

```
public NSTextStorage(NSData aURL, NSDictionary options, NSMutableDictionary
    attributes)
```

Discussion
options can contain one of the values described in `readFromURL`.

Instance Methods

addLayoutManager

Adds *aLayoutManager* to the receiver's set of NSLayoutManagers.

```
public void addLayoutManager(NSLayoutManager aLayoutManager)
```

See Also
`removeLayoutManager` (page 1591)
`layoutManagers` (page 1591)

changeInLength

Returns the difference between the current length of the edited range and its length before editing began (that is, before the receiver was sent the first `beginEditing` message or a single `editedInRange` (page 1589) message).

```
public int changeInLength()
```

Discussion
This difference is accumulated with each invocation of `editedInRange` (page 1589), until a final message processes the changes.

The receiver's delegate and layout managers can use this information to determine the nature of edits in their respective notification methods.

See Also
`editedRange` (page 1589)
`editedMask` (page 1589)

delegate

Returns the receiver's delegate.

```
public Object delegate()
```

See Also
`setDelegate` (page 1592)

editedInRange

Tracks changes made to the receiver, allowing the NSTextStorage to record the full extent of changes made.

```
public void editedInRange(int mask, NSRange oldRange, int lengthChange)
```

Discussion

This method invokes `processEditing` (page 1591). NSTextStorage invokes this method automatically each time it makes a change to its attributed string. Subclasses that override or add methods that alter their attributed strings directly should invoke this method after making those changes; otherwise you should not invoke this method. The information accumulated with this method is then used in an invocation of `processEditing` (page 1591) to report the affected portion of the receiver.

The *mask* argument specifies the nature of the changes. Its value is made by combining these options with the C bitwise OR operator:

Option	Meaning
`TextStorageEditedAttributes`	Attributes were added, removed, or changed.
`TextStorageEditedCharacters`	Characters were added, removed, or replaced.

The *oldRange* argument indicates the extent of characters affected before the change took place. If the `TextStorageEditedCharacters` bit of *mask* is set, *lengthChange* gives the number of characters added to or removed from *oldRange* (otherwise its value is irrelevant). For example, when replacing "The" with "Several" in the string "The files couldn't be saved", *oldRange* is {0, 3} and *lengthChange* is 4.

The methods for querying changes, `editedRange` (page 1589) and `changeInLength` (page 1588), indicate the extent of characters affected after the change. This method expects the characters before the change because that information is readily available as the argument to whatever method performs the change (such as `replaceCharactersInRange`).

editedMask

Returns the kinds of edits pending for the receiver, as a mask containing either or both of `TextStorageEditedAttributes` and `TextStorageEditedCharacters`.

```
public int editedMask()
```

Discussion

Use the C bitwise AND operator to test the mask; testing for equality will fail if additional mask flags are added later. The receiver's delegate and layout managers can use this information to determine the nature of edits in their respective notification methods.

See Also

`editedRange` (page 1589)

`changeInLength` (page 1588)

editedRange

Returns the range of the receiver to which pending changes have been made, whether of characters or of attributes.

```
public NSRange editedRange()
```

Discussion
The receiver's delegate and layout managers can use this information to determine the nature of edits in their respective notification methods.

See Also
`changeInLength` (page 1588)
`editedMask` (page 1589)

ensureAttributesAreFixedInRange

An NSTextStorage using lazy attribute fixing is required to call this method before accessing any attributes within *range*.

```
public void ensureAttributesAreFixedInRange(NSRange range)
```

Discussion
This method gives attribute fixing a chance to occur if necessary. NSTextStorage subclasses wishing to support laziness must call this method from all attribute accessors they implement.

See Also
`fixesAttributesLazily` (page 1590)
`invalidateAttributesInRange` (page 1590)

fixesAttributesLazily

Returns whether the receiver fixes attributes lazily.

```
public boolean fixesAttributesLazily()
```

Discussion
This method can control whether an instance fixes attributes lazily by returning `true`. By default, custom NSTextStorage subclasses are not lazy, but the provided concrete subclass is lazy by default.

invalidateAttributesInRange

Invalidates attributes in *range*.

```
public void invalidateAttributesInRange(NSRange range)
```

Discussion
Called from `processEditing` (page 1591) to invalidate attributes when the text storage changes. If the receiver is not lazy, this method simply calls `fixAttributesInRange`. If lazy attribute fixing is in effect, this method instead records the range needing fixing.

See Also
`ensureAttributesAreFixedInRange` (page 1590)
`fixesAttributesLazily` (page 1590)

layoutManagers

Returns the receiver's NSLayoutManagers.

```
public NSArray layoutManagers()
```

See Also

`addLayoutManager` (page 1588)

`removeLayoutManager` (page 1591)

processEditing

Cleans up changes made to the receiver and notifies its delegate and layout managers of changes.

```
public void processEditing()
```

Discussion

This method is automatically invoked in response to an `editedInRange` (page 1589) message. You should never need to invoke it directly.

This method begins by posting a `TextStorageWillProcessEditingNotification` (page 1593) to the default notification center (which results in the delegate receiving a `textStorageWillProcessEditing` (page 1592) message). It then invokes the inherited `fixAttributesInRange` method to fix up attributes after a batch of editing changes. After this, it posts a `TextStorageDidProcessEditingNotification` (page 1593) to the default notification center (which results in the delegate receiving a `textStorageDidProcessEditing` (page 1592) message). Finally, it sends a `textStorageChanged` (page 854) message to each of the receiver's NSLayoutManagers using the argument values provided.

removeLayoutManager

Removes *aLayoutManager* from the receiver's set of NSLayoutManagers.

```
public void removeLayoutManager(NSLayoutManager aLayoutManager)
```

See Also

`addLayoutManager` (page 1588)

`layoutManagers` (page 1591)

replaceCharactersInRange

Replaces the characters in *aRange* with the characters of *aString*.

```
public void replaceCharactersInRange(NSRange aRange, String aString)
```

Discussion

The new characters inherit the attributes of the first replaced character from *aRange*. Where the length of *aRange* is 0, the new characters inherit the attributes of the character preceding *aRange* if it has any, otherwise of the character following *aRange*.

Throws a `RangeException` if any part of *aRange* lies beyond the end of the receiver's characters.

See Also
`deleteCharactersInRange` (NSMutableAttributedString)

setAttributesInRange

Sets the attributes for the characters in *aRange* to the attributes listed in *attributes*.

```
public void setAttributesInRange(NSDictionary attributes, NSRange aRange)
```

Discussion
These new attributes replace any attributes previously associated with the characters in *aRange*. Throws a `RangeException` if any part of *aRange* lies beyond the end of the receiver's characters.

See Also
`addAttributesInRange` (NSMutableAttributedString)
`removeAttributeInRange` (NSMutableAttributedString)

setDelegate

Sets the receiver's delegate to *anObject*.

```
public void setDelegate(Object anObject)
```

See Also
`delegate` (page 1588)

Delegate Methods

textStorageDidProcessEditing

Informs the delegate that an NSTextStorage object has finished processing edits.

```
public abstract void textStorageDidProcessEditing(NSNotification aNotification)
```

Discussion
The text storage object is available by sending `object` to *aNotification*, which is always a `TextStorageDidProcessEditingNotification` (page 1593). The delegate can use this notification to verify the final state of the text storage object; it can't change the text storage object's characters without leaving it in an inconsistent state, but if necessary it can change attributes. Note that even in this case it's possible to put a text storage object into an inconsistent state—for example, by changing the font of a range to one that doesn't support the characters in that range (such as using a Latin font for Kanji text).

textStorageWillProcessEditing

Informs the delegate that an NSTextStorage object is about to process edits.

```
public abstract void textStorageWillProcessEditing(NSNotification aNotification)
```

Discussion

The text storage object is available by sending `object` to *`aNotification`*, which is always a `TextStorageWillProcessEditingNotification` (page 1593). The delegate can use this notification to verify the changed state of the text storage object and to make changes to the text storage object's characters or attributes to enforce whatever constraints it establishes (which doesn't result in this message being sent again, however).

Notifications

TextStorageDidProcessEditingNotification

Posted after an NSTextStorage finishes processing edits in `processEditing` (page 1591).

Observers other than the delegate shouldn't make further changes to the NSTextStorage. The notification object is the NSTextStorage object that processed the edits. This notification does not contain a *`userInfo`* dictionary.

TextStorageWillProcessEditingNotification

Posted before an NSTextStorage finishes processing edits in `processEditing` (page 1591).

Observers other than the delegate shouldn't make further changes to the NSTextStorage. The notification object is the NSTextStorage object that is about to process the edits. This notification does not contain a *`userInfo`* dictionary.

NSTextTab

Inherits from	NSObject
Implements	NSCoding
Package:	com.apple.cocoa.application
Companion guide	Text System Overview

Overview

An NSTextTab represents a tab in an NSParagraphStyle object, storing an alignment type and location. NSTextTabs are most frequently used with the Application Kit's text system and with NSRulerView (page 1209) and NSRulerMarker (page 1201) objects. See the appropriate class specifications for more information on these uses.

The text system supports four alignment types: left, center, right, and decimal (based on the decimal separator character of the locale in effect). These alignment types are absolute, not based on the line sweep direction of text. For example, tabbed text is always positioned to the left of a right-aligned tab, whether the line sweep direction is left to right or right to left. A tab's location, on the other hand, is relative to the back margin. A tab set at 1.5", for example, is at 1.5" from the right in right to left text.

Tasks

Constructors

`NSTextTab` (page 1596)
Creates an empty NSTextTab.

Getting Tab Stop Information

`location` (page 1597)
Returns the receiver's ruler location relative to the back margin.

`tabStopType` (page 1597)
Returns the receiver's tab stop type.

Getting Text Tab Information

`alignment` (page 1596)

`options` (page 1597)
Returns the dictionary of attributes associated with the tab.

Constructors

NSTextTab

Creates an empty NSTextTab.

```
public NSTextTab()
```

Creates an NSTextTab with an alignment of *type* at *location* on the paragraph.

```
public NSTextTab(int type, float location)
```

Discussion
The location is relative to the back margin, based on the line sweep direction of the paragraph. *type* can be any of the values described in "Constants" (page 1597).

Creates an NSTextTab with the text *alignment*, *location*, and *options*.

```
public NSTextTab(int alignment, float loc, NSDictionary options)
```

Discussion
The text alignment is used to determine the position of text inside the tab column. See the "Constants" (page 1597) section for a mapping between alignments and tab stop types.

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

Instance Methods

alignment

```
public int alignment()
```

Discussion
Returns the text alignment of the tab, as defined by the "Constants" (page 1530) of NSText.

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

location

Returns the receiver's ruler location relative to the back margin.

```
public float location()
```

options

Returns the dictionary of attributes associated with the tab.

```
public NSDictionary options()
```

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

tabStopType

Returns the receiver's tab stop type.

```
public int tabStopType()
```

Discussion
The possible values are listed in "Constants" (page 1597).

Constants

These constants describe the various type of tab stops:

Constant	Description
`LeftTabStopType`	A left-aligned tab stop.
`RightTabStopType`	A right-aligned tab stop.
`CenterTabStopType`	A center-aligned tab stop.
`DecimalTabStopType`	Aligns columns of numbers by the decimal point.

The following constant specifies the terminating character for a tab column:

Constant	Description
`TabColumnTerminatorsAttributeName`	The value is an NSCharacterSet. The character set is used to determine the terminating character for a tab column. The tab and newline characters are implied even if they don't exist in the character set. This attribute is optional.

The following mappings define the conversions between text alignment in NSTextTab and tab stop types defined by NSTextTab:

To tab stop type:

Alignment	Tab Stop Type
LeftTextAlignment	LeftTabStopType
RightTextAlignment	RightTabStopType
CenterTextAlignment	CenterTabStopType
JustifiedTextAlignment	LeftTabStopType
NaturalTextAlignment	LeftTabStopType or RightTabStopType, depending on the user setting
RightTextAlignment with terminator	DecimalTabStopType

From tab stop type:

Tab Stop Type	Alignment
LeftTabStopType	LeftTextAlignment
RightTabStopType	RightTextAlignment
CenterTabStopType	CenterTextAlignment
DecimalTabStopType	RightTextAlignment with the decimal character for the user setting

NSTextTable

Inherits from	NSTextBlock
Package:	com.apple.cocoa.application
Availability	Available in Mac OS X v10.4 and later.
Companion guides	Text System Overview Text Layout Programming Guide for Cocoa

Overview

NSTextTable represents a text table as a whole. It is responsible for laying out and drawing the text table blocks it contains, and it maintains the basic parameters of the table.

Tasks

Constructors

`NSTextTable` (page 1600)

Getting and Setting Number of Columns

`numberOfColumns` (page 1602)
Returns the number of columns in the text table.

`setNumberOfColumns` (page 1604)
Sets the number of columns in the text table.

Getting and Setting Layout Algorithm

`layoutAlgorithm` (page 1602)
Returns the text table layout algorithm.

`setLayoutAlgorithm` (page 1603)
Sets the text table layout algorithm.

Collapsing Borders

`collapsesBorders` (page 1601)
Returns a Boolean value indicating whether or not the text table borders are collapsible.

`setCollapsesBorders` (page 1603)
Sets whether or not the text table borders are collapsible.

Hiding Empty Cells

`hidesEmptyCells` (page 1602)
Returns a Boolean value indicating whether or not the text table hides empty cells, allowing the background of the enclosing block orr text container to show through.

`setHidesEmptyCells` (page 1603)
Sets whether or not the text table hides empty cells, allowing the background of the enclosing block orr text container to show through.

Determining Layout Rectangles

`rectForBlockLayoutAtPoint` (page 1602)
Called by the text table block *`block`* to determine the rectangle within which glyphs should be laid out for the text table block.

`boundsRectForBlock` (page 1601)
Called by the text table block *`block`* after it is laid out to determine the rectangle the text table block actually occupies, including padding, borders, and margins.

Drawing the Table

`drawBackgroundForBlock` (page 1601)
Called by the text table block *`block`* to draw any colors and other decorations before the text is drawn.

Constructors

NSTextTable

```
public NSTextTable()
```

Discussion
The constructor for the NSTextTable object.

Instance Methods

boundsRectForBlock

Called by the text table block *block* after it is laid out to determine the rectangle the text table block actually occupies, including padding, borders, and margins.

```
public NSRect boundsRectForBlock(NSTextTableBlock block, NSRect contentRect, NSRect
     rect, NSTextContainer textContainer, NSRange charRange)
```

Discussion

The *contentRect* is the rectangle in which the text was laid out. The *rect* is the initial rectangle in *textContainer* proposed by the typesetter in which to lay out the characters in *charRange*.

Availability

Available in Mac OS X v10.4 and later.

See Also

`rectForBlockLayoutAtPoint` (page 1602)

collapsesBorders

Returns a Boolean value indicating whether or not the text table borders are collapsible.

```
public boolean collapsesBorders()
```

Availability

Available in Mac OS X v10.4 and later.

See Also

`setCollapsesBorders` (page 1603)

drawBackgroundForBlock

Called by the text table block *block* to draw any colors and other decorations before the text is drawn.

```
public void drawBackgroundForBlock(NSTextTableBlock block, NSRect frameRect, NSView
     controlView, NSRange charRange, NSLayoutManager layoutManager)
```

Discussion

The *frameRect* describes the area in which drawing occurs. The *controlView* is the view controlling the drawing. The *charRange* describes the characters whose glyphs are to be drawn, and the *layoutManager* is the layout manager controlling the typesetting.

Availability

Available in Mac OS X v10.4 and later.

hidesEmptyCells

Returns a Boolean value indicating whether or not the text table hides empty cells, allowing the background of the enclosing block orr text container to show through.

```
public boolean hidesEmptyCells()
```

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

See Also
`setHidesEmptyCells` (page 1603)

layoutAlgorithm

Returns the text table layout algorithm.

```
public int layoutAlgorithm()
```

Discussion
The method returns one of the values listed in "Constants" (page 1604).

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

See Also
`setLayoutAlgorithm` (page 1603)

numberOfColumns

Returns the number of columns in the text table.

```
public int numberOfColumns()
```

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

See Also
`setNumberOfColumns` (page 1604)

rectForBlockLayoutAtPoint

Called by the text table block *block* to determine the rectangle within which glyphs should be laid out for the text table block.

```
public NSRect rectForBlockLayoutAtPoint(NSTextTableBlock block, NSPoint
    startingPoint, NSRect rect, NSTextContainer textContainer, NSRange charRange)
```

Discussion

The `startingPoint` argument specifies the location, in container coordinates, where layout begins. The `rect` is the rectangle in which the block is constrained to lie: for top-level blocks, the container rectangle of `textContainer`; for nested blocks, the layout rectangle of the enclosing block. The `charRange` argument is the range of the characters to be laid out.

Availability

Available in Mac OS X v10.4 and later.

See Also

`boundsRectForBlock` (page 1601)

setCollapsesBorders

Sets whether or not the text table borders are collapsible.

```
public void setCollapsesBorders(boolean flag)
```

Discussion

If `flag` is `true`, the borders are collapsible.

Availability

Available in Mac OS X v10.4 and later.

See Also

`collapsesBorders` (page 1601)

setHidesEmptyCells

Sets whether or not the text table hides empty cells, allowing the background of the enclosing block orr text container to show through.

```
public void setHidesEmptyCells(boolean flag)
```

Discussion

If `flag` is `true`, empty cells are hidden.

Availability

Available in Mac OS X v10.4 and later.

See Also

`hidesEmptyCells` (page 1602)

setLayoutAlgorithm

Sets the text table layout algorithm.

```
public void setLayoutAlgorithm(int algorithm)
```

Discussion

The algorithm argument can be one of the values listed in "Constants" (page 1604).

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

See Also
`layoutAlgorithm` (page 1602)

setNumberOfColumns

Sets the number of columns in the text table.

```
public void setNumberOfColumns(int numCols)
```

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

See Also
`numberOfColumns` (page 1602)

Constants

The following constants specify values used with method parameters used for specifying the layout algorithm.

Constant	Description
`AutomaticLayoutAlgorithm`	Specifies automatic layout algorithm
`FixedLayoutAlgorithm`	Specifies fixed layout algorithm

NSTextTableBlock

Inherits from	NSTextBlock
Package:	com.apple.cocoa.application
Availability	Available in Mac OS X v10.4 and later.
Companion guides	Text System Overview Text Layout Programming Guide for Cocoa

Overview

NSTextTableBlock represents a text block that appears as a cell in a text table.

Tasks

Constructors

`NSTextTableBlock` (page 1606)

Getting the Block's Enclosing Table

`table` (page 1607)
Returns a pointer to the table containing this text table block.

Getting Information About the Block's Position in Its Enclosing Table

`startingRow` (page 1607)
Returns the table row at which this text table block starts.

`rowSpan` (page 1606)
Returns the number of table rows spanned by this text table block.

`startingColumn` (page 1606)
Returns the table column at which this text table block starts.

`columnSpan` (page 1606)
Returns the number of table columns spanned by this text table block.

Constructors

NSTextTableBlock

```
public NSTextTableBlock()
```

Discussion
Creates a new NSTextTableBlock object.

```
public NSTextTableBlock(NSTextTable table, int row, int rowSpan, int col, int
    colSpan)
```

Discussion
Creates a new NSTextTableBlock object from an existing object. The *table* argument points to the text table containing this text table block. The *row* argument specifies the table row at which the text table block starts, and the *rowSpan* argument specifies how many rows it covers. The *col* argument specifies the table column at which the text table block starts, and the *colSpan* argument specifies how many columns it covers.

Instance Methods

columnSpan

Returns the number of table columns spanned by this text table block.

```
public int columnSpan()
```

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

rowSpan

Returns the number of table rows spanned by this text table block.

```
public int rowSpan()
```

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

startingColumn

Returns the table column at which this text table block starts.

```
public int startingColumn()
```

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

startingRow

Returns the table row at which this text table block starts.

```
public int startingRow()
```

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

table

Returns a pointer to the table containing this text table block.

```
public NSTextTable table()
```

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

NSTextView

Inherits from	NSText : NSView : NSResponder : NSObject
Implements	NSTextInput NSCoding (NSResponder)
Package:	com.apple.cocoa.application
Companion guides	Text System Overview Text System User Interface Layer Programming Guide for Cocoa

Class at a Glance

NSTextView is the front-end component of the Application Kit's text system. It displays and manipulates text laid out in an area defined by an NSTextContainer and adds many features to those defined by its superclass, NSText. Many of the methods that you'll use most frequently are declared by the superclass; see the NSText (page 1505) class specification for details.

Instances of this class can be created using Interface Builder or using one of its constructors.

Commonly Used Methods

The methods most commonly used with NSTextView objects are declared in NSText, the superclass. These methods provide access to the other major components of the text system:

`textStorage` (page 1659)
Returns the associated NSTextStorage object.

`textContainer` (page 1659)
Returns the associated NSTextContainer object.

`layoutManager` (page 1632)
Returns the associated NSLayoutManager object.

Overview

NSTextView is the front-end class to the Application Kit's text-handling system. It draws the text managed by the back-end components and handles user events to select and modify its text. NSTextView is the principal means to obtain a text object that caters to almost all needs for displaying and managing text at the user interface level. While NSTextView is a subclass of NSText—which declares the most general Cocoa interface to the text system—NSTextView adds major features beyond the capabilities of NSText.

NSTextView communicates with its delegate through methods declared both by NSTextView and by its superclass, NSText. See the NSText (page 1505) class specification for those other delegate methods. Note that all delegation messages come from the first text view.

Interfaces Implemented

NSTextInput

- `attributedSubstringWithRange` (page 2026)
- `characterIndexForPoint` (page 2026)
- `conversationIdentifier` (page 2027)
- `doCommandBySelector` (page 2027)
- `firstRectForCharacterRange` (page 2027)
- `hasMarkedText` (page 2027)
- `insertText` (page 2028)
- `markedRange` (page 2028)
- `selectedRange` (page 2028)
- `setMarkedTextAndSelectedRange` (page 2028)
- `unmarkText` (page 2029)
- `validAttributesForMarkedText` (page 2029)

Tasks

Constructors

`NSTextView` (page 1622)
Creates an NSTextView with a zero-sized frame rectangle.

Registering Services Information

`registerForServices` (page 1623)
Registers send and return types for the Services facility.

Accessing Related Text System Objects

`characterIndexForPoint` (page 1626)
Returns the index for the character that is nearest to *`thePoint`*.

`conversationIdentifier` (page 1627)
Returns a number used to identify the receiver’s input management session to the input server.

`firstRectForCharacterRange` (page 1629)

`setTextContainer` (page 1653)
    Sets the receiver's text container to *aTextContainer*.

`replaceTextContainer` (page 1640)
    Replaces the NSTextContainer for the group of text system objects containing the receiver with *aTextContainer*, keeping the association between the receiver and its layout manager intact, unlike `setTextContainer` (page 1653).

`textContainer` (page 1659)
    Returns the receiver's text container.

`setTextContainerInset` (page 1654)
    Sets the empty space the receiver leaves around its associated text container to *inset*.

`textContainerInset` (page 1659)
    Returns the empty space the receiver leaves around its text container.

`textContainerOrigin` (page 1659)
    Returns the origin of the receiver's text container, which is calculated from the receiver's bounds rectangle, container inset, and the container's used rect.

`invalidateTextContainerOrigin` (page 1631)
    Informs the receiver that it needs to recalculate the origin of its text container, usually because it's been resized or the contents of the text container have changed.

`layoutManager` (page 1632)
    Returns the NSLayoutManager that lays out text for the receiver's text container, or `null` if there's no such object (which is the case when a text view isn't linked into a group of text objects).

`textStorage` (page 1659)
    Returns the receiver's text storage object.

Setting Graphics Attributes

`setBackgroundColor` (page 1646)
    Sets the receiver's background color to *aColor*.

`backgroundColor` (page 1624)

`setDrawsBackground` (page 1648)
    Controls whether the receiver draws its background.

`drawsBackground` (page 1629)
    Returns `true` if the receiver draws its background, `false` if it doesn't.

`setAllowsDocumentBackgroundColorChange` (page 1646)
    Sets whether or not the receiver allows its background color to change.

`allowsDocumentBackgroundColorChange` (page 1624)
    Returns `true` if the receiver allows the its background color to change, otherwise `false`.

`changeDocumentBackgroundColor` (page 1626)

Controlling Display

`setNeedsDisplay` (page 1650)
Marks the receiver as requiring display within *aRect*.

`shouldDrawInsertionPoint` (page 1656)
Returns `true` if the receiver should draw its insertion point, `false` if the insertion point can't or shouldn't be drawn.

`drawInsertionPointInRect` (page 1628)

`drawViewBackgroundInRect` (page 1629)
Called when the text view intends to draw its background

`setConstrainedFrameSize` (page 1647)
Attempts to set the frame size for the receiver to *desiredSize*, constrained by the receiver's existing minimum and maximum sizes and by whether resizing is permitted.

`cleanUpAfterDragOperation` (page 1626)
Releases the drag information still existing after the dragging session has completed.

Inserting Text

`insertText` (page 1630)
Inserts *aString* into the receiver's text at the insertion point if there is one, otherwise replacing the selection.

Setting Behavioral Attributes

`allowsUndo` (page 1624)
Returns `true` if the receiver allows undo, otherwise `false`.

`setAllowsUndo` (page 1646)
If *flag* is `true` enables undo support; otherwise disables it.

`setEditable` (page 1648)
Controls whether the text views sharing the receiver's NSLayoutManager allow the user to edit text.

`isEditable` (page 1631)
Returns `true` if the text views sharing the receiver's NSLayoutManager allow the user to edit text, `false` if they don't.

`setSelectable` (page 1651)
Controls whether the text views sharing the receiver's NSLayoutManager allow the user to select text.

`isSelectable` (page 1632)
Returns `true` if the text views sharing the receiver's NSLayoutManager allow the user to select text, `false` if they don't.

`setFieldEditor` (page 1648)
Controls whether the text views sharing the receiver's NSLayoutManager interpret Tab, Shift-Tab, and Return (Enter) as cues to end editing and possibly to change the first responder.

`isFieldEditor` (page 1631)
: Returns `true` if the text views sharing the receiver's NSLayoutManager interpret Tab, Shift-Tab, and Return (Enter) as cues to end editing and possibly to change the first responder; `false` if they accept them as text input.

`setRichText` (page 1650)
: Controls whether the text views sharing the receiver's NSLayoutManager allow the user to apply attributes to specific ranges of the text.

`isRichText` (page 1632)
: Returns `true` if the text views sharing the receiver's NSLayoutManager allow the user to apply attributes to specific ranges of the text, `false` if they don't.

`setImportsGraphics` (page 1649)
: Controls whether the text views sharing the receiver's NSLayoutManager allow the user to import files by dragging.

`importsGraphics` (page 1629)
: Returns `true` if the text views sharing the receiver's NSLayoutManager allow the user to import files by dragging, `false` if they don't.

`setBaseWritingDirection` (page 1647)
: Sets the base writing direction of the text in *`range`* to *`writingDirection`*.

`setDefaultParagraphStyle` (page 1647)
: Sets the receiver's default paragraph style.

`defaultParagraphStyle` (page 1627)
: Returns the receiver's default paragraph style.

`outline` (page 1635)
: Adds the outline attribute to the selected text attributes if absent; removes the attribute if not.

`underline` (page 1661)

Using the Ruler

`setUsesRuler` (page 1655)
: Controls whether the text views sharing the receiver's NSLayoutManager use an NSRulerView and respond to Format menu commands.

`usesRuler` (page 1663)
: Returns `true` if the text views sharing the receiver's NSLayoutManager use a ruler view, `false` otherwise.

`setRulerVisible` (page 1651)
: Controls whether the scroll view enclosing text views sharing the receiver's NSLayoutManager displays the ruler.

`isRulerVisible` (page 1632)
: Returns `true` if the scroll view enclosing the text views sharing the receiver's NSLayoutManager shows its ruler, `false` otherwise.

Managing the Selection

`attributedSubstringWithRange` (page 1624)
Returns attributed string at *theRange*.

`selectedRange` (page 1644)
Returns the range of characters selected in the receiver's layout manager.

`selectedRanges` (page 1644)
Returns an array containing the ranges of characters selected in the receiver's layout manager.

`setSelectedRange` (page 1651)

`setSelectedRanges` (page 1652)

`selectionAffinity` (page 1644)
Returns the preferred direction of selection, either `SelectionAffinityUpstream` or `SelectionAffinityDownstream`.

`setMarkedTextAndSelectedRange` (page 1650)
Replaces text in *aRange* within receiver's text storage with the contents of *aString*, which the receiver must display distinctively to indicate that it is marked text.

`setSelectionGranularity` (page 1653)
Sets the selection granularity for subsequent extension of a selection to *granularity*.

`selectionGranularity` (page 1645)
Returns the current selection granularity, used during mouse tracking to modify the range of the selection.

`setInsertionPointColor` (page 1649)
Sets the color of the insertion point to *aColor*.

`insertionPointColor` (page 1630)
Returns the color used to draw the insertion point.

`updateInsertionPointStateAndRestartTimer` (page 1662)
Updates the insertion point's location and, if *flag* is `true`, restarts the blinking cursor timer.

`hasMarkedText` (page 1629)
Returns `true` if the receiver has text that's still being interpreted by the input manager, `false` if it doesn't.

`setSelectedTextAttributes` (page 1652)
Sets the attributes used to indicate the selection to *attributes*.

`selectedTextAttributes` (page 1644)
Returns the attributes used to indicate the selection.

`markedRange` (page 1633)
Returns the range of marked text.

`setMarkedTextAttributes` (page 1650)
Sets the attributes used to draw marked text to *attributes*.

`markedTextAttributes` (page 1634)
Returns the attributes used to draw marked text.

`unmarkText` (page 1661)
Removes any marking from pending input text and accepts the text in its current state.

`validAttributesForMarkedText` (page 1664)
Returns an array of NSString names for the attributes supported by the receiver.

`setLinkTextAttributes` (page 1649)
Sets the attributes corresponding to the onscreen generation of link text.

`linkTextAttributes` (page 1633)
Returns the attributes corresponding to the onscreen generation of link text.

Managing the Pasteboard

`preferredPasteboardTypeFromArray` (page 1636)
Returns whatever type on the pasteboard would be most preferred for copying data.

`readSelectionFromPasteboard` (page 1640)
Reads the text view's preferred type of data from the pasteboard specified by the *pboard* parameter.

`readSelectionFromPasteboardOfType` (page 1640)
Reads data of the given *type* from *pboard*.

`readablePasteboardTypes` (page 1639)
Returns an array of strings describing the types this text view can read immediately from the pasteboard.

`writablePasteboardTypes` (page 1664)

`writeSelectionToPasteboardOfType` (page 1664)
Writes the current selection to *pboard* using the given *type*.

`writeSelectionToPasteboardOfTypes` (page 1665)
Writes the current selection to *pboard* under each type in the *types* array.

Setting Text Attributes

`alignJustified` (page 1623)
This action method applies full justification to selected paragraphs (or all text, if the receiver is a plain text object).

`changeAttributes` (page 1625)
This action method changes the attributes of the current selection.

`changeColor` (page 1625)
Invoked by the NSColorPanel *sender* to set the color of the selected text.

`setAlignmentInRange` (page 1645)
Sets the alignment of the paragraphs containing characters in *aRange* to *alignment*.

`setTypingAttributes` (page 1654)
Sets the receiver's typing attributes to *attributes*.

`typingAttributes` (page 1660)
Returns the current typing attributes.

`useStandardKerning` (page 1663)
This action method causes the receiver to use pair kerning data for the glyphs in its selection, or for all glyphs if the receiver is a plain text view.

lowerBaseline (page 1633)
: This action method lowers the baseline offset of selected text by 1 point, or of all text if the receiver is a plain text view.

raiseBaseline (page 1636)
: This action method raises the baseline offset of selected text by 1 point, or of all text if the receiver is a plain text view.

turnOffKerning (page 1660)
: This action method causes the receiver to use nominal glyph spacing for the glyphs in its selection, or for all glyphs if the receiver is a plain text view.

loosenKerning (page 1633)
: This action method increases the space between glyphs in the receiver's selection, or in all text if the receiver is a plain text view.

tightenKerning (page 1659)
: This action method decreases the space between glyphs in the receiver's selection, or for all glyphs if the receiver is a plain text view.

useStandardLigatures (page 1663)
: This action method causes the receiver to use the standard ligatures available for the fonts and languages used when setting text, for the glyphs in the selection if the receiver is a rich text view, or for all glyphs if it's a plain text view.

turnOffLigatures (page 1660)
: This action method causes the receiver to use only required ligatures when setting text, for the glyphs in the selection if the receiver is a rich text view, or for all glyphs if it's a plain text view.

useAllLigatures (page 1662)
: This action method causes the receiver to use all ligatures available for the fonts and languages used when setting text, for the glyphs in the selection if the receiver is a rich text view, or for all glyphs if it's a plain text view.

toggleTraditionalCharacterShape (page 1660)
: This action method toggles the NSCharacterShapeAttibuteName attribute at the current selection.

Other Action Methods

clickedOnLinkAtIndex (page 1626)
: Notifies the delegate that the user clicked a link at the specified *charIndex*.

doCommandBySelector (page 1627)
: Attempts to invoke *aSelector* or pass the message up the responder chain.

pasteAsPlainText (page 1635)
: This action method inserts the contents of the pasteboard into the receiver's text as plain text, in the manner of insertText (page 1630).

pasteAsRichText (page 1635)
: This action method inserts the contents of the pasteboard into the receiver's text as rich text, maintaining its attributes.

Undo Support

breakUndoCoalescing (page 1625)

Methods That Subclasses Should Use or Override

updateFontPanel (page 1661)
Updates the Font panel to contain the font attributes of the selection.

updateRuler (page 1662)
Updates the NSRulerView in the receiver's enclosing scroll view to reflect the selection's paragraph and marker attributes.

acceptableDragTypes (page 1623)
Returns the data types that the receiver accepts as the destination view of a dragging operation.

updateDragTypeRegistration (page 1661)

selectionRangeForProposedRange (page 1645)
Adjusts the *proposedSelRange* if necessary, based on *granularity*.

rangeForUserCharacterAttributeChange (page 1636)
Returns the range of characters affected by an action method that changes character (not paragraph) attributes, such as the NSText action method changeFont (page 1513).

rangesForUserCharacterAttributeChange (page 1638)
Returns an array containing the ranges of characters affected by an action method that changes character (not paragraph) attributes, such as the NSText action method changeFont (page 1513).

rangeForUserParagraphAttributeChange (page 1637)
Returns the range of characters affected by a method that changes paragraph (not character) attributes, such as the NSText action method alignLeft (page 1512).

rangesForUserParagraphAttributeChange (page 1638)
Returns an array containing the ranges of characters affected by a method that changes paragraph (not character) attributes, such as the NSText action method alignLeft (page 1512).

rangeForUserTextChange (page 1638)
Returns the range of characters affected by a method that changes characters (as opposed to attributes), such as insertText (page 1630).

rangesForUserTextChange (page 1639)
Returns an array containing the ranges of characters affected by a method that changes characters (as opposed to attributes), such as insertText (page 1630).

shouldChangeTextInRange (page 1655)
Initiates a series of delegate messages (and general notifications) to determine whether modifications can be made to the receiver's text.

shouldChangeTextInRanges (page 1656)
Initiates a series of delegate messages (and general notifications) to determine whether modifications can be made to the receiver's text.

didChangeText (page 1627)
Invoked automatically at the end of a series of changes, this method posts aTextDidChangeNotification (page 1532) to the default notification center, which also results in the delegate receiving an NSText delegate textDidChange (page 1531) message.

`setSmartInsertDeleteEnabled` (page 1653)
: Controls whether the receiver inserts or deletes space around selected words so as to preserve proper spacing and punctuation.

`smartInsertDeleteEnabled` (page 1658)
: Returns `true` if the receiver inserts or deletes space around selected words so as to preserve proper spacing and punctuation, `false` if it inserts and deletes exactly what's selected.

`smartDeleteRangeForProposedRange` (page 1657)
: Given *proposedCharRange*, returns an extended range that includes adjacent whitespace that should be deleted along with the proposed range in order to preserve proper spacing and punctuation of the text surrounding the deletion.

`smartInsertAfterStringForString` (page 1657)
: Returns any whitespace that needs to be added after *aString* to preserve proper spacing and punctuation when *aString* is inserted into the receiver's text over *charRange*.

`smartInsertBeforeStringForString` (page 1657)
: Returns any whitespace that needs to be added before *aString* to preserve proper spacing and punctuation when *aString* is inserted into the receiver's text over *charRange*.

Changing First Responder Status

`becomeFirstResponder` (page 1625)
: Informs the receiver that it's becoming the first responder.

`resignFirstResponder` (page 1641)
: Notifies the receiver that it's been asked to relinquish its status as first responder in its NSWindow.

`validRequestorForTypes` (page 1664)
: Returns `this` if *sendType* specifies a type of data the text view can put on the pasteboard and *returnType* contains a type of data the text view can read from the pasteboard; otherwise returns `null`.

Working with the Spelling Checker

`isContinuousSpellCheckingEnabled` (page 1631)
: Returns `true` if the object has continuous spell checking enabled, otherwise `false`.

`setContinuousSpellCheckingEnabled` (page 1647)
: If *flag* is `true` enables continuous spell checking; otherwise disables it.

`spellCheckerDocumentTag` (page 1658)
: Returns a tag identifying the NSTextView text as a document for the spell checker server.

`toggleContinuousSpellChecking` (page 1660)
: This action method toggles whether continuous spell checking is enabled for the receiver.

NSRulerView Client Methods

`rulerViewDidMoveMarker` (page 1641)

rulerViewDidRemoveMarker (page 1642)

rulerViewDidAddMarker (page 1641)

rulerViewShouldMoveMarker (page 1643)

rulerViewShouldAddMarker (page 1642)

rulerViewWillMoveMarker (page 1643)

rulerViewShouldRemoveMarker (page 1643)

rulerViewWillAddMarker (page 1643)

rulerViewHandleMouseDown (page 1642)

Assigning a Delegate

setDelegate (page 1648)
: Sets the delegate for all NSTextViews sharing the receiver's NSLayoutManager to *anObject*, without retaining it.

delegate (page 1627)
: Returns the delegate used by the receiver (and by all other NSTextViews sharing the receiver's NSLayoutManager), or null if there is none.

Dragging

dragOperationForDraggingInfo (page 1628)
: Returns the type of drag operation that should be performed if the image were released now.

Speech Support

startSpeaking (page 1658)
: This action method speaks the selected text, or all text if no selection.

stopSpeaking (page 1658)
: This action method stops the speaking of text.

Working with Panels

setUsesFontPanel (page 1655)
: Controls whether the text views sharing the receiver's NSLayoutManager use the Font panel and Font menu.

`usesFontPanel` (page 1663)
: Returns `true` if the text views sharing the receiver's NSLayoutManager use the Font panel, `false` otherwise.

`setUsesFindPanel` (page 1654)
: Specifies whether the receiver allows for a find panel.

`usesFindPanel` (page 1662)
: Returns whether the receiver allows for a find panel.

`performFindPanelAction` (page 1635)
: This is the generic action method for the find menu and find panel, and can be overridden to provide a custom find panel

`orderFrontLinkPanel` (page 1634)
: Brings forward a panel allowing the user to manipulate links in the text view.

`orderFrontListPanel` (page 1634)
: Brings forward a panel allowing the user to manipulate text lists in the text view.

`orderFrontSpacingPanel` (page 1634)
: Brings forward a panel allowing the user to manipulate text line heights, interline spacing, and paragraph spacing, in the text view.

`orderFrontTablePanel` (page 1634)
: Brings forward a panel allowing the user to manipulate text tables in the text view.

Text Completion

`insertCompletion` (page 1630)

`rangeForUserCompletion` (page 1637)
: Returns the partial range from the most recent beginning of a word up to the insertion point.

Clicking cells and links

`textViewClickedCell` (page 1667) *delegate method*
: Invoked after the user clicks on *`attachmentCell`* within *`cellFrame`* in *`aTextView`* and the cell wants to track the mouse.

`textViewClickedCellAtIndex` (page 1667) *delegate method*
: Invoked after the user clicks on cell within *`cellFrame`* at the specified *`charIndex`* in an NSTextView and the cell wants to track the mouse.

`textViewDoubleClickedCell` (page 1669) *delegate method*
: Invoked when the user double-clicks *`attachmentCell`* within *`cellFrame`* in *`aTextView`* and the cell wants to track the mouse.

`textViewDoubleClickedCellAtIndex` (page 1669) *delegate method*
: Invoked when the user double-clicks *`cell`* within *`cellFrame`* at the specified *`charIndex`* in an NSTextView and the cell wants to track the mouse.

`textViewClickedOnLink` (page 1667) *delegate method*
: Invoked after the user clicks *`link`* in *`aTextView`* if the delegate does not respond to the `textViewClickedOnLinkAtIndex` (page 1668) message.

`textViewClickedOnLinkAtIndex` (page 1668) *delegate method*
: Invoked after the user clicks `link` at the specified `charIndex` in an NSTextView.

Selecting

`textViewWillChangeSelection` (page 1671) *delegate method*
: Invoked before an NSTextView finishes changing the selection—that is, when the last argument to a `setSelectedRange` (page 1651) message is `false`.

`textViewDidChangeSelection` (page 1668) *delegate method*
: Invoked when the selection changes in the NSTextView.

Dragging cells

`textViewDraggedCell` (page 1669) *delegate method*
: Invoked when the user attempts to drag `cell` from `aRect` within `aTextView` and `cell` wants to track the mouse.

`textViewDraggedCellAtIndex` (page 1670) *delegate method*
: Invoked when the user attempts to drag `cell` from `aRect` within `aTextView` and the cell wants to track the mouse.

Editing text and attributes

`textViewShouldChangeTextInRange` (page 1670) *delegate method*
: Invoked when an NSTextView needs to determine if text in the range `affectedCharRange` should be changed.

`textViewShouldChangeTextInRanges` (page 1670) *delegate method*
: Invoked when an NSTextView needs to determine if text in the range `affectedRanges` should be changed.

`textViewShouldChangeTypingAttributes` (page 1671) *delegate method*
: Allows the delegate to intervene to allow, prevent, or modify changes to the typing attributes in `textView` from those contained in `oldTypingAttributes` to those contained in `newTypingAttributes`.

`textViewDidChangeTypingAttributes` (page 1668) *delegate method*
: Delegate method invoked when text view's typing attributes change.

Obtaining undo manager

`undoManagerForTextView` (page 1672) *delegate method*
: Returns the undo manager instance for the text view specified by `aTextView`.

Performing commands

`textViewDoCommandBySelector` (page 1668) *delegate method*
Sent from NSTextView's `doCommandBySelector` (page 1627), this method allows the delegate to perform the command for the text view.

Working with pasteboards

`textView` (page 1666) *delegate method*

Working with tool tips

`textViewWillDisplayToolTip` (page 1672) *delegate method*
Allows the delegate to modify the tool tip that will be displayed from that specified by `NSToolTipAttributeName`, or to suppress display of the tooltip (by returning `null`).

Text completion

`textView:completionsForPartialWordRange` (page 1666) *delegate method*
Allows the delegate to modify the list of completions that will be presented for the partial word at the given range

Constructors

NSTextView

Creates an NSTextView with a zero-sized frame rectangle.

```
public NSTextView()
```

Creates an NSTextView object with *`frameRect`* as its frame rectangle.

```
public NSTextView(NSRect frameRect)
```

Discussion
This method creates the entire collection of objects associated with an NSTextView—its NSTextContainer, NSLayoutManager, and NSTextStorage—and invokes the following constructor that takes a *`frameRect`* and *`aTextContainer`*.

This method creates the text web in such a manner that the NSTextView object is the principal owner of the objects in the web.

Creates an NSTextView object with *`frameRect`* as its frame rectangle and *`aTextContainer`* as its text container.

```
public NSTextView(NSRect frameRect, NSTextContainer aTextContainer)
```

Discussion
Unlike the constructor that takes only a *frameRect*, which builds up an entire group of text-handling objects, you use this constructor after you've created the other components of the text-handling system—an NSTextStorage object, an NSLayoutManager object, and an NSTextContainer object. Assembling the components in this fashion means that the NSTextStorage, not the NSTextView, is the principal owner of the component objects.

Static Methods

registerForServices

Registers send and return types for the Services facility.

```
public static void registerForServices()
```

Discussion
This method is invoked automatically when the first instance of an NSTextView is created; you should never need to invoke it directly.

Subclassing NSTextView is necessary to add support for new service types (in addition to actually supporting writing or reading the types, of course). Override "registerForServices" to call super and then register your own new types.

Instance Methods

acceptableDragTypes

Returns the data types that the receiver accepts as the destination view of a dragging operation.

```
public NSArray acceptableDragTypes()
```

Discussion
These types are automatically registered as necessary by the NSTextView. Subclasses should override this method as necessary to add their own types to those returned by NSTextView's implementation. They must then also override the appropriate methods of the "NSDraggingDestination" (page 1955) interface to support import of those types. See that interface's specification for more information.

See Also
updateDragTypeRegistration (page 1661)

alignJustified

This action method applies full justification to selected paragraphs (or all text, if the receiver is a plain text object).

```
public void alignJustified(Object sender)
```

See Also

`alignCenter` (page 1512) (NSText)

`alignLeft` (page 1512) (NSText)

`alignRight` (page 1513) (NSText)

`alignment` (page 1513) (NSText)

`setAlignment` (page 1522) (NSText)

allowsDocumentBackgroundColorChange

Returns `true` if the receiver allows the its background color to change, otherwise `false`.

```
public boolean allowsDocumentBackgroundColorChange()
```

Discussion

This corresponds to the background color of the entirety of the text view, not just to a selected range of text.

Availability

Available in Mac OS X v10.3 and later.

See Also

`setAllowsDocumentBackgroundColorChange` (page 1646)

`changeDocumentBackgroundColor` (page 1626)

allowsUndo

Returns `true` if the receiver allows undo, otherwise `false`.

```
public boolean allowsUndo()
```

See Also

`setAllowsUndo` (page 1646)

attributedSubstringWithRange

Returns attributed string at *theRange*.

```
public NSAttributedString attributedSubstringWithRange(NSRange theRange)
```

Discussion

This allows input mangers to query any range in backing store.

backgroundColor

```
public NSColor backgroundColor()
```

Discussion

Returns the receiver’s background color.

See Also

`drawsBackground` (page 1629)

`setBackgroundColor` (page 1646)

becomeFirstResponder

Informs the receiver that it's becoming the first responder.

```
public boolean becomeFirstResponder()
```

Discussion

If the previous first responder was not an NSTextView on the same NSLayoutManager as the receiving NSTextView, this method draws the selection and updates the insertion point if necessary. Returns `true`.

Use NSWindow's `makeFirstResponder` (page 1841), not this method, to make an NSTextView the first responder. Never invoke this method directly.

See Also

`resignFirstResponder` (page 1641)

breakUndoCoalescing

```
public void breakUndoCoalescing()
```

Discussion

Informs the receiver that it should begin coalescing sucessive typing operations in a new undo grouping. Typically this is invoked when saving the receiver's contents.

Availability

Available in Mac OS X v10.4 and later.

changeAttributes

This action method changes the attributes of the current selection.

```
public void changeAttributes(Object sender)
```

Discussion

This method changes the attributes by invoking `convertAttributes` (page 674) on *sender* and applying the returned attributes to the appropriate text. See the "NSFontManager" (page 667) class reference for more information on attribute conversion.

Availability

Available in Mac OS X v10.3 and later.

changeColor

Invoked by the NSColorPanel *sender* to set the color of the selected text.

```
public void changeColor(Object sender)
```

Discussion

NSTextView's implementation queries sender for the color by sending it a `color` (page 390) message.

changeDocumentBackgroundColor

```
public void changeDocumentBackgroundColor(Object sender)
```

Discussion

Invoked by the NSColorPanel *sender* to set the background color of the selected text. NSTextView's implementation queries *sender* by sending it a `color` (page 390) message.

This will only set the background color if `allowsDocumentBackgroundColorChange` (page 1624) returns `true`.

Availability

Available in Mac OS X v10.3 and later.

See Also

`setAllowsDocumentBackgroundColorChange` (page 1646)
`allowsDocumentBackgroundColorChange` (page 1624)

characterIndexForPoint

Returns the index for the character that is nearest to *thePoint*.

```
public int characterIndexForPoint(NSPoint thePoint)
```

Discussion

thePoint is in the screen coordinate system.

cleanUpAfterDragOperation

Releases the drag information still existing after the dragging session has completed.

```
public void cleanUpAfterDragOperation()
```

Discussion

Subclasses may override this method to clean up any additional data structures used for dragging. In your overridden method, be sure to invoke the `super`'s implementation of this method.

clickedOnLinkAtIndex

Notifies the delegate that the user clicked a link at the specified *charIndex*.

```
public void clickedOnLinkAtIndex(Object link, int charIndex)
```

Discussion

The delegate may take any appropriate actions to handle the click in its `textViewClickedOnLinkAtIndex` (page 1668) method.

See Also
`textViewClickedOnLinkAtIndex` (page 1668) (delegate method)

conversationIdentifier

Returns a number used to identify the receiver's input management session to the input server.

```
public int conversationIdentifier()
```

defaultParagraphStyle

Returns the receiver's default paragraph style.

```
public NSParagraphStyle defaultParagraphStyle()
```

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

See Also
`setDefaultParagraphStyle` (page 1647)

delegate

Returns the delegate used by the receiver (and by all other NSTextViews sharing the receiver's NSLayoutManager), or `null` if there is none.

```
public Object delegate()
```

See Also
`setDelegate` (page 1648)

didChangeText

Invoked automatically at the end of a series of changes, this method posts a`TextDidChangeNotification` (page 1532) to the default notification center, which also results in the delegate receiving an NSText delegate `textDidChange` (page 1531) message.

```
public void didChangeText()
```

Discussion
Subclasses implementing methods that change their text should invoke this method at the end of those methods. See the "Class Description" (page 1609) for more information.

See Also
`shouldChangeTextInRange` (page 1655)

doCommandBySelector

Attempts to invoke *`aSelector`* or pass the message up the responder chain.

```
public void doCommandBySelector(NSSelector aSelector)
```

Discussion

This method is invoked by an input manager in response to an `interpretKeyEvents` (page 1191) message.

See Also

`interpretKeyEvents` (page 1191) (NSResponder)

dragOperationForDraggingInfo

Returns the type of drag operation that should be performed if the image were released now.

```
public int dragOperationForDraggingInfo(NSDraggingInfo dragInfo, String type)
```

Discussion

`type` is the pasteboard type that will be read from the dragging pasteboard, and *`dragInfo`* is an object the Application Kit creates that holds information about the dragging session. The returned value should be one of the following:

Option	Meaning
`NSDraggingInfo.DragOperationCopy`	The data represented by the image will be copied.
`NSDraggingInfo.DragOperationLink`	The data will be shared.
`NSDraggingInfo.DragOperationGeneric`	The operation will be defined by the destination.
`NSDraggingInfo.DragOperationPrivate`	The operation is negotiated privately between the source and the destination.

If none of the operations is appropriate, this method should return `NSDraggingInfo.DragOperationNone`.

This method is called repeatedly from `draggingEntered` (page 1956) and `draggingUpdated` (page 1957) as the user drags the image.

See Also

`draggingEntered` (page 1956) (NSDraggingDestination)

`draggingUpdated` (page 1957) (NSDraggingDestination)

drawInsertionPointInRect

```
public void drawInsertionPointInRect(NSRect aRect, NSColor aColor, boolean flag)
```

Discussion

If *`flag`* is `true`, draws the insertion point in *`aRect`* using *`aColor`*. If *`flag`* is `false`, this method erases the insertion point. The focus must be locked on the receiver when this method is invoked.

See Also

`insertionPointColor` (page 1630)

`shouldDrawInsertionPoint` (page 1656)

`backgroundColor` (page 1624)

`lockFocus` (page 1759) (NSView)

drawsBackground

Returns `true` if the receiver draws its background, `false` if it doesn't.

```
public boolean drawsBackground()
```

See Also
`backgroundColor` (page 1624)
`setDrawsBackground` (page 1648)

drawViewBackgroundInRect

Called when the text view intends to draw its background

```
public void drawViewBackgroundInRect(NSRect rect)
```

Discussion
. Subclasses can override this method to perform additional drawing behind the text of an NSTextView.

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

firstRectForCharacterRange

```
public NSRect firstRectForCharacterRange(NSRange theRange)
```

Discussion
Returns the first frame of rectangles for *theRange* in the screen coordinate system.

hasMarkedText

Returns `true` if the receiver has text that's still being interpreted by the input manager, `false` if it doesn't.

```
public boolean hasMarkedText()
```

importsGraphics

Returns `true` if the text views sharing the receiver's NSLayoutManager allow the user to import files by dragging, `false` if they don't.

```
public boolean importsGraphics()
```

Discussion
A text view that accepts dragged files is also a rich text view.

See Also
`isRichText` (page 1632)

`textStorage` (page 1659)
`insertAttributedStringAtIndex` (NSMutableAttributedString)
`setImportsGraphics` (page 1649)

insertCompletion

```
public void insertCompletion(String word, NSRange charRange, int movement, boolean
     flag)
```

Discussion

Called with *`flag`* set to `false` as the user moves through the potential text completions, then with *`flag`* set to `true` when a completion is definitively selected or cancelled and the original value is reinserted.

The default implementation inserts the selected completion into the text at the appropriate location. *`movement`* takes its values from the movement codes defined in the NSText "Constants" (page 1530) section, and allows subclassers to distinguish between cancelling completion and selection by arrow keys, by return, by tab, or by other means such as clicking.

Availability

Available in Mac OS X v10.3 and later.

insertionPointColor

Returns the color used to draw the insertion point.

```
public NSColor insertionPointColor()
```

See Also

`drawInsertionPointInRect` (page 1628)
`shouldDrawInsertionPoint` (page 1656)
`setInsertionPointColor` (page 1649)

insertText

Inserts *`aString`* into the receiver's text at the insertion point if there is one, otherwise replacing the selection.

```
public void insertText(Object aString)
```

Discussion

The inserted text is assigned the current typing attributes.

This method is the means by which text typed by the user enters an NSTextView. See the NSInputManager (page 801) class and "NSTextInput" (page 2025) interface specifications for more information.

This method is the entry point for inserting text typed by the user and is generally not suitable for other purposes. Programmatic modification of the text is best done by operating on the NSTextStorage. Because this method pertains to the actions of the user, the text view must be editable for the insertion to work.

See Also

`typingAttributes` (page 1660)

invalidateTextContainerOrigin

Informs the receiver that it needs to recalculate the origin of its text container, usually because it's been resized or the contents of the text container have changed.

```
public void invalidateTextContainerOrigin()
```

Discussion
This method is invoked automatically; you should never need to invoke it directly.

See Also
`textContainer` (page 1659)
`textContainerOrigin` (page 1659)

isContinuousSpellCheckingEnabled

Returns `true` if the object has continuous spell checking enabled, otherwise `false`.

```
public boolean isContinuousSpellCheckingEnabled()
```

See Also
`setContinuousSpellCheckingEnabled` (page 1647)
`toggleContinuousSpellChecking` (page 1660)

isEditable

Returns `true` if the text views sharing the receiver's NSLayoutManager allow the user to edit text, `false` if they don't.

```
public boolean isEditable()
```

Discussion
If a text view is editable, it's also selectable.

See Also
`isSelectable` (page 1632)
`setEditable` (page 1648)

isFieldEditor

Returns `true` if the text views sharing the receiver's NSLayoutManager interpret Tab, Shift-Tab, and Return (Enter) as cues to end editing and possibly to change the first responder; `false` if they accept them as text input.

```
public boolean isFieldEditor()
```

Discussion
See the NSWindow (page 1795) class specification for more information on field editors. By default, NSTextViews don't behave as field editors.

See Also
`setFieldEditor` (page 1648)

isRichText

Returns `true` if the text views sharing the receiver's NSLayoutManager allow the user to apply attributes to specific ranges of the text, `false` if they don't.

```
public boolean isRichText()
```

See Also

`importsGraphics` (page 1629)
`textStorage` (page 1659)
`setRichText` (page 1650)

isRulerVisible

Returns `true` if the scroll view enclosing the text views sharing the receiver's NSLayoutManager shows its ruler, `false` otherwise.

```
public boolean isRulerVisible()
```

See Also

`usesRuler` (page 1663)
`setRulerVisible` (page 1651)
`toggleRuler` (page 1529) (NSText)

isSelectable

Returns `true` if the text views sharing the receiver's NSLayoutManager allow the user to select text, `false` if they don't.

```
public boolean isSelectable()
```

See Also

`isEditable` (page 1631)
`setSelectable` (page 1651)

layoutManager

Returns the NSLayoutManager that lays out text for the receiver's text container, or `null` if there's no such object (which is the case when a text view isn't linked into a group of text objects).

```
public NSLayoutManager layoutManager()
```

See Also

`textContainer` (page 1659)
`setLayoutManager` (page 1560) (NSTextContainer)
`replaceLayoutManager` (page 1559) (NSTextContainer)

linkTextAttributes

Returns the attributes corresponding to the onscreen generation of link text.

```
public native NSDictionary linkTextAttributes()
```

Discussion

In applications created prior to Mac OS X v10.3, the default value is an empty dictionary. In applications created with Mac OS X v10.3 or greater, the default attributes specify blue text with an underline.

Availability

Available in Mac OS X v10.3 and later.

See Also

`setLinkTextAttributes` (page 1649)

loosenKerning

This action method increases the space between glyphs in the receiver's selection, or in all text if the receiver is a plain text view.

```
public void loosenKerning(Object sender)
```

Discussion

Kerning values are determined by the point size of the fonts in the selection.

See Also

`tightenKerning` (page 1659)

`turnOffKerning` (page 1660)

`useStandardKerning` (page 1663)

lowerBaseline

This action method lowers the baseline offset of selected text by 1 point, or of all text if the receiver is a plain text view.

```
public void lowerBaseline(Object sender)
```

Discussion

As such, this method defines a more primitive operation than subscripting.

See Also

`raiseBaseline` (page 1636)

`subscript` (page 1528) (NSText)

`unscript` (page 1529) (NSText)

markedRange

Returns the range of marked text.

```
public NSRange markedRange()
```

Discussion

If there's no marked text, returns a range whose location is `NSArray.NotFound`.

See Also

`setMarkedTextAttributes` (page 1650)

markedTextAttributes

Returns the attributes used to draw marked text.

```
public NSDictionary markedTextAttributes()
```

See Also

`setMarkedTextAttributes` (page 1650)

orderFrontLinkPanel

Brings forward a panel allowing the user to manipulate links in the text view.

```
public void orderFrontLinkPanel(Object sender)
```

Availability

Available in Mac OS X v10.4 and later.

orderFrontListPanel

Brings forward a panel allowing the user to manipulate text lists in the text view.

```
public void orderFrontListPanel(Object sender)
```

Availability

Available in Mac OS X v10.4 and later.

orderFrontSpacingPanel

Brings forward a panel allowing the user to manipulate text line heights, interline spacing, and paragraph spacing, in the text view.

```
public void orderFrontSpacingPanel(Object sender)
```

Availability

Available in Mac OS X v10.4 and later.

orderFrontTablePanel

Brings forward a panel allowing the user to manipulate text tables in the text view.

```
public void orderFrontTablePanel(Object sender)
```

Availability

Available in Mac OS X v10.4 and later.

outline

Adds the outline attribute to the selected text attributes if absent; removes the attribute if not.

```
public void outline(Object sender)
```

Discussion

Uses NSStrokeWidthAttributeName with a default value of 3.0.

Availability

Available in Mac OS X v10.3 and later.

pasteAsPlainText

This action method inserts the contents of the pasteboard into the receiver's text as plain text, in the manner of insertText (page 1630).

```
public void pasteAsPlainText(Object sender)
```

See Also

pasteAsRichText (page 1635)
insertText (page 1630)

pasteAsRichText

This action method inserts the contents of the pasteboard into the receiver's text as rich text, maintaining its attributes.

```
public void pasteAsRichText(Object sender)
```

Discussion

The text is inserted at the insertion point if there is one, otherwise replacing the selection.

See Also

pasteAsRichText (page 1635)
insertText (page 1630)

performFindPanelAction

This is the generic action method for the find menu and find panel, and can be overridden to provide a custom find panel

```
public void performFindPanelAction(Object sender)
```

Discussion

. The actual operation is determined by the tag of the sender, corresponding to the list of tags in "Constants" (page 1665).

Availability

Available in Mac OS X v10.3 and later.

preferredPasteboardTypeFromArray

Returns whatever type on the pasteboard would be most preferred for copying data.

```
public String preferredPasteboardTypeFromArray(NSArray availableTypes, NSArray
    allowedTypes)
```

Discussion

The *availableTypes* parameter lists the types that are currently available on the pasteboard. If the *allowedTypes* parameter is not `null`, then only types in that array may be returned; otherwise, if *allowedTypes* is `null`, any of the available pasteboard types may be returned.

You should not need to override this method. You should also not need to invoke it unless you are implementing a new type of pasteboard to handle services other than copy/paste or dragging.

See Also

`pasteAsPlainText` (page 1635)
`pasteAsRichText` (page 1635)

raiseBaseline

This action method raises the baseline offset of selected text by 1 point, or of all text if the receiver is a plain text view.

```
public void raiseBaseline(Object sender)
```

Discussion

As such, this method defines a more primitive operation than superscripting.

See Also

`lowerBaseline` (page 1633)
`superscript` (page 1528) (NSText)
`unscript` (page 1529) (NSText)

rangeForUserCharacterAttributeChange

Returns the range of characters affected by an action method that changes character (not paragraph) attributes, such as the NSText action method `changeFont` (page 1513).

```
public NSRange rangeForUserCharacterAttributeChange()
```

Discussion

For rich text this range is typically the range of the selection. For plain text this range is the entire contents of the receiver. In Mac OS X v10.4 and later, returns the first subrange where there is a multiple-range selection.

If the receiver isn't editable or doesn't use the Font panel, the range returned has a location of `NSArray.NotFound`.

See Also

rangesForUserCharacterAttributeChange (page 1638)

rangeForUserParagraphAttributeChange (page 1637)

rangeForUserTextChange (page 1638)

isEditable (page 1631)

usesFontPanel (page 1663)

rangeForUserCompletion

Returns the partial range from the most recent beginning of a word up to the insertion point.

```
public NSRange rangeForUserCompletion()
```

Discussion

May be overridden by subclassers to alter the range to be completed. Returning (NSRange.NotFound, 0) suppresses completion.

The resulting value from this method is intended to be used for the range argument in the text completion methods.

In Mac OS X version 10.4 and later, if there are multiple selections, this method acts on the first selected subrange.

Availability

Available in Mac OS X v10.3 and later.

rangeForUserParagraphAttributeChange

Returns the range of characters affected by a method that changes paragraph (not character) attributes, such as the NSText action method alignLeft (page 1512).

```
public NSRange rangeForUserParagraphAttributeChange()
```

Discussion

For rich text this range is typically calculated by extending the range of the selection to paragraph boundaries. For plain text this range is the entire contents of the receiver.

If the receiver isn’t editable, the range returned has a location of NSArray.NotFound.

In Mac OS X version 10.4 and later, if there are multiple selections, this method acts on the first selected subrange.

See Also

rangesForUserParagraphAttributeChange (page 1638)

rangeForUserCharacterAttributeChange (page 1636)

rangeForUserTextChange (page 1638)

isEditable (page 1631)

usesRuler (page 1663)

rangeForUserTextChange

Returns the range of characters affected by a method that changes characters (as opposed to attributes), such as insertText (page 1630).

```
public NSRange rangeForUserTextChange()
```

Discussion
This is typically the range of the selection.

If the receiver isn't editable or doesn't use a ruler, the range returned has a location of NSArray.NotFound.

In Mac OS X version 10.4 and later, if there are multiple selections, this method acts on the first selected subrange.

See Also
rangesForUserTextChange (page 1639)
rangeForUserParagraphAttributeChange (page 1637)
rangeForUserCharacterAttributeChange (page 1636)
isEditable (page 1631)
usesRuler (page 1663)

rangesForUserCharacterAttributeChange

Returns an array containing the ranges of characters affected by an action method that changes character (not paragraph) attributes, such as the NSText action method changeFont (page 1513).

```
public NSArray rangesForUserCharacterAttributeChange()
```

Discussion
For rich text these ranges are typically the ranges of the selections. For plain text the range is the entire contents of the receiver.

Returns null if the receiver isn't editable or doesn't use the Font panel.

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

See Also
rangesForUserParagraphAttributeChange (page 1638)
rangesForUserTextChange (page 1639)
isEditable (page 1631)
usesFontPanel (page 1663)

rangesForUserParagraphAttributeChange

Returns an array containing the ranges of characters affected by a method that changes paragraph (not character) attributes, such as the NSText action method alignLeft (page 1512).

```
public NSArray rangesForUserParagraphAttributeChange()
```

Discussion

For rich text these ranges are typically calculated by extending the range of the current selections to paragraph boundaries. For plain text the range is the entire contents of the receiver.

Returns null if the receiver isn't editable or doesn't use the Font panel.

Availability

Available in Mac OS X v10.4 and later.

See Also

`rangesForUserCharacterAttributeChange` (page 1638)
`rangesForUserTextChange` (page 1639)
`isEditable` (page 1631)
`usesRuler` (page 1663)

rangesForUserTextChange

Returns an array containing the ranges of characters affected by a method that changes characters (as opposed to attributes), such as `insertText` (page 1630).

```
public NSArray rangesForUserTextChange()
```

Discussion

These are typically the ranges of the selections.

Returns null if the receiver isn't editable or doesn't use a ruler.

Availability

Available in Mac OS X v10.4 and later.

See Also

`rangesForUserCharacterAttributeChange` (page 1638)
`rangesForUserParagraphAttributeChange` (page 1638)
`isEditable` (page 1631)
`usesRuler` (page 1663)

readablePasteboardTypes

Returns an array of strings describing the types this text view can read immediately from the pasteboard.

```
public NSArray readablePasteboardTypes()
```

Discussion

The strings are ordered by the default preferences.

You can override this method to provide support for new types of data. If you want to add support for the default types, you can invoke the superclass version of this method or add the types directly in your overridden version.

See Also

`preferredPasteboardTypeFromArray` (page 1636)
`writablePasteboardTypes` (page 1664)

readSelectionFromPasteboard

Reads the text view's preferred type of data from the pasteboard specified by the *pboard* parameter.

```
public boolean readSelectionFromPasteboard(NSPasteboard pboard)
```

Discussion

This method invokes the `preferredPasteboardTypeFromArray` (page 1636) method to determine the text view's preferred type of data and then reads the data using the `readSelectionFromPasteboardOfType` (page 1640) method. Returns `true` if the data was successfully read.

You should not need to override this method. You might need to invoke this method if you are implementing a new type of pasteboard to handle services other than copy/paste or dragging.

See Also

`preferredPasteboardTypeFromArray` (page 1636)

`readSelectionFromPasteboardOfType` (page 1640)

readSelectionFromPasteboardOfType

Reads data of the given *type* from *pboard*.

```
public boolean readSelectionFromPasteboardOfType(NSPasteboard pboard, String type)
```

Discussion

The new data is placed at the current insertion point, replacing the current selection if one exists. Returns `true` if the data was successfully read.

You should override this method to read pasteboard types other than the default types. Use the `rangeForUserTextChange` (page 1638) method to obtain the range of characters (if any) to be replaced by the new data.

See Also

`rangeForUserTextChange` (page 1638)

replaceTextContainer

Replaces the NSTextContainer for the group of text system objects containing the receiver with *aTextContainer*, keeping the association between the receiver and its layout manager intact, unlike `setTextContainer` (page 1653).

```
public void replaceTextContainer(NSTextContainer aTextContainer)
```

Discussion

Throws `InvalidArgumentException` if *aTextContainer* is `null`.

See Also

`setTextContainer` (page 1653)

resignFirstResponder

Notifies the receiver that it's been asked to relinquish its status as first responder in its NSWindow.

```
public boolean resignFirstResponder()
```

Discussion

If the object that will become the new first responder is an NSTextView attached to the same NSLayoutManager as the receiver, this method returns `true` with no further action. Otherwise, this method sends a `textShouldEndEditing` (page 1532) message to its delegate (if any). If the delegate returns `false`, this method returns `false`. If the delegate returns `true`, this method hides the selection highlighting and posts a `TextDidEndEditingNotification` (page 1533) to the default notification center.

Use NSWindow's `makeFirstResponder` (page 1841), not this method, to make an NSTextView the first responder. Never invoke this method directly.

See Also

`breakUndoCoalescing` (page 1625)

rulerViewDidAddMarker

```
public void rulerViewDidAddMarker(NSRulerView aRulerView, NSRulerMarker aMarker)
```

Discussion

This NSRulerView client method modifies the paragraph style of the paragraphs containing the selection to accommodate a new NSTextTab represented by *aMarker*. It then records the change by invoking `didChangeText` (page 1627).

NSTextView checks for permission to make the change in its `rulerViewShouldAddMarker` (page 1642) method, which invokes `shouldChangeTextInRange` (page 1655) to send out the proper request and notifications, and only invokes this method if permission is granted.

See Also

`representedObject` (page 1205) (NSRulerMarker)

`rulerViewDidMoveMarker` (page 1641)

`rulerViewDidRemoveMarker` (page 1642)

rulerViewDidMoveMarker

```
public void rulerViewDidMoveMarker(NSRulerView aRulerView, NSRulerMarker aMarker)
```

Discussion

This NSRulerView client method modifies the paragraph style of the paragraphs containing the selection to record the new location of the NSTextTab represented by *aMarker*. It then records the change by invoking `didChangeText` (page 1627).

NSTextView checks for permission to make the change in its `rulerViewShouldMoveMarker` (page 1643) method, which invokes `shouldChangeTextInRange` (page 1655) to send out the proper request and notifications, and only invokes this method if permission is granted.

See Also

`representedObject` (page 1205) (NSRulerMarker) `rulerViewDidAddMarker` (page 1641)

`rulerViewDidRemoveMarker` (page 1642)

rulerViewDidRemoveMarker

```
public void rulerViewDidRemoveMarker(NSRulerView aRulerView, NSRulerMarker aMarker)
```

Discussion

This NSRulerView client method modifies the paragraph style of the paragraphs containing the selection—if possible—by removing the NSTextTab represented by *aMarker*. It then records the change by invoking `didChangeText` (page 1627).

NSTextView checks for permission to move or remove a tab stop in its `rulerViewShouldMoveMarker` (page 1643) method, which invokes `shouldChangeTextInRange` (page 1655) to send out the proper request and notifications, and only invokes this method if permission is granted.

See Also

`representedObject` (page 1205) (NSRulerMarker)

`shouldChangeTextInRange` (page 1655)

`rulerViewDidAddMarker` (page 1641)

`rulerViewDidMoveMarker` (page 1641)

rulerViewHandleMouseDown

```
public void rulerViewHandleMouseDown(NSRulerView aRulerView, NSEvent theEvent)
```

Discussion

This NSRulerView client method adds a left tab marker to the ruler, but a subclass can override this method to provide other behavior, such as creating guidelines. This method is invoked once with *theEvent* when the user first clicks the ruler area of *aRulerView*, as described in the NSRulerView class specification.

rulerViewShouldAddMarker

```
public boolean rulerViewShouldAddMarker(NSRulerView aRulerView, NSRulerMarker
    aMarker)
```

Discussion

This NSRulerView client method controls whether a new tab stop *aMarker* can be added. The receiver checks for permission to make the change by invoking `shouldChangeTextInRange` (page 1655) and returning the return value of that message. If the change is allowed, the receiver is then sent a `rulerViewDidAddMarker` (page 1641) message.

See Also

`rulerViewShouldMoveMarker` (page 1643)

`rulerViewShouldRemoveMarker` (page 1643)

rulerViewShouldMoveMarker

```
public boolean rulerViewShouldMoveMarker(NSRulerView aRulerView, NSRulerMarker
    aMarker)
```

Discussion

This NSRulerView client method controls whether an existing tab stop `aMarker` can be moved. The receiver checks for permission to make the change by invoking `shouldChangeTextInRange` (page 1655) and returning the return value of that message. If the change is allowed, the receiver is then sent a `rulerViewDidAddMarker` (page 1641) message.

See Also

`rulerViewShouldAddMarker` (page 1642)

`rulerViewShouldRemoveMarker` (page 1643)

rulerViewShouldRemoveMarker

```
public boolean rulerViewShouldRemoveMarker(NSRulerView aRulerView, NSRulerMarker
    aMarker)
```

Discussion

This NSRulerView client method controls whether an existing tab stop `aMarker` can be removed. Returns `true` if `aMarker` represents an NSTextTab, `false` otherwise. Because this method can be invoked repeatedly as the user drags a ruler marker, it returns that value immediately. If the change is allows and the user actually removes the marker, the receiver is also sent a `rulerViewDidRemoveMarker` (page 1642) message.

See Also

`rulerViewShouldAddMarker` (page 1642)

`rulerViewShouldMoveMarker` (page 1643)

rulerViewWillAddMarker

```
public float rulerViewWillAddMarker(NSRulerView aRulerView, NSRulerMarker aMarker,
     float location)
```

Discussion

This NSRulerView client method ensures that the proposed `location` of `aMarker` lies within the appropriate bounds for the receiver's text container, returning the modified location.

See Also

`rulerViewDidAddMarker` (page 1641)

rulerViewWillMoveMarker

```
public float rulerViewWillMoveMarker(NSRulerView aRulerView, NSRulerMarker aMarker,
     float location)
```

Discussion

This NSRulerView client method ensures that the proposed `location` of `aMarker` lies within the appropriate bounds for the receiver's text container, returning the modified location.

See Also

rulerViewDidMoveMarker (page 1641)

selectedRange

Returns the range of characters selected in the receiver’s layout manager.

```
public NSRange selectedRange()
```

See Also

selectedTextAttributes (page 1644)

selectionRangeForProposedRange (page 1645)

setSelectedRange (page 1651)

selectedRanges

Returns an array containing the ranges of characters selected in the receiver’s layout manager.

```
public NSArray selectedRanges()
```

Discussion

The return value is a non-null, non-empty array of objects responding to the NSValue rangeValue method, and in addition its elements are sorted, non-overlapping, non-contiguous, and (except for the case of a single range) have non-zero-length.

Availability

Available in Mac OS X v10.4 and later.

See Also

setSelectedRanges (page 1652)

selectedTextAttributes

Returns the attributes used to indicate the selection.

```
public NSDictionary selectedTextAttributes()
```

Discussion

This attribute is typically just the text background color.

See Also

selectedRange (page 1644)

setSelectedTextAttributes (page 1652)

selectionAffinity

Returns the preferred direction of selection, either SelectionAffinityUpstream or SelectionAffinityDownstream.

```
public int selectionAffinity()
```

Discussion

Selection affinity determines whether, for example, the insertion point appears after the last character on a line or before the first character on the following line in cases where text wraps across line boundaries.

selectionGranularity

Returns the current selection granularity, used during mouse tracking to modify the range of the selection.

```
public int selectionGranularity()
```

Discussion

This is one of:

```
SelectByCharacter
SelectByWord
SelectByParagraph
```

See Also

`selectionRangeForProposedRange` (page 1645)
`setSelectionGranularity` (page 1653)

selectionRangeForProposedRange

Adjusts the *proposedSelRange* if necessary, based on *granularity*.

```
public NSRange selectionRangeForProposedRange(NSRange proposedSelRange, int
    granularity)
```

Discussion

granularity is one of:

```
SelectByCharacter
SelectByWord
SelectByParagraph
```

Returns the adjusted range. This method is invoked repeatedly during mouse tracking to modify the range of the selection. Override this method to specialize selection behavior.

See Also

`setSelectionGranularity` (page 1653)

setAlignmentInRange

Sets the alignment of the paragraphs containing characters in *aRange* to *alignment*.

```
public void setAlignmentInRange(int alignment, NSRange aRange)
```

Discussion

alignment is one of:

```
NSText.LeftTextAlignment
NSText.RightTextAlignment
NSText.CenterTextAlignment
NSText.JustifiedTextAlignment
NSText.NaturalTextAlignment
```

This method does not include undo support by default. Clients must invoke `shouldChangeTextInRanges` (page 1656) or `shouldChangeTextInRange` (page 1655) to include this method in an undoable action.

See Also
`rangeForUserParagraphAttributeChange` (page 1637)

setAllowsDocumentBackgroundColorChange

Sets whether or not the receiver allows its background color to change.

```
public void setAllowsDocumentBackgroundColorChange(boolean flag)
```

Discussion
`flag` should be set to `true` if the receiver allows the change, otherwise `false`. This corresponds to the background color of the entirety of the text view, not just to a selected range of text.

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

See Also
`allowsDocumentBackgroundColorChange` (page 1624)
`changeDocumentBackgroundColor` (page 1626)

setAllowsUndo

If *`flag`* is `true` enables undo support; otherwise disables it.

```
public void setAllowsUndo(boolean flag)
```

See Also
`allowsUndo` (page 1624)

setBackgroundColor

Sets the receiver's background color to *`aColor`*.

```
public void setBackgroundColor(NSColor aColor)
```

Discussion
This method does not include undo support by default. Clients must invoke `shouldChangeTextInRanges` (page 1656) or `shouldChangeTextInRange` (page 1655) to include this method in an undoable action.

See Also
`setDrawsBackground` (page 1648)

backgroundColor (page 1624)

setBaseWritingDirection

Sets the base writing direction of the text in *range* to *writingDirection*.

```
public void setBaseWritingDirection(int writingDirection, NSRange range)
```

Discussion
Invoke this method to change the base writing direction from left-to-right to right-to-left for languages like Hebrew and Arabic, for example.

This method does not include undo support by default. Clients must invoke shouldChangeTextInRanges (page 1656) or shouldChangeTextInRange (page 1655) to include this method in an undoable action.

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

setConstrainedFrameSize

Attempts to set the frame size for the receiver to *desiredSize*, constrained by the receiver's existing minimum and maximum sizes and by whether resizing is permitted.

```
public void setConstrainedFrameSize(NSSize desiredSize)
```

See Also
minSize (page 1518) (NSText)
maxSize (page 1518) (NSText)
isHorizontallyResizable (page 1517) (NSText)
isVerticallyResizable (page 1518) (NSText)

setContinuousSpellCheckingEnabled

If *flag* is `true` enables continuous spell checking; otherwise disables it.

```
public void setContinuousSpellCheckingEnabled(boolean flag)
```

See Also
isContinuousSpellCheckingEnabled (page 1631)
toggleContinuousSpellChecking (page 1660)

setDefaultParagraphStyle

Sets the receiver's default paragraph style.

```
public void setDefaultParagraphStyle(NSParagraphStyle paragraphStyle)
```

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

See Also

`defaultParagraphStyle` (page 1627)

setDelegate

Sets the delegate for all NSTextViews sharing the receiver's NSLayoutManager to *anObject*, without retaining it.

```
public void setDelegate(Object anObject)
```

See Also

`delegate` (page 1627)

setDrawsBackground

Controls whether the receiver draws its background.

```
public void setDrawsBackground(boolean flag)
```

Discussion

If *flag* is `true`, the receiver fills its background with the background color; if *flag* is `false`, it doesn't.

See Also

`setBackgroundColor` (page 1646)

`drawsBackground` (page 1629)

setEditable

Controls whether the text views sharing the receiver's NSLayoutManager allow the user to edit text.

```
public void setEditable(boolean flag)
```

Discussion

If *flag* is `true`, they allow the user to edit text and attributes; if *flag* is `false`, they don't. If an NSTextView is made editable, it's also made selectable. NSTextViews are by default editable.

See Also

`setSelectable` (page 1651)

`isEditable` (page 1631)

setFieldEditor

Controls whether the text views sharing the receiver's NSLayoutManager interpret Tab, Shift-Tab, and Return (Enter) as cues to end editing and possibly to change the first responder.

```
public void setFieldEditor(boolean flag)
```

Discussion

If *flag* is `true`, they do; if *flag* is `false`, they don't, instead accepting these characters as text input. See the NSWindow (page 1795) class specification for more information on field editors. By default, NSTextViews don't behave as field editors.

See Also

`isFieldEditor` (page 1631)

setImportsGraphics

Controls whether the text views sharing the receiver's NSLayoutManager allow the user to import files by dragging.

```
public void setImportsGraphics(boolean flag)
```

Discussion

If *flag* is `true`, they do; if *flag* is `false`, they don't. If an NSTextView is set to accept dragged files, it's also set for rich text. By default, NSTextViews don't accept dragged files.

See Also

`textStorage` (page 1659)

`setRichText` (page 1650)

`importsGraphics` (page 1629)

setInsertionPointColor

Sets the color of the insertion point to *aColor*.

```
public void setInsertionPointColor(NSColor aColor)
```

See Also

`drawInsertionPointInRect` (page 1628)

`shouldDrawInsertionPoint` (page 1656)

`insertionPointColor` (page 1630)

setLinkTextAttributes

Sets the attributes corresponding to the onscreen generation of link text.

```
public void setLinkTextAttributes(NSDictionary attributeDictionary)
```

Availability

Available in Mac OS X v10.3 and later.

See Also

`linkTextAttributes` (page 1633)

setMarkedTextAndSelectedRange

Replaces text in *aRange* within receiver's text storage with the contents of *aString*, which the receiver must display distinctively to indicate that it is marked text.

```
public void setMarkedTextAndSelectedRange(Object aString, NSRange aRange)
```

Discussion
aString must be either a String or an NSAttributedString, and not `null`.

See Also
`selectedRange` (page 1644)
`unmarkText` (page 1661)

setMarkedTextAttributes

Sets the attributes used to draw marked text to *attributes*.

```
public void setMarkedTextAttributes(NSDictionary attributes)
```

Discussion
Text color, background color, and underline are the only supported attributes for marked text.

See Also
`markedTextAttributes` (page 1634)
`markedRange` (page 1633)

setNeedsDisplay

Marks the receiver as requiring display within *aRect*.

```
public void setNeedsDisplay(NSRect aRect, boolean flag)
```

Discussion
If *flag* is `true`, the receiver won't perform any layout that might be required to complete the display, even if this means that portions of the NSTextView remain empty. If *flag* is `false`, the receiver performs at least as much layout as needed to display *aRect*.

NSTextView overrides the NSView `setNeedsDisplay` (page 1779) method such that it invokes this method with a *flag* argument of `false`.

setRichText

Controls whether the text views sharing the receiver's NSLayoutManager allow the user to apply attributes to specific ranges of the text.

```
public void setRichText(boolean flag)
```

Discussion
If *flag* is `true` they do; if *flag* is `false` they don't. If *flag* is `false`, they're also set not to accept dragged files. By default, NSTextViews let the user apply multiple attributes to text, but don't accept dragged files.

See Also

`textStorage` (page 1659)

`isRichText` (page 1632)

`setImportsGraphics` (page 1649)

setRulerVisible

Controls whether the scroll view enclosing text views sharing the receiver's NSLayoutManager displays the ruler.

```
public void setRulerVisible(boolean flag)
```

Discussion

If *flag* is `true` it shows the ruler; if *flag* is `false` it hides the ruler. By default, the ruler is not visible.

See Also

`setUsesRuler` (page 1655)

`isRulerVisible` (page 1632)

`toggleRuler` (page 1529) (NSText)

setSelectable

Controls whether the text views sharing the receiver's NSLayoutManager allow the user to select text.

```
public void setSelectable(boolean flag)
```

Discussion

If *flag* is `true`, users are allowed to select text; if *flag* is `false`, users are not allowed to select text. If an NSTextView is made not selectable, it's also made not editable. NSTextViews are by default both editable and selectable.

See Also

`setEditable` (page 1648)

`isSelectable` (page 1632)

setSelectedRange

```
public void setSelectedRange(NSRange charRange, int affinity, boolean flag)
```

Discussion

Sets the selection to the characters in *charRange*, using *affinity* if needed to determine how to display the selection or insertion point (see the description for `selectionAffinity` (page 1644) for more information). *flag* indicates whether this method is being invoked during mouse dragging or after the user releases the mouse button. If *flag* is `true` the receiver doesn't send notifications or remove the marking from its marked text; if *flag* is `false` it does as appropriate. This method also resets the selection granularity to `SelectByCharacter`.

The *charRange* argument must begin and end on glyph boundaries and not split base glyphs and their nonspacing marks.

```
public void setSelectedRange(NSRange charRange)
```

Discussion

Sets the selection to the characters in *charRange*, resets the selection granularity to `SelectByCharacter`, and posts an `TextViewDidChangeSelectionNotification` (page 1672) to the default notification center. Also removes the marking from marked text if the new selection is greater than the marked region.

The *charRange* argument must begin and end on glyph boundaries and not split base glyphs and their nonspacing marks.

See Also

`selectionAffinity` (page 1644)

`selectionGranularity` (page 1645)

`selectedRange` (page 1644)

setSelectedRanges

```
public void setSelectedRanges(NSArray ranges)
```

Discussion

Sets the selection to the characters in the *ranges* array, resets the selection granularity to `NSSelectByCharacter`, and posts an `TextViewDidChangeSelectionNotification` (page 1672) to the default notification center. Also removes the marking from marked text if the new selection is greater than the marked region.

The *ranges* argument must be a non-nil, non-empty array of objects responding to the NSValue `rangeValue` method. The ranges in the *ranges* array must begin and end on glyph boundaries and not split base glyphs and their nonspacing marks.

```
public void setSelectedRanges(NSArray ranges, int affinity, boolean
    stillSelectingFlag)
```

Discussion

The method variant with three arguments uses *affinity* if needed to determine how to display the selection or insertion point (see the description for `selectionAffinity` (page 1644) for more information). *stillSelectingFlag* indicates whether this method is being invoked during mouse dragging or after the user releases the mouse button. If *stillSelectingFlag* is `true` the receiver doesn't send notifications or remove the marking from its marked text; if *stillSelectingFlag* is `false` it does as appropriate.

Availability

Available in Mac OS X v10.4 and later.

See Also

`selectedRanges` (page 1644),

`selectionAffinity` (page 1644)

`selectionGranularity` (page 1645)

setSelectedTextAttributes

Sets the attributes used to indicate the selection to *attributes*.

```
public void setSelectedTextAttributes(NSDictionary attributes)
```

Discussion

Text color, background color, and underline are the only supported attributes for selected text.

See Also

`selectedRange` (page 1644)

`selectedTextAttributes` (page 1644)

setSelectionGranularity

Sets the selection granularity for subsequent extension of a selection to *granularity*.

```
public void setSelectionGranularity(int granularity)
```

Discussion

granularity may be one of:

```
SelectByCharacter
SelectByWord
SelectByParagraph
```

Selection granularity is used to determine how the selection is modified when the user Shift-clicks or drags the mouse after a double or triple click. For example, if the user selects a word by double-clicking, the selection granularity is set to `SelectByWord`. Subsequent Shift-clicks then extend the selection by words.

Selection granularity is reset to `SelectByCharacter` whenever the selection is set. You should always set the selection granularity after setting the selection.

See Also

`selectionGranularity` (page 1645)

`setSelectedRange` (page 1651)

setSmartInsertDeleteEnabled

Controls whether the receiver inserts or deletes space around selected words so as to preserve proper spacing and punctuation.

```
public void setSmartInsertDeleteEnabled(boolean flag)
```

Discussion

If *flag* is `true` it does; if *flag* is `false` it inserts and deletes exactly what's selected.

See Also

`smartDeleteRangeForProposedRange` (page 1657)

`smartInsertDeleteEnabled` (page 1658)

setTextContainer

Sets the receiver's text container to *aTextContainer*.

```
public void setTextContainer(NSTextContainer aTextContainer)
```

Discussion

The receiver then uses the layout manager and text storage of *aTextContainer*. This method is invoked automatically when you create an NSTextView; you should never invoke it directly, but might want to override it. To change the text view for an established group of text system objects, send `setTextView` (page 1561) to the text container. To replace the text container for a text view and maintain the view's association with the existing layout manager and text storage, use `replaceTextContainer` (page 1640).

See Also

`textContainer` (page 1659)

setTextContainerInset

Sets the empty space the receiver leaves around its associated text container to *inset*.

```
public void setTextContainerInset(NSSize inset)
```

Discussion

It is possible to set the text container and view sizes and resizing behavior so that the inset cannot be maintained exactly, although the text system tries to maintain the inset wherever possible. In any case, the `textContainerOrigin` (page 1659) and size of the text container are authoritative as to the location of the text container within the view.

The text itself can have an additional inset, inside the text container, specified by the `setLineFragmentPadding` (page 1560) method of NSTextContainer.

See Also

`textContainerOrigin` (page 1659)
`invalidateTextContainerOrigin` (page 1631)
`textContainerInset` (page 1659)

setTypingAttributes

Sets the receiver's typing attributes to *attributes*.

```
public void setTypingAttributes(NSDictionary attributes)
```

Discussion

Typing attributes are reset automatically whenever the selection changes. If you add any user actions that change text attributes, you should use this method to apply those attributes to a zero-length selection.

See Also

`typingAttributes` (page 1660)

setUsesFindPanel

Specifies whether the receiver allows for a find panel.

```
public void setUsesFindPanel(boolean flag)
```

Availability

Available in Mac OS X v10.3 and later.

See Also

`usesFindPanel` (page 1662)

setUsesFontPanel

Controls whether the text views sharing the receiver's NSLayoutManager use the Font panel and Font menu.

```
public void setUsesFontPanel(boolean flag)
```

Discussion

If *flag* is `true`, they respond to messages from the Font panel and from the Font menu, and update the Font panel with the selection font whenever it changes. If *flag* is `false` they disallow character attribute changes. By default, NSTextView objects use the Font panel and menu.

See Also

`rangeForUserCharacterAttributeChange` (page 1636)

`usesFontPanel` (page 1663)

setUsesRuler

Controls whether the text views sharing the receiver's NSLayoutManager use an NSRulerView and respond to Format menu commands.

```
public void setUsesRuler(boolean flag)
```

Discussion

If *flag* is `true`, they respond to NSRulerView client messages and to paragraph-related menu actions, and update the ruler (when visible) as the selection changes with its paragraph and tab attributes. If *flag* is `false`, the ruler is hidden, and the text views disallow paragraph attribute changes. By default, NSTextView objects use the ruler.

See Also

`setRulerVisible` (page 1651)

`rangeForUserParagraphAttributeChange` (page 1637)

`usesRuler` (page 1663)

shouldChangeTextInRange

Initiates a series of delegate messages (and general notifications) to determine whether modifications can be made to the receiver's text.

```
public boolean shouldChangeTextInRange(NSRange affectedCharRange, String
    replacementString)
```

Discussion

If characters in the text string are being changed, *replacementString* contains the characters that will replace the characters in *affectedCharRange*. If only text attributes are being changed, *replacementString* is `null`. This method checks with the delegate as needed using `textShouldBeginEditing` (page 1532) and `textViewShouldChangeTextInRange` (page 1670), returning `true` to allow the change, and `false` to prohibit it.

This method must be invoked at the start of any sequence of user-initiated editing changes. If your subclass of NSTextView implements new methods that modify the text, make sure to invoke this method to determine whether the change should be made. If the change is allowed, complete the change by invoking the `didChangeText` (page 1627) method. If you can't determine the affected range or replacement string before beginning changes, pass (`NSArray.NotFound`, 0) and `null` for these values.

If the receiver is not editable, this method automatically returns `false`. This result prevents instances in which a text view could be changed by user actions even though it had been set to be noneditable. In Mac OS X version 10.4 and later, if there are multiple selections, this method acts on the first selected subrange.

See Also

`isEditable` (page 1631)

`shouldChangeTextInRanges` (page 1656)

shouldChangeTextInRanges

Initiates a series of delegate messages (and general notifications) to determine whether modifications can be made to the receiver's text.

```
public boolean shouldChangeTextInRanges(NSArray affectedRanges, NSArray
    replacementStrings)
```

Discussion

The *replacementStrings* array should either be null or else contain one element for each range in *affectedRanges*. If characters in the text string are being changed, the *replacementStrings* array contains the characters that replace the characters in *affectedRanges*. If only text attributes are being changed, *replacementStrings* is `null`. This method checks with the delegate as needed using `textShouldBeginEditing` (page 1532) and `textViewShouldChangeTextInRanges` (page 1670), returning `true` to allow the change, and `false` to prohibit it.

This method must be invoked at the start of any sequence of user-initiated editing changes. If your subclass of NSTextView implements new methods that modify the text, make sure to invoke this method to determine whether the change should be made. If the change is allowed, complete the change by invoking the `didChangeText` (page 1627) method. If you can't determine the affected range or replacement string before beginning changes, pass `null` for these values.

If the receiver is not editable, this method automatically returns `false`. This result prevents instances in which a text view could be changed by user actions even though it had been set to be noneditable.

Availability

Available in Mac OS X v10.4 and later.

See Also

`isEditable` (page 1631)

shouldDrawInsertionPoint

Returns `true` if the receiver should draw its insertion point, `false` if the insertion point can't or shouldn't be drawn.

```
public boolean shouldDrawInsertionPoint()
```

Discussion
For example, you might not want to draw the insertion point if the receiver's window isn't key.)

See Also
`drawInsertionPointInRect` (page 1628)

smartDeleteRangeForProposedRange

Given *proposedCharRange*, returns an extended range that includes adjacent whitespace that should be deleted along with the proposed range in order to preserve proper spacing and punctuation of the text surrounding the deletion.

```
public NSRange smartDeleteRangeForProposedRange(NSRange proposedCharRange)
```

Discussion
NSTextView uses this method as necessary; you can also use it in implementing your own methods that delete text, typically when the selection granularity is `SelectByWord`. To do so, invoke this method with the proposed range to delete, then actually delete the range returned. If placing text on the pasteboard, however, you should put only the characters from the proposed range onto the pasteboard.

See Also
`selectionGranularity` (page 1645)
`smartInsertDeleteEnabled` (page 1658)

smartInsertAfterStringForString

Returns any whitespace that needs to be added after *aString* to preserve proper spacing and punctuation when *aString* is inserted into the receiver's text over *charRange*.

```
public String smartInsertAfterStringForString(String aString, NSRange charRange)
```

Discussion
If *aString* is `null` or if smart insertion and deletion are disabled, this method returns `null`.

When inserting text, insert the following over *charRange*:

- The results of `smartInsertBeforeStringForString` (page 1657)
- *aString*
- The results of this method

smartInsertBeforeStringForString

Returns any whitespace that needs to be added before *aString* to preserve proper spacing and punctuation when *aString* is inserted into the receiver's text over *charRange*.

```
public String smartInsertBeforeStringForString(String aString, NSRange charRange)
```

Discussion
If *aString* is `null` or if smart insertion and deletion are disabled, this method returns `null`.

When inserting text, insert the following over *charRange*:

- The results of this method
- *aString*
- The results of smartInsertAfterStringForString (page 1657)

smartInsertDeleteEnabled

Returns true if the receiver inserts or deletes space around selected words so as to preserve proper spacing and punctuation, false if it inserts and deletes exactly what's selected.

```
public boolean smartInsertDeleteEnabled()
```

See Also
smartDeleteRangeForProposedRange (page 1657)
setSmartInsertDeleteEnabled (page 1653)

spellCheckerDocumentTag

Returns a tag identifying the NSTextView text as a document for the spell checker server.

```
public int spellCheckerDocumentTag()
```

Discussion
The document tag is obtained by sending a "uniqueSpellDocumentTag" (page 1381) message to the spell server the first time this method is invoked for a particular group of NSTextViews. See the NSSpellChecker (page 1379) and NSSpellServer class specifications for more information on how this tag is used.

startSpeaking

This action method speaks the selected text, or all text if no selection.

```
public void startSpeaking(Object sender)
```

See Also
stopSpeaking (page 1658)

stopSpeaking

This action method stops the speaking of text.

```
public void stopSpeaking(Object sender)
```

See Also
startSpeaking (page 1658)

textContainer

Returns the receiver's text container.

```
public NSTextContainer textContainer()
```

See Also

setTextContainer (page 1653)

textContainerInset

Returns the empty space the receiver leaves around its text container.

```
public NSSize textContainerInset()
```

See Also

textContainerOrigin (page 1659)

invalidateTextContainerOrigin (page 1631)

setTextContainerInset (page 1654)

textContainerOrigin

Returns the origin of the receiver's text container, which is calculated from the receiver's bounds rectangle, container inset, and the container's used rect.

```
public NSPoint textContainerOrigin()
```

See Also

invalidateTextContainerOrigin (page 1631)

textContainerInset (page 1659)

usedRectForTextContainer (page 855) (NSLayoutManager)

textStorage

Returns the receiver's text storage object.

```
public NSTextStorage textStorage()
```

tightenKerning

This action method decreases the space between glyphs in the receiver's selection, or for all glyphs if the receiver is a plain text view.

```
public void tightenKerning(Object sender)
```

Discussion

Kerning values are determined by the point size of the fonts in the selection.

See Also

loosenKerning (page 1633)

`useStandardKerning` (page 1663)
`turnOffKerning` (page 1660)

toggleContinuousSpellChecking

This action method toggles whether continuous spell checking is enabled for the receiver.

```
public void toggleContinuousSpellChecking(Object sender)
```

See Also
`isContinuousSpellCheckingEnabled` (page 1631)
`setContinuousSpellCheckingEnabled` (page 1647)

toggleTraditionalCharacterShape

This action method toggles the `NSCharacterShapeAttibuteName` attribute at the current selection.

```
public void toggleTraditionalCharacterShape(Object sender)
```

turnOffKerning

This action method causes the receiver to use nominal glyph spacing for the glyphs in its selection, or for all glyphs if the receiver is a plain text view.

```
public void turnOffKerning(Object sender)
```

See Also
`useStandardKerning` (page 1663)
`loosenKerning` (page 1633)
`tightenKerning` (page 1659)
`isRichText` (page 1632)

turnOffLigatures

This action method causes the receiver to use only required ligatures when setting text, for the glyphs in the selection if the receiver is a rich text view, or for all glyphs if it's a plain text view.

```
public void turnOffLigatures(Object sender)
```

See Also
`useAllLigatures` (page 1662)
`isRichText` (page 1632)
`useStandardLigatures` (page 1663)

typingAttributes

Returns the current typing attributes.

```
public NSDictionary typingAttributes()
```

See Also
setTypingAttributes (page 1654)

underline

```
public void underline(Object sender)
```

Discussion
Adds the underline attribute to the selected text attributes if absent; removes the attribute if not. Uses NSUnderlineStyleAttribute with a default value of 1.

unmarkText

Removes any marking from pending input text and accepts the text in its current state.

```
public void unmarkText()
```

updateDragTypeRegistration

```
public void updateDragTypeRegistration()
```

Discussion
If the receiver is editable and is a rich text view, causes all NSTextViews associated with the receiver's NSLayoutManager to register their acceptable drag types. If the NSTextView isn't editable or isn't rich text, causes those NSTextViews to unregister their dragged types.

Subclasses can override this method to change the conditions for registering and unregistering drag types, whether as a group or individually based on the current state of the NSTextView. They can then invoke this method when that state changes to perform that reregistration.

See Also
acceptableDragTypes (page 1623)
registerForDraggedTypes (page 1768) (NSView)
unregisterDraggedTypes (page 1784) (NSView)
isEditable (page 1631)
importsGraphics (page 1629)
isRichText (page 1632)

updateFontPanel

Updates the Font panel to contain the font attributes of the selection.

```
public void updateFontPanel()
```

Discussion
Does nothing if the receiver doesn't use the Font panel. You should never need to invoke this method directly, but you can override it if needed to handle additional font attributes.

See Also
`usesFontPanel` (page 1663)

updateInsertionPointStateAndRestartTimer

Updates the insertion point's location and, if *`flag`* is `true`, restarts the blinking cursor timer.

```
public void updateInsertionPointStateAndRestartTimer(boolean flag)
```

Discussion
This method is invoked automatically whenever the insertion point needs to be moved; you should never need to invoke it directly, but you can override it to add different insertion point behavior.

See Also
`shouldDrawInsertionPoint` (page 1656)
`drawInsertionPointInRect` (page 1628)

updateRuler

Updates the NSRulerView in the receiver's enclosing scroll view to reflect the selection's paragraph and marker attributes.

```
public void updateRuler()
```

Discussion
Does nothing if the ruler isn't visible or if the receiver doesn't use the ruler. You should never need to invoke this method directly, but you can override this method if needed to handle additional ruler attributes.

See Also
`usesRuler` (page 1663)

useAllLigatures

This action method causes the receiver to use all ligatures available for the fonts and languages used when setting text, for the glyphs in the selection if the receiver is a rich text view, or for all glyphs if it's a plain text view.

```
public void useAllLigatures(Object sender)
```

See Also
`turnOffLigatures` (page 1660)
`useStandardLigatures` (page 1663)

usesFindPanel

Returns whether the receiver allows for a find panel.

```
public boolean usesFindPanel()
```

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

See Also
`setUsesFindPanel` (page 1654)

usesFontPanel

Returns `true` if the text views sharing the receiver's NSLayoutManager use the Font panel, `false` otherwise.

```
public boolean usesFontPanel()
```

Discussion
See `setUsesFontPanel` (page 1655) and `rangeForUserCharacterAttributeChange` (page 1636) for the effect this method has on an NSTextView's behavior.

usesRuler

Returns `true` if the text views sharing the receiver's NSLayoutManager use a ruler view, `false` otherwise.

```
public boolean usesRuler()
```

Discussion
See `setUsesRuler` (page 1655) and `rangeForUserParagraphAttributeChange` (page 1637) for the effect this has on an NSTextView's behavior

See Also
`setUsesRuler` (page 1655)

useStandardKerning

This action method causes the receiver to use pair kerning data for the glyphs in its selection, or for all glyphs if the receiver is a plain text view.

```
public void useStandardKerning(Object sender)
```

Discussion
This data is taken from a font's AFM file

See Also
`isRichText` (page 1632)
`loosenKerning` (page 1633)
`tightenKerning` (page 1659)
`turnOffKerning` (page 1660)

useStandardLigatures

This action method causes the receiver to use the standard ligatures available for the fonts and languages used when setting text, for the glyphs in the selection if the receiver is a rich text view, or for all glyphs if it's a plain text view.

```
public void useStandardLigatures(Object sender)
```

See Also

`turnOffLigatures` (page 1660)

`useAllLigatures` (page 1662)

validAttributesForMarkedText

Returns an array of NSString names for the attributes supported by the receiver.

```
public NSArray validAttributesForMarkedText()
```

Discussion

The input server may choose to use some of these attributes in the text it inserts or in marked text. Returns an empty array if no attributes are supported. See NSAttributedString for the set of string constants that you could return in the array.

validRequestorForTypes

Returns `this` if *`sendType`* specifies a type of data the text view can put on the pasteboard and *`returnType`* contains a type of data the text view can read from the pasteboard; otherwise returns `null`.

```
public Object validRequestorForTypes(String sendType, String returnType)
```

See Also

`validRequestorForTypes` (page 1200) (NSResponder)

writablePasteboardTypes

```
public NSArray writablePasteboardTypes()
```

Discussion

If the text view contains some selected data, this method returns an array of strings describing the types that can be written to the pasteboard immediately. You can override this method to add new supported types to the array of strings.

See Also

`readablePasteboardTypes` (page 1639)

writeSelectionToPasteboardOfType

Writes the current selection to *`pboard`* using the given *`type`*.

```
public boolean writeSelectionToPasteboardOfType(NSPasteboard pboard, String type)
```

Discussion

The complete set of data types being written to *`pboard`* should be declared before invoking this method. Returns `true` if the data was successfully written.

This method should be invoked only from `writeSelectionToPasteboardOfTypes` (page 1665). You can override this method to add support for writing new types of data to the pasteboard. You should invoke `super`'s implementation of the method to handle any types of data your overridden version does not.

See Also
`readSelectionFromPasteboardOfType` (page 1640)

writeSelectionToPasteboardOfTypes

Writes the current selection to *pboard* under each type in the *types* array.

```
public boolean writeSelectionToPasteboardOfTypes(NSPasteboard pboard, NSArray types)
```

Discussion
This method declares the data types on *pboard* and then invokes `writeSelectionToPasteboardOfType` (page 1664) for each type in the *types* array. Returns `true` if the data for any single type was written successfully.

You should not need to override this method. You might need to invoke this method if you are implementing a new type of pasteboard to handle services other than copy/paste or dragging.

Constants

These constants specify how much the text view extends the selection when the user drags the mouse. They're used by `selectionGranularity` (page 1645), `setSelectionGranularity` (page 1653), and `selectionRangeForProposedRange` (page 1645):

Constant	Description
`SelectByCharacter`	Extends the selection character by character.
`SelectByWord`	Extends the selection word by word.
`SelectByParagraph`	Extends the selection paragraph by paragraph.

These constants specify the preferred direction of selection. They're used by `selectionAffinity` (page 1644) and `setSelectedRange` (page 1651):

Constant	Description
`SelectionAffinityUpstream`	The selection is moving toward the top of the document.
`SelectionAffinityDownstream`	The selection is moving toward the bottom of the document.

These constants define the tags for `performFindPanelAction` (page 1635):

Constant	Description
`FindPanelActionShowFindPanel`	Displays the find panel.

Constant	Description
`FindPanelActionNext`	Finds the next instance of the queried text.
`FindPanelActionPrevious`	Finds the previous instance of the queried text.
`FindPanelActionReplaceAll`	Replaces all query instances within the text view.
`FindPanelActionReplace`	Replaces a single query instance within the text view.
`FindPanelActionReplaceAndFind`	Replaces a single query instance and finds the next.
`FindPanelActionSetFindString`	Sets the query string to the current selection.
`FindPanelActionReplaceAllInSelection`	Replaces all query instances within the selection.
`FindPanelActionSelectAll`	Selects all query instances in the text view. Available in Mac OS X v10.4 and later.
`FindPanelActionSelectAllInSelection`	Selects all query instances within the selection. Available in Mac OS X v10.4 and later.

Delegate Methods

textView

```
public NSArray textView(NSTextView aTextView, NSCellForTextAttachment cell, int
    charIndex)
```

Discussion

Invoked after the user clicks on *cell* at the specified *charIndex* location in *aTextView*. If `textViewDraggedCellAtIndex` (page 1670) is not used, both variations of this method allows *aTextView* to take care of dragging and pasting attachment *cell*, with the delegate responsible only for writing the attachment to the pasteboard. An array of types that can be written to the pasteboard for the given attachment should be returned.

The receiver should attempt to write the given attachment to *pasteboard* with the given *type* and return success or failure.

```
public boolean textView(NSTextView aTextView, NSCellForTextAttachment cell, int
    charIndex, NSPasteboard pasteboard, String type)
```

textView:completionsForPartialWordRange

Allows the delegate to modify the list of completions that will be presented for the partial word at the given range

```
public abstract NSArray textViewCompletionsForPartialWordRange(NSTextView textView,
     NSArray words, NSRange charRange)
```

Discussion
. Returning `null` or a zero-length array suppresses completion.

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

textViewClickedCell

Invoked after the user clicks on *attachmentCell* within *cellFrame* in *aTextView* and the cell wants to track the mouse.

```
public abstract void textViewClickedCell(NSTextView aTextView,
    NSCellForTextAttachment attachmentCell, NSRect cellFrame)
```

Discussion
This method has been deprecated in favor of `textViewClickedCellAtIndex` (page 1667).

See Also
`wantsToTrackMouse` (page 1939) (NSTextAttachmentCell)

textViewClickedCellAtIndex

Invoked after the user clicks on cell within *cellFrame* at the specified *charIndex* in an NSTextView and the cell wants to track the mouse.

```
public abstract void textViewClickedCellAtIndex(NSTextView aTextView,
    NSCellForTextAttachment cell, NSRect cellFrame, int charIndex)
```

Discussion
The delegate can use this message as its cue to perform an action or select the attachment cell's character. *aTextView* is the first NSTextView in a series shared by an NSLayoutManager, not necessarily the one that draws *cell*.

The delegate may subsequently receive a `textViewDoubleClickedCellAtIndex` (page 1669) message if the user continues to perform a double click.

See Also
`textViewDoubleClickedCellAtIndex` (page 1669)

textViewClickedOnLink

Invoked after the user clicks *link* in *aTextView* if the delegate does not respond to the `textViewClickedOnLinkAtIndex` (page 1668) message.

```
public abstract boolean textViewClickedOnLink(NSTextView aTextView, Object link)
```

Discussion
This method has been deprecated in favor of `textViewClickedOnLinkAtIndex` (page 1668).

See Also
`clickedOnLinkAtIndex` (page 1626) (NSTextView)
`textViewClickedOnLinkAtIndex` (page 1668)

textViewClickedOnLinkAtIndex

Invoked after the user clicks *link* at the specified *charIndex* in an NSTextView.

```
public abstract boolean textViewClickedOnLinkAtIndex(NSTextView aTextView, Object
     link, int charIndex)
```

Discussion
The delegate can use this method to handle the click on the link. Returns `true` to indicate that the click was handled; otherwise returns `false` to allow the next responder to handle it.

See Also
`clickedOnLinkAtIndex` (page 1626) (NSTextView)

textViewDidChangeSelection

Invoked when the selection changes in the NSTextView.

```
public abstract void textViewDidChangeSelection(NSNotification aNotification)
```

Discussion
The name of *aNotification* is `TextViewDidChangeSelectionNotification` (page 1672).

textViewDidChangeTypingAttributes

Delegate method invoked when text view's typing attributes change.

```
public abstract void textViewDidChangeTypingAttributes(NSNotification aNotification)
```

Discussion
The default name of *aNotification* is `TextViewDidChangeTypingAttributesNotification` (page 1673).

Allows the delegate to modify the notification sent when the typing attributes of the text view change.

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

textViewDoCommandBySelector

Sent from NSTextView's `doCommandBySelector` (page 1627), this method allows the delegate to perform the command for the text view.

```
public abstract boolean textViewDoCommandBySelector(NSTextView aTextView, NSSelector
     aSelector)
```

Discussion
If the delegate returns `true`, the text view doesn't perform *aSelector*; if the delegate returns `false`, the text view attempts to perform it. *aTextView* is the first NSTextView in a series shared by an NSLayoutManager.

textViewDoubleClickedCell

Invoked when the user double-clicks *attachmentCell* within *cellFrame* in *aTextView* and the cell wants to track the mouse.

```
public abstract void textViewDoubleClickedCell(NSTextView aTextView,
    NSCellForTextAttachment attachmentCell, NSRect cellFrame)
```

Discussion

This method has been deprecated in favor of `textViewDoubleClickedCellAtIndex` (page 1669).

See Also

`wantsToTrackMouse` (page 1939) (NSTextAttachmentCell)

`textViewDoubleClickedCellAtIndex` (page 1669)

textViewDoubleClickedCellAtIndex

Invoked when the user double-clicks *cell* within *cellFrame* at the specified *charIndex* in an NSTextView and the cell wants to track the mouse.

```
public abstract void textViewDoubleClickedCellAtIndex(NSTextView aTextView,
    NSCellForTextAttachment cell, NSRect cellFrame, int charIndex)
```

Discussion

The delegate can use this message as its cue to perform an action, such as opening the file represented by the attachment. *aTextView* is the first NSTextView in a series shared by an NSLayoutManager, not necessarily the one that draws *cell*.

See Also

`wantsToTrackMouse` (page 1939) (NSTextAttachmentCell)

textViewDraggedCell

Invoked when the user attempts to drag *cell* from *aRect* within *aTextView* and *cell* wants to track the mouse.

```
public abstract void textViewDraggedCell(NSTextView aTextView,
    NSCellForTextAttachment cell, NSRect aRect, NSEvent theEvent)
```

Discussion

theEvent is the mouse-down event that preceded the mouse-dragged event.

This method has been deprecated in favor of `textViewDraggedCellAtIndex` (page 1670).

See Also

`wantsToTrackMouse` (page 1939) (NSTextAttachmentCell)

`dragImage` (page 1751) (NSView)

`dragFile` (page 1749) (NSView)

textViewDraggedCellAtIndex

Invoked when the user attempts to drag *cell* from *aRect* within *aTextView* and the cell wants to track the mouse.

```
public abstract void textViewDraggedCellAtIndex(NSTextView aTextView,
    NSCellForTextAttachment cell, NSRect aRect, NSEvent theEvent, int charIndex)
```

Discussion

theEvent is the mouse-down event that preceded the mouse-dragged event. The *charIndex* parameter indicates the character position where the mouse button was clicked. The delegate can use this message as its cue to initiate a dragging operation.

See Also

`wantsToTrackMouse` (page 1939) (NSTextAttachmentCell)

`dragImage` (page 1751) (NSView)

`dragFile` (page 1749) (NSView)

textViewShouldChangeTextInRange

Invoked when an NSTextView needs to determine if text in the range *affectedCharRange* should be changed.

```
public abstract boolean textViewShouldChangeTextInRange(NSTextView aTextView,
    NSRange affectedCharRange, String replacementString)
```

Discussion

If characters in the text string are being changed, *replacementString* contains the characters that will replace the characters in *affectedCharRange*. If only text attributes are being changed, *replacementString* is `null`. The delegate can return `true` to allow the replacement, or `false` to reject the change.

The *aTextView* argument is the first NSTextView in a series shared by an NSLayoutManager.

If a delegate implements this method and not its multiple-selection replacement, `textViewShouldChangeTextInRanges` (page 1670), it is called with an appropriate range and string. If a delegate implements the new method, then this one is ignored.

textViewShouldChangeTextInRanges

Invoked when an NSTextView needs to determine if text in the range *affectedRanges* should be changed.

```
public abstract boolean textViewShouldChangeTextInRanges(NSTextView textView,
    NSArray affectedRanges, NSArray replacementStrings)
```

Discussion

If characters in the text string are being changed, *replacementStrings* contains an array of elements indicating the characters that will replace the characters in *affectedRanges*. The *replacementStrings* array must either be null or else contain one element for each range in *affectedRanges*. If only text attributes are being changed, *replacementStrings* is `null`. The delegate can return `true` to allow the replacement, or `false` to reject the change.

The *affectedRanges* argument must be a non-null, non-empty array of objects responding to the NSValue `rangeValue` method, and in addition its elements must be sorted, non-overlapping, non-contiguous, and (except for the case of a single range) have non-zero-length.

The *textView* argument is the first NSTextView in a series shared by an NSLayoutManager.

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

textViewShouldChangeTypingAttributes

Allows the delegate to intervene to allow, prevent, or modify changes to the typing attributes in *textView* from those contained in *oldTypingAttributes* to those contained in *newTypingAttributes*.

```
public abstract NSDictionary textViewShouldChangeTypingAttributes(NSTextView
    textView, NSDictionary oldTypingAttributes, NSDictionary newTypingAttributes)
```

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

textViewWillChangeSelection

Invoked before an NSTextView finishes changing the selection—that is, when the last argument to a `setSelectedRange` (page 1651) message is `false`.

```
public abstract NSRange textViewWillChangeSelection(NSTextView aTextView, NSRange
     oldSelectedCharRange, NSRange newSelectedCharRange)
```

Discussion
oldSelectedCharRange is the original range of the selection. *newSelectedCharRange* is the proposed character range for the new selection. The delegate can return an adjusted range or return *newSelectedCharRange* unmodified.

The *aTextView* argument is the first NSTextView in a series shared by an NSLayoutManager.

This is a multiple-selection variant of the method.

```
public abstract NSArray textViewWillChangeSelection(NSTextView aTextView, NSArray
     oldSelectedCharRanges, NSArray newSelectedCharRanges)
```

Discussion
The *oldSelectedCharRanges* argument is an array containing the original ranges of the selection. The *newSelectedCharRanges* argument is and array containing the proposed character ranges for the new selection. The delegate can return an array containing the adjusted ranges or return *newSelectedCharRanges* unmodified.

The *oldSelectedCharRanges* and *newSelectedCharRanges* arguments must be non-null, non-empty arrays of objects responding to the NSValue `rangeValue` method, and in addition their elements must be sorted, non-overlapping, non-contiguous, and (except for the case of a single range) have non-zero-length.

In Mac OS X version 10.4 and later, if a delegate implements the old single-selection variant of this delegate method and not its multiple-selection replacement, then multiple selection is effectively disallowed; attempts to set the selected ranges call the old delegate method with the first subrange, and afterwards only a single selected range is set.

If a delegate implements both the multiple-selection variant and the single-selection variant, then the latter is ignored.

Availability
Multiple-selection variant available in Mac OS X v10.4 and later.

textViewWillDisplayToolTip

Allows the delegate to modify the tool tip that will be displayed from that specified by `NSToolTipAttributeName`, or to suppress display of the tooltip (by returning `null`).

```
public abstract String textViewWillDisplayToolTip(NSTextView textView, String
    tooltip, int characterIndex)
```

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

undoManagerForTextView

Returns the undo manager instance for the text view specified by *aTextView*.

```
public abstract NSUndoManager undoManagerForTextView(NSTextView aTextView)
```

Discussion
This method provides the flexibility to return a custom undo manager for the text view. Although NSTextView implements undo and redo for changes to text, applications may need a custom undo manager to handle interactions between changes to text and changes to other items in the application.

Notifications

NSTextView posts the following notifications as well as those declared by its superclasses, particularly NSText. See the "Notifications" (page 1532) section in the NSText class specification for those other notifications.

TextViewDidChangeSelectionNotification

Posted when the selected range of characters changes.

NSTextView posts this notification whenever `setSelectedRange` (page 1651) is invoked, either directly or through the many methods (`mouseDown` (page 1192), `selectAll` (page 1521), and so on) that invoke it indirectly. When the user is selecting text, this notification is posted only once, at the end of the selection operation. The NSTextView's delegate receives a `textViewDidChangeSelection` (page 1668) message when this notification is posted.

Key	Value
`"NSOldSelectedCharacterRange"`	An NSRange with the originally selected range.

The notification object is the notifying NSTextView. The *`userInfo`* dictionary contains the following information:

TextViewWillChangeNotifyingTextViewNotification

Posted when a new NSTextView is established as the NSTextView that sends notifications.

This notification allows observers to reregister themselves for the new NSTextView. Methods such as `removeTextContainerAtIndex` (page 843), `textContainerChangedTextView` (page 853), and `insertTextContainerAtIndex` (page 837) cause this notification to be posted.

Key	Value
`"NSOldNotifyingTextView"`	The old NSTextView, if one exists, otherwise `null`.
`"NSNewNotifyingTextView"`	The new NSTextView, if one exists, otherwise `null`.

The notification object is the old notifying NSTextView, or `null`. The *`userInfo`* dictionary contains the following information:

There's no delegate method associated with this notification. The text-handling system ensures that when a new NSTextView replaces an old one as the notifying NSTextView, the existing delegate becomes the delegate of the new NSTextView, and the delegate is registered to receive NSTextView notifications from the new notifying NSTextView. All other observers are responsible for registering themselves on receiving this notification.

See Also

`removeObserver` (NSNotificationCenter)

`addObserver` (NSNotificationCenter)

TextViewDidChangeTypingAttributesNotification

Posted when there is a change in the typing attributes within an NSTextView. This notification is posted, via the `textViewDidChangeTypingAttributes` (page 1668) delegate method, whether or not text has changed as a result of the attribute change.

Availability

Available in Mac OS X v10.3 and later.

NSTokenField

Inherits from	NSTextField
Package:	com.apple.cocoa.application
Availability	Availabile in Mac OS X v10.4 and later.

Overview

NSTokenField is a subclass of NSTextField that provides tokenized editing similar to the address field in Mail.app.

NSTokenField uses an NSTokenFieldCell to implement much of the control's functionality. NSTokenField provides cover methods for most of NSTokenFieldCell's methods, which invoke the corresponding cell method.

Tasks

Constructors

`NSTokenField` (page 1677)

Setting the Completion Delay

`completionDelay` (page 1677)
Returns the receiver's completion delay.

`defaultCompletionDelay` (page 1677)
Returns the default completion delay.

`setCompletionDelay` (page 1678)
Sets the receiver's completion delay to delay.

Setting the Token Field Appearance

`setTokenStyle` (page 1678)
Returns the token style of the receiver.

`tokenStyle` (page 1679)
Returns the receiver's token style.

Setting the Tokenizing Character Set

`defaultTokenizingCharacterSet` (page 1677)
Returns the default tokenizing character set.

`tokenizingCharacterSet` (page 1678)
Returns the receiver's tokenizing character set.

`setTokenizingCharacterSet` (page 1678)
Sets the recevier's tokenizing character set to *characterSet*.

Display

`tokenFieldDisplayStringForRepresentedObject` (page 1679) *delegate method*
Allows the delegate to provide a string to be displayed as a proxy for the *representedObject*.

`tokenFieldStyleForRepresentedObject` (page 1681) *delegate method*
Allows the delegate to return the token style for editing the specified *representedObject*.

Editing

`tokenFieldCompletionsForSubstring` (page 1679) *delegate method*
Allows the delegate to provide an array of appropriate completions for the contents of the receiver.

`tokenFieldEditingStringForRepresentedObject` (page 1680) *delegate method*
Allows the delegate to provide a string to be edited as a proxy for the *representedObject*.

`tokenFieldRepresentedObjectForEditingString` (page 1681) *delegate method*
Allows the delegate to provide a represented object for *editingString*.

`tokenFieldShouldAddObjects` (page 1681) *delegate method*
Allows the delegate to validate the *tokens* to be added to the receiver at *index*.

Pasteboard

`tokenFieldReadFromPasteboard` (page 1680) *delegate method*
Allows the delegate to return an array of objects representing the data read from *pboard*.

`tokenFieldWriteRepresentedObjectsToPasteboard` (page 1681) *delegate method*
Allows the delegate the opportunity to write custom pasteboard types to the pasteboard for the represented objects in *objects*.

Menu

`tokenFieldHasMenuForRepresentedObject` (page 1680) *delegate method*
Allows the delegate to specify whether the *representedObject* provides a menu.

`tokenFieldMenuForRepresentedObject` (page 1680) *delegate method*
Allows the delegate to provide a menu for the specified *representedObject*.

Constructors

NSTokenField

```
public NSTokenField()
```

Discussion
Creates an NSTokenField object.

```
public NSTokenField(NSRect frame)
```

Discussion
Creates an NSTokenField object with the specified frame rectangle.

Static Methods

defaultCompletionDelay

Returns the default completion delay.

```
public static double defaultCompletionDelay()
```

Discussion
The default completion delay is 0.

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

defaultTokenizingCharacterSet

Returns the default tokenizing character set.

```
public static NSCharacterSet defaultTokenizingCharacterSet()
```

Discussion
The default tokenizing character set is “,”.

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

Instance Methods

completionDelay

Returns the receiver’s completion delay.

```
public double completionDelay()
```

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

See Also
`setCompletionDelay` (page 1678)
`defaultCompletionDelay` (page 1677)

setCompletionDelay

Sets the receiver's completion delay to delay.

```
public void setCompletionDelay(double delay)
```

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

See Also
`completionDelay` (page 1677)

setTokenizingCharacterSet

Sets the recevier's tokenizing character set to *characterSet*.

```
public void setTokenizingCharacterSet(NSCharacterSet characterSet)
```

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

See Also
`tokenizingCharacterSet` (page 1678)

setTokenStyle

Returns the token style of the receiver.

```
public void setTokenStyle(int style)
```

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

See Also
`tokenStyle` (page 1679)

tokenizingCharacterSet

Returns the receiver's tokenizing character set.

```
public NSCharacterSet tokenizingCharacterSet()
```

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

See Also
`setTokenizingCharacterSet` (page 1678)
`defaultTokenizingCharacterSet` (page 1677)

tokenStyle

Returns the receiver's token style.

```
public int tokenStyle()
```

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

See Also
`setTokenStyle` (page 1678)

Delegate Methods

tokenFieldCompletionsForSubstring

Allows the delegate to provide an array of appropriate completions for the contents of the receiver.

```
public abstract NSArray tokenFieldCompletionsForSubstring(NSTokenField tokenField,
     String substring, int tokenIndex)
```

Discussion
The *substring* is the partial string that to be completed. The *tokenIndex* is the index of the token being edited.

The default behavior provides no completions.

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

tokenFieldDisplayStringForRepresentedObject

Allows the delegate to provide a string to be displayed as a proxy for the *representedObject*.

```
public abstract String tokenFieldDisplayStringForRepresentedObject(NSTokenField
    tokenField, Object representedObject)
```

Discussion
If you return `null` or do not implement this method, then *representedObject* is displayed as the string. The represented object must implement the NSCoding interface.

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

tokenFieldEditingStringForRepresentedObject

Allows the delegate to provide a string to be edited as a proxy for the *representedObject*.

```
public abstract String tokenFieldEditingStringForRepresentedObject(NSTokenField
    tokenField, Object representedObject)
```

Discussion

If you return `null` or do not implement this method, then the value returned by `tokenFieldDisplayStringForRepresentedObject` (page 1679) is used for editing. The represented object must implement the NSCoding interface.

Availability

Available in Mac OS X v10.4 and later.

tokenFieldHasMenuForRepresentedObject

Allows the delegate to specify whether the *representedObject* provides a menu.

```
public abstract boolean tokenFieldHasMenuForRepresentedObject(NSTokenField
    tokenField, Object representedObject)
```

Discussion

By default tokens have no menus.

Availability

Available in Mac OS X v10.4 and later.

tokenFieldMenuForRepresentedObject

Allows the delegate to provide a menu for the specified *representedObject*.

```
public abstract NSMenu tokenFieldMenuForRepresentedObject(NSTokenField tokenField,
     Object representedObject)
```

Discussion

The returned menu should be autoreleased. By default tokens do not return menus.

Availability

Available in Mac OS X v10.4 and later.

tokenFieldReadFromPasteboard

Allows the delegate to return an array of objects representing the data read from *pboard*.

```
public abstract NSArray tokenFieldReadFromPasteboard(NSTokenField tokenField,
    NSPasteboard pboard)
```

Availability

Available in Mac OS X v10.4 and later.

tokenFieldRepresentedObjectForEditingString

Allows the delegate to provide a represented object for *editingString*.

```
public abstract Object tokenFieldRepresentedObjectForEditingString(NSTokenField
    tokenField, String editingString)
```

Discussion
The represented object must implement the NSCoding interface. If your application uses some object other than an NSString for their represented objects, you should return a new, autoreleased instance of that object from this method.

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

tokenFieldShouldAddObjects

Allows the delegate to validate the *tokens* to be added to the receiver at *index*.

```
public abstract NSArray tokenFieldShouldAddObjects(NSTokenField tokenField, NSArray
     tokens, int index)
```

Discussion
The delegate can return the array unchanged or return a modified array of tokens. To reject the add completely, return an empty array. Returning `null` causes an error.

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

tokenFieldStyleForRepresentedObject

Allows the delegate to return the token style for editing the specified *representedObject*.

```
public abstract int tokenFieldStyleForRepresentedObject(NSTokenField tokenField,
    Object representedObject)
```

Discussion
The delegate should return NSDefaultTokenStyle, NSPlainTextTokenStyle or NSRoundedTokenStyle.

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

tokenFieldWriteRepresentedObjectsToPasteboard

Allows the delegate the opportunity to write custom pasteboard types to the pasteboard for the represented objects in *objects*.

```
public abstract boolean tokenFieldWriteRepresentedObjectsToPasteboard(NSTokenField
     tokenField, NSArray objects, NSPasteboard pboard)
```

Discussion
The display strings for the represented objects have already been placed on the pasteboard as both `NSStringPboardType` and an array of NSStrings.

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

NSTokenFieldCell

Inherits from	NSTextFieldCell
Package:	com.apple.cocoa.application
Availability	Available in Mac OS X v10.4 and later.

Overview

NSTokenFieldCell is a subclass of NSTextFieldCell that provides tokenized editing of an array of objects similar to the address field in Mail.app. The objects may be strings or objects that can be represented as strings.

Tasks

Constructors

`NSTokenFieldCell` (page 1685)

Setting the Completion Delay

`completionDelay` (page 1686)
Returns the receiver's completion delay.

`defaultCompletionDelay` (page 1685)
Returns the default completion delay.

`setCompletionDelay` (page 1686)
Sets the receiver's completion delay to delay.

Setting the Token Style

`setTokenStyle` (page 1687)
Returns the token style of the receiver.

`tokenStyle` (page 1687)
Returns the receiver's token style.

Setting the Tokenizing Character Set

`defaultTokenizingCharacterSet` (page 1685)
    Returns the default tokenizing character set.

`tokenizingCharacterSet` (page 1687)
    Returns the receiver's tokenizing character set.

`setTokenizingCharacterSet` (page 1687)
    Sets the recevier's tokenizing character set to *characterSet*.

Getting and Setting the Delegate

`delegate` (page 1686)
    Returns the receiver's delegate.

`setDelegate` (page 1686)
    Sets the receiver's delegate to *anObject*.

Display

`tokenFieldCellDisplayStringForRepresentedObject` (page 1688) *delegate method*
    Allows the delegate to provide a string to be displayed as a proxy for the *representedObject*.

`tokenFieldCellStyleForRepresentedObject` (page 1690) *delegate method*
    Allows the delegate to return the token style for editing the specified *representedObject*.

Editing

`tokenFieldCellCompletionsForSubstring` (page 1688) *delegate method*
    Allows the delegate to provide an array of appropriate completions for the contents of the receiver.

`tokenFieldCellEditingStringForRepresentedObject` (page 1689) *delegate method*
    Allows the delegate to provide a string to be edited as a proxy for the *representedObject*.

`tokenFieldCellRepresentedObjectForEditingString` (page 1690) *delegate method*
    Allows the delegate to provide a represented object for *editingString*.

`tokenFieldCellShouldAddObjects` (page 1690) *delegate method*
    Allows the delegate to validate the *tokens* to be added to the receiver at *index*.

Pasteboard

`tokenFieldCellReadFromPasteboard` (page 1690) *delegate method*
    Allows the delegate to return an array of objects representing the data read from *pboard*.

`tokenFieldCellWriteRepresentedObjectsToPasteboard` (page 1691) *delegate method*
    Allows the delegate the opportunity to write custom pasteboard types to the pasteboard for the represented objects in *objects*.

Menu

`tokenFieldCellHasMenuForRepresentedObject` (page 1689) *delegate method*
Allows the delegate to specify whether the *`representedObject`* provides a menu.

`tokenFieldCellMenuForRepresentedObject` (page 1689) *delegate method*
Allows the delegate to provide a menu for the specified *`representedObject`*.

Constructors

NSTokenFieldCell

```
public NSTokenFieldCell()
```

Discussion
Creates an empty NSTokenFieldCell.

```
public NSTokenFieldCell(String aString)
```

Discussion
Creates an NSTokenFieldCell initialized with *`aString`* and set to have the cell's default menu. If no field editor (a shared NSText object) has been created for all cells, one is created.

```
public NSTokenFieldCell(NSImage anImage)
```

Discussion
Creates an NSTokenFieldCell initialized with *`anImage`* and set to have the cell's default menu. If *`anImage`* is `null`, no image is set.

Static Methods

defaultCompletionDelay

Returns the default completion delay.

```
public static double defaultCompletionDelay()
```

Discussion
The default completion delay is 0.

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

defaultTokenizingCharacterSet

Returns the default tokenizing character set.

```
public static NSCharacterSet defaultTokenizingCharacterSet()
```

Discussion
The default tokenizing character set is ",".

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

Instance Methods

completionDelay

Returns the receiver's completion delay.

```
public double completionDelay()
```

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

See Also
`setCompletionDelay` (page 1686)
`defaultCompletionDelay` (page 1685)

delegate

Returns the receiver's delegate.

```
public Object delegate()
```

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

See Also
`setDelegate` (page 1686)

setCompletionDelay

Sets the receiver's completion delay to delay.

```
public void setCompletionDelay(double delay)
```

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

See Also
`completionDelay` (page 1686)

setDelegate

Sets the receiver's delegate to *anObject*.

```
public void setDelegate(Object anObject)
```

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

See Also
`delegate` (page 1686)

setTokenizingCharacterSet

Sets the recevier's tokenizing character set to *characterSet*.

```
public void setTokenizingCharacterSet(NSCharacterSet characterSet)
```

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

See Also
`tokenizingCharacterSet` (page 1687)

setTokenStyle

Returns the token style of the receiver.

```
public void setTokenStyle(int style)
```

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

See Also
`tokenStyle` (page 1687)

tokenizingCharacterSet

Returns the receiver's tokenizing character set.

```
public NSCharacterSet tokenizingCharacterSet()
```

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

See Also
`setTokenizingCharacterSet` (page 1687)
`defaultTokenizingCharacterSet` (page 1685)

tokenStyle

Returns the receiver's token style.

```
public int tokenStyle()
```

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

See Also
`setTokenStyle` (page 1687)

Constants

The NSTokenStyle constants define how tokens are displayed and editable in the NSTokenFieldCell. These values are used by `tokenStyle` (page 1687), `setTokenStyle` (page 1687) and the delegate method `tokenFieldCellStyleForRepresentedObject` (page 1690)

Constant	Description
`DefaultTokenStyle`	Use the specified token style for the receiver. Available in Mac OS X v10.4 and later.
`PlainTextTokenStyle`	Plain text tokens. Available in Mac OS X v10.4 and later.
`RoundedTokenStyle`	Rounded token style. Available in Mac OS X v10.4 and later.

Delegate Methods

tokenFieldCellCompletionsForSubstring

Allows the delegate to provide an array of appropriate completions for the contents of the receiver.

```
public abstract NSArray tokenFieldCellCompletionsForSubstring(NSTokenFieldCell
    tokenFieldCell, String substring, int tokenIndex)
```

Discussion
The *substring* is the partial string that to be completed. The *tokenIndex* is the index of the token being edited. The *selectedIndex* allows you to return by-reference an index in the array specifying which of the completions should be initially selected.

The default behavior provides no completions.

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

tokenFieldCellDisplayStringForRepresentedObject

Allows the delegate to provide a string to be displayed as a proxy for the *representedObject*.

```
public abstract String
    tokenFieldCellDisplayStringForRepresentedObject(NSTokenFieldCell tokenFieldCell,
     Object representedObject)
```

Discussion
If you return `null` or do not implement this method, then *`representedObject`* is displayed as the string.

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

tokenFieldCellEditingStringForRepresentedObject

Allows the delegate to provide a string to be edited as a proxy for the *`representedObject`*.

```
public abstract String
    tokenFieldCellEditingStringForRepresentedObject(NSTokenFieldCell tokenFieldCell,
     Object representedObject)
```

Discussion
If you return `null` or do not implement this method, then the value returned by `tokenFieldCellDisplayStringForRepresentedObject` (page 1688) is used for editing.

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

tokenFieldCellHasMenuForRepresentedObject

Allows the delegate to specify whether the *`representedObject`* provides a menu.

```
public abstract boolean tokenFieldCellHasMenuForRepresentedObject(NSTokenFieldCell
     tokenFieldCell, Object representedObject)
```

Discussion
By default tokens have no menus.

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

tokenFieldCellMenuForRepresentedObject

Allows the delegate to provide a menu for the specified *`representedObject`*.

```
public abstract NSMenu tokenFieldCellMenuForRepresentedObject(NSTokenFieldCell
    tokenFieldCell, Object representedObject)
```

Discussion
The returned menu should be autoreleased. By default tokens do not return menus.

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

tokenFieldCellReadFromPasteboard

Allows the delegate to return an array of objects representing the data read from *pboard*.

```
public abstract NSArray tokenFieldCellReadFromPasteboard(NSTokenFieldCell
    tokenFieldCell, NSPasteboard pboard)
```

Availability

Available in Mac OS X v10.4 and later.

tokenFieldCellRepresentedObjectForEditingString

Allows the delegate to provide a represented object for *editingString*.

```
public abstract Object
    tokenFieldCellRepresentedObjectForEditingString(NSTokenFieldCell tokenFieldCell,
     String editingString)
```

Discussion

If your application uses some object other than an NSString for their represented objects, you should return a new, autoreleased instance of that object from this method.

Availability

Available in Mac OS X v10.4 and later.

tokenFieldCellShouldAddObjects

Allows the delegate to validate the *tokens* to be added to the receiver at *index*.

```
public abstract NSArray tokenFieldCellShouldAddObjects(NSTokenFieldCell
    tokenFieldCell, NSArray tokens, int index)
```

Discussion

The delegate can return the array unchanged or return a modified array of tokens. To reject the add completely, return an empty array. Returning `null` causes an error.

Availability

Available in Mac OS X v10.4 and later.

tokenFieldCellStyleForRepresentedObject

Allows the delegate to return the token style for editing the specified *representedObject*.

```
public abstract int tokenFieldCellStyleForRepresentedObject(NSTokenFieldCell
    tokenFieldCell, Object representedObject)
```

Discussion

The delegate should return NSDefaultTokenStyle, NSPlainTextTokenStyle or NSRoundedTokenStyle.

Availability

Available in Mac OS X v10.4 and later.

tokenFieldCellWriteRepresentedObjectsToPasteboard

Allows the delegate the opportunity to write custom pasteboard types to the pasteboard for the represented objects in *objects*.

```
public abstract boolean
    tokenFieldCellWriteRepresentedObjectsToPasteboard(NSTokenFieldCell
    tokenFieldCell, NSArray objects, NSPasteboard pboard)
```

Discussion

The display strings for the represented objects have already been placed on the pasteboard as both `NSStringPboardType` and an array of NSStrings.

Availability

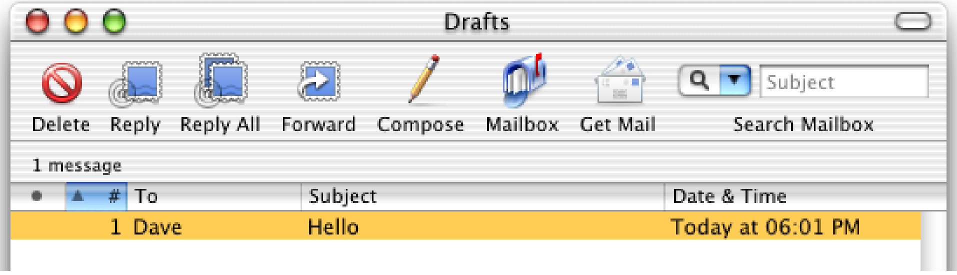
Available in Mac OS X v10.4 and later.

NSToolbar

Inherits from	NSObject
Package:	com.apple.cocoa.application
Companion guide	Toolbar Programming Topics for Cocoa

Overview

NSToolbar and NSToolbarItem (page 1707) provide the mechanism for a titled window to display a toolbar just below its title bar, as shown below:



1 message

●	▲	#	To	Subject	Date & Time
		1	Dave	Hello	Today at 06:01 PM

Tasks

Constructors

`NSToolbar` (page 1696)
Creates and initializes an NSToolbar with *`identifier`*, which is used by the toolbar to identify the kind of the toolbar.

Toolbar Attributes

`displayMode` (page 1697)
Returns the receiver's display mode, which is one of the values described in "Constants" (page 1703).

`setDisplayMode` (page 1700)
Sets the receiver's display mode.

`showsBaselineSeparator` (page 1702)
Returns whether the toolbar shows the separator between the toolbar and the main window contents.

`setShowsBaselineSeparator` (page 1701)
    Sets whether the toolbar shows the separator between the toolbar and the main window contents.

`allowsUserCustomization` (page 1696)
    Answers whether or not users are allowed to modify the toolbar.

`setAllowsUserCustomization` (page 1699)
    Sets whether or not users are allowed to modify the toolbar to *allowsCustomization*.

`identifier` (page 1698)
    Returns the receiver's identifier, which identifies the kind of the toolbar.

`items` (page 1698)
    Returns all current items in the receiver, in order.

`visibleItems` (page 1703)
    Returns the receiver's currently visible items. An item can be invisible if it has spilled over into the overflow menu.

`sizeMode` (page 1702)
    Returns the receiver's size mode, which is one of the values described in "Constants" (page 1703).

`setSizeMode` (page 1701)
    Sets the receiver's size mode.

Managing the Delegate

`delegate` (page 1697)
    Returns the receiver's delegate.

`setDelegate` (page 1700)
    Sets the receiver's delegate to *delegate*.

Managing Items on the Toolbar

`insertItemWithItemIdentifierAtIndex` (page 1698)
    Inserts the item identified by *itemIdentifier* at *index*.

`removeItemAtIndex` (page 1698)
    Removes the item at *index*.

`setSelectedItemIdentifier` (page 1701)
    Sets the receiver's selected item to the toolbar item specified by *identifier*.

`selectedItemIdentifier` (page 1699)
    Returns the identifier of the receiver's currently selected item, or `null`

Displaying the Toolbar

`isVisible` (page 1698)
    Returns whether the receiver is visible.

`setVisible` (page 1702)
    Sets whether the receiver is visible or hidden to *shown*.

Toolbar Customization

`runCustomizationPalette` (page 1699)
    Runs the receiver's customization palette.

`customizationPaletteIsRunning` (page 1697)
    Returns whether the receiver's customization palette is running (in use).

Autosaving the Configuration

`autosavesConfiguration` (page 1696)
    Returns whether the receiver autosaves its configuration.

`setAutosavesConfiguration` (page 1700)
    Sets whether the receiver autosaves its configuration.

`configurationDictionary` (page 1697)
    Returns the receiver's configuration as a dictionary.

`setConfigurationFromDictionary` (page 1700)
    Sets the receiver's configuration using *configDict*.

Toolbar Item Validation

`validateVisibleItems` (page 1702)
    Called on window updates with the purpose of validating each of the visible items.

Adding and removing items

`toolbarWillAddItem` (page 1705) *delegate method*
    Posted just before a new item is added to the toolbar.

`toolbarDidRemoveItem` (page 1704) *delegate method*
    Posted just after an item has been removed from *toolbar*.

Working with item identifiers

`toolbarAllowedItemIdentifiers` (page 1704) *delegate method*
    Returns an array of toolbar item identifiers for *toolbar*, specifying the contents and the order of the items in the configuration palette.

`toolbarDefaultItemIdentifiers` (page 1704) *delegate method*
    Returns an array of toolbar item identifiers for *toolbar*, specifying the contents and the order of the items in the default toolbar configuration.

`toolbarItemForItemIdentifier` (page 1704) *delegate method*
    Returns a toolbar item of the kind identified by the given toolbar *itemIdentifier*.

`toolbarSelectableItemIdentifiers` (page 1705) *delegate method*
    Returns an array of item identifiers that should indicate selection in the specified *toolbar*.

Constructors

NSToolbar

Creates and initializes an NSToolbar with `identifier`, which is used by the toolbar to identify the kind of the toolbar.

```
public NSToolbar(String identifier)
```

Discussion

`identifier` is never seen by users and should not be localized. See `identifier` (page 1698) for important information.

See Also

`identifier` (page 1698)

Instance Methods

allowsUserCustomization

Answers whether or not users are allowed to modify the toolbar.

```
public boolean allowsUserCustomization()
```

Discussion

The default value is `false`. If the value is `false`, then the Customize Toolbar… menu item is disabled and other modification is disabled. If the value is `true`, be sure to also set `autosavesConfiguration` to `true`, so the user's changes persist. This attribute does not affect the user's ability to show or hide the toolbar.

See Also

`setAllowsUserCustomization` (page 1699)

`autosavesConfiguration` (page 1696)

autosavesConfiguration

Returns whether the receiver autosaves its configuration.

```
public boolean autosavesConfiguration()
```

Discussion

If `true`, the receiver will automatically write the toolbar settings to user defaults if the toolbar configuration changes. The default is `false`.

The information about the toolbar configuration is identified in user defaults by the toolbar identifier. If there are multiple toolbars active with the same identifier, then they all share the same configuration, so it is this shared configuration that is saved.

See Also

`setAutosavesConfiguration` (page 1700)

configurationDictionary (page 1697)

configurationDictionary

Returns the receiver's configuration as a dictionary.

```
public NSDictionary configurationDictionary()
```

Discussion
Contains displayMode, isVisible, and a list of the item identifiers currently in the toolbar. You can add your own items if you need to.

Do not depend on any details of the normal contents of a configuration dictionary.

See Also
setConfigurationFromDictionary (page 1700)

customizationPaletteIsRunning

Returns whether the receiver's customization palette is running (in use).

```
public boolean customizationPaletteIsRunning()
```

See Also
runCustomizationPalette (page 1699)

delegate

Returns the receiver's delegate.

```
public Object delegate()
```

Discussion
Every toolbar must have a delegate, which must implement the required delegate methods.

See Also
setDelegate (page 1700)

displayMode

Returns the receiver's display mode, which is one of the values described in "Constants" (page 1703).

```
public int displayMode()
```

See Also
setDisplayMode (page 1700)

identifier

Returns the receiver's identifier, which identifies the kind of the toolbar.

```
public String identifier()
```

Discussion

Within the application all toolbars with the same identifier are synchronized to maintain the same state, including for example, the display mode and item order. Also, if a toolbar autosaves its configuration, the item identifier is used as the autosave name.

insertItemWithItemIdentifierAtIndex

Inserts the item identified by *itemIdentifier* at *index*.

```
public void insertItemWithItemIdentifierAtIndex(String itemIdentifier, int index)
```

Discussion

If the toolbar needs a new instance, it will get it from `toolbarItemForItemIdentifier` (page 1704). Typically, you should not call this method; you should let the user reconfigure the toolbar. See `identifier` (page 1698) for important information.

See Also

`removeItemAtIndex` (page 1698)

isVisible

Returns whether the receiver is visible.

```
public boolean isVisible()
```

See Also

`setVisible` (page 1702)

items

Returns all current items in the receiver, in order.

```
public NSArray items()
```

See Also

`visibleItems` (page 1703)

removeItemAtIndex

Removes the item at *index*.

```
public void removeItemAtIndex(int index)
```

Discussion

Typically, you should not call this method; you should let the user reconfigure the toolbar. See `identifier` (page 1698) for important information.

See Also

`insertItemWithItemIdentifierAtIndex` (page 1698)

runCustomizationPalette

Runs the receiver's customization palette.

```
public void runCustomizationPalette(Object sender)
```

Discussion

When the user is done customizing, the palette will be dismissed.

See Also

`customizationPaletteIsRunning` (page 1697)

selectedItemIdentifier

Returns the identifier of the receiver's currently selected item, or `null`

```
public String selectedItemIdentifier()
```

Discussion

if there is no selection.

Availability

Available in Mac OS X v10.3 and later.

See Also

`setSelectedItemIdentifier` (page 1701)

setAllowsUserCustomization

Sets whether or not users are allowed to modify the toolbar to *`allowsCustomization`*.

```
public void setAllowsUserCustomization(boolean allowsCustomization)
```

Discussion

This value can be changed at any time. For instance, you may not want users to be able to customize the toolbar while some event is being processed. This attribute does not affect the user's ability to show or hide the toolbar.

See Also

`allowsUserCustomization` (page 1696)

setAutosavesConfiguration

Sets whether the receiver autosaves its configuration.

```
public void setAutosavesConfiguration(boolean flag)
```

Discussion

If `true`, the receiver will automatically write changes the user makes to user defaults. Customizable toolbars will want to set *flag* to `true`. Setting it to `false` means changes in configuration are not written automatically; however you can use the `configurationDictionary` (page 1697) to do it yourself.

See Also

`allowsUserCustomization` (page 1696)
`autosavesConfiguration` (page 1696)

setConfigurationFromDictionary

Sets the receiver's configuration using *configDict*.

```
public void setConfigurationFromDictionary(NSDictionary configDict)
```

Discussion

This method also immediately affects toolbars of the same kind in other windows.

See Also

`configurationDictionary` (page 1697)

setDelegate

Sets the receiver's delegate to *delegate*.

```
public void setDelegate(Object delegate)
```

Discussion

Every toolbar must have a delegate, which must implement the required delegate methods.

See Also

`delegate` (page 1697)

setDisplayMode

Sets the receiver's display mode.

```
public void setDisplayMode(int displayMode)
```

Discussion

displayMode is one of the values described in "Constants" (page 1703).

See Also

`displayMode` (page 1697)

setSelectedItemIdentifier

Sets the receiver's selected item to the toolbar item specified by *identifier*.

```
public void setSelectedItemIdentifier(String itemIdentifier)
```

Discussion

Typically, a toolbar will manage the selection of items automatically. This method can be used to select identifiers of custom view items, or to force a selection change. See `toolbarSelectableItemIdentifiers` (page 1705) for more details. If *itemIdentifier* is not recognized by the receiver, the current selected item identifier does not change.

itemIdentifier may be any identifier returned by `toolbarSelectableItemIdentifiers`, even if it is not currently in the toolbar. If the selected item is removed from the toolbar, the `selectedItemIdentifier` does not change.

Availability

Available in Mac OS X v10.3 and later.

See Also

`selectedItemIdentifier` (page 1699)
`toolbarSelectableItemIdentifiers` (page 1705)

setShowsBaselineSeparator

Sets whether the toolbar shows the separator between the toolbar and the main window contents.

```
public void setShowsBaselineSeparator(boolean flag)
```

Discussion

If *flag* is `true` the baseline is shown. The default is `true`.

Availability

Available in Mac OS X v10.4 and later.

See Also

`showsBaselineSeparator` (page 1702)

setSizeMode

Sets the receiver’s size mode.

```
public void setSizeMode(int sizeMode)
```

Discussion

The size can be:

- `SizeModeRegular`.The toolbar uses regular-sized controls and 32 by 32 pixel icons.
- `SizeModeSmall`. The toolbar uses small-sized controls and 24 by 24 pixel icons.

If there is no icon of the given size for a toolbar item, the toolbar item creates one by scaling another icon.

Availability
Available in Mac OS X v10.2 and later.

See Also
`sizeMode` (page 1702)

setVisible

Sets whether the receiver is visible or hidden to *shown*.

```
public void setVisible(boolean shown)
```

See Also
`isVisible` (page 1698)

showsBaselineSeparator

Returns whether the toolbar shows the separator between the toolbar and the main window contents.

```
public boolean showsBaselineSeparator
```

Discussion
The default is `true`.

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

See Also
`setShowsBaselineSeparator` (page 1701)

sizeMode

Returns the receiver's size mode, which is one of the values described in "Constants" (page 1703).

```
public int sizeMode()
```

Availability
Available in Mac OS X v10.2 and later.

See Also
`setSizeMode` (page 1701)

validateVisibleItems

Called on window updates with the purpose of validating each of the visible items.

```
public void validateVisibleItems()
```

Discussion

You use this method by overriding it in a subclass; you should not invoke this method. The toolbar calls this method to iterate through the list of visible items, sending each a `validate` message. Override it and call `super` if you want to know when this happens.

visibleItems

Returns the receiver's currently visible items. An item can be invisible if it has spilled over into the overflow menu.

```
public NSArray visibleItems()
```

See Also

`items` (page 1698)

Constants

The following constants toolbar display modes and are used by `displayMode` (page 1697) and `setDisplayMode` (page 1700):

Constant	Description
`DisplayModeDefault`	The default display mode.
`DisplayModeIconAndLabel`	The toolbar will display icons and labels.
`DisplayModeIconOnly`	The toolbar will display only icons.
`DisplayModeLabelOnly`	The toolbar will display only labels.

The following constants specify toolbar sizes and are used by `sizeMode` (page 1702) and `setSizeMode` (page 1701):

Constant	Description
`SizeModeDefault`	The toolbar uses the system-defined default size, which is `SizeModeRegular`.
`SizeModeRegular`	The toolbar uses regular-sized controls and 32 by 32 pixel icons.
`SizeModeSmall`	The toolbar uses small-sized controls and 24 by 24 pixel icons.

Delegate Methods

toolbarAllowedItemIdentifiers

Returns an array of toolbar item identifiers for `toolbar`, specifying the contents and the order of the items in the configuration palette.

```
public abstract NSArray toolbarAllowedItemIdentifiers(NSToolbar toolbar)
```

Discussion
Every allowed item must be explicitly listed, even the standard ones. The identifiers returned should include all of those returned by `toolbarDefaultItemIdentifiers`.

Implementation of this method is required.

toolbarDefaultItemIdentifiers

Returns an array of toolbar item identifiers for `toolbar`, specifying the contents and the order of the items in the default toolbar configuration.

```
public abstract NSArray toolbarDefaultItemIdentifiers(NSToolbar toolbar)
```

Discussion
During initialization of `toolbar`, this method is called only if a toolbar configuration for the identifier of `toolbar` is not found in the user preferences. This method is called during initialization of the toolbar customization palette.

Implementation of this method is required.

toolbarDidRemoveItem

Posted just after an item has been removed from `toolbar`.

```
public abstract void toolbarDidRemoveItem(NSNotification notification)
```

Discussion
This method allows the chance to remove information related to the item that may have been cached. The notification's `object` is the toolbar from which the item is being removed. The notification's `userInfo` dictionary contains the item being removed under the key `"item"`. If this method is implemented, then when you set the toolbar's delegate, the toolbar automatically registers this method for `DidRemoveItemNotification` (page 1706).

Implementation of this method is optional.

toolbarItemForItemIdentifier

Returns a toolbar item of the kind identified by the given toolbar `itemIdentifier`.

```
public abstract NSToolbarItem toolbarItemForItemIdentifier(NSToolbar toolbar, String
     itemIdentifier, boolean flag)
```

Discussion

This is where you create new toolbar item instances. This method is called lazily on behalf of a toolbar instance, which must be the sole owner of the toolbar item. A toolbar may ask again for a kind of toolbar item already supplied to it, in which case this method can and should return the same toolbar item it returned before.

If the item is a custom view item, the NSView must be fully-formed when the item is returned. Do not assume that the returned item is going to be added as an active item in the toolbar, as it could be that it will be used only in the customization palette. (The customization palette makes a copy of the returned item.)

A `null` return value tells the toolbar that the identified kind of toolbar item is not supported.

If *`flag`* is `true`, the returned item will be inserted into the toolbar, and you can expect that `toolbarWillAddItem` (page 1705) is about to be called.

Implementation of this method is required.

toolbarSelectableItemIdentifiers

Returns an array of item identifiers that should indicate selection in the specified *`toolbar`*.

```
public abstract NSArray toolbarSelectableItemIdentifiers(NSToolbar toolbar)
```

Discussion

Toolbars that need to indicate item selection should return an array containing the identifiers of the selectable toolbar items.

If implemented, *`toolbar`* will display the currently selected item with a visual highlight. Clicking on an item whose identifier is selectable will automatically update the toolbars selected item identifier, when possible.

Implementation of this method is optional.

Availability

Available in Mac OS X v10.3 and later.

See Also

`setSelectedItemIdentifier` (page 1701)

toolbarWillAddItem

Posted just before a new item is added to the toolbar.

```
public abstract void toolbarWillAddItem(NSNotification notification)
```

Discussion

If you need to cache a reference to the toolbar item or need to set up some initial state, this is where to do it. The object attribute of *`notification`* is the toolbar to which the item is being added. The notification's `userInfo` dictionary contains the item being added under the key `"item"`. If this method is implemented, then when you set the toolbar's delegate, the toolbar automatically registers this method for `WillAddItemNotification` (page 1706).

Implementation of this method is optional.

Notifications

DidRemoveItemNotification

Posted after an item is removed from a toolbar. The notification item is the NSToolbar that had an item removed from it. The *userInfo* dictionary contains the following information:

Key	Value
"item"	The NSToolbarItem that was removed.

See Also
toolbarDidRemoveItem (page 1704)

WillAddItemNotification

Posted before a new item is added to the toolbar. The notification item is the NSToolbar having an item added to it. The *userInfo* dictionary contains the following information:

Key	Value
"item"	The NSToolbarItem being added.

See Also
toolbarWillAddItem (page 1705)

NSToolbarItem

Inherits from	NSObject
Implements	NSValidatedUserInterfaceItem
Package:	com.apple.cocoa.application
Companion guide	Toolbar Programming Topics for Cocoa

Overview

Each item in an NSToolbar (page 1693) is an instance of NSToolbarItem.

Interfaces Implemented

NSValidatedUserInterfaceItem
- `action` (page 2033)
- `tag` (page 2033)

Tasks

Constructors

`NSToolbarItem` (page 1709)
Creates and initializes an NSToolbarItem with *`itemIdentifier`*, which is used by the toolbar and its delegate to identify the kind of the toolbar item.

Managing Attributes

`itemIdentifier` (page 1711)
Returns the receiver's identifier, which was provided in the constructor.

`toolbar` (page 1716)
Returns the toolbar that is using the receiver.

`label` (page 1711)
Returns the receiver's label, which normally appears in the toolbar and in the overflow menu.

`setLabel` (page 1713)
: Sets the receiver's label that appears in the toolbar to *`label`*.

`paletteLabel` (page 1712)
: Returns the label that appears when the receiver is in the customization palette.

`setPaletteLabel` (page 1714)
: Sets the receiver's label that appears when it is in the customization palette to *`paletteLabel`*.

`toolTip` (page 1716)
: Returns the tooltip used when the receiver is displayed in the toolbar.

`setToolTip` (page 1715)
: Sets the tooltip to be used when the receiver is displayed in the toolbar to *`toolTip`*.

`menuFormRepresentation` (page 1711)
: Returns the receiver's menu form representation.

`setMenuFormRepresentation` (page 1714)
: Sets the receiver's menu form to *`menuItem`*.

`tag` (page 1716)
: Returns the receiver's tag, which can be used for your own custom purpose.

`setTag` (page 1715)
: Sets the receiver's tag to *`tag`*, which can be used for your own custom purpose.

`target` (page 1716)
: Returns the receiver's target.

`setTarget` (page 1715)
: Sets the receiver's target.

`action` (page 1709)
: Returns the receiver's action.

`setAction` (page 1712)
: Sets the receiver's action to *`action`*.

`isEnabled` (page 1710)
: Returns the receiver's enabled status.

`setEnabled` (page 1713)
: Sets the receiver's enabled flag to *`enabled`*.

`image` (page 1710)
: Returns the image of the receiver.

`setImage` (page 1713)
: Sets the image of the receiver or of the view to *`image`*, if the view has already been set.

`view` (page 1717)
: Returns the receiver's view.

`setView` (page 1715)
: Use this method to make the receiver into a view item.

`minSize` (page 1712)
: Returns the receiver's minimum size.

`setMinSize` (page 1714)
: Sets the receiver's minimum size to *`size`*.

`maxSize` (page 1711)
: Returns the receiver's maximum size.

`setMaxSize` (page 1714)
    Sets the receiver's maximum size to *size*.

Visibility Priority

`visibilityPriority` (page 1717)
    Returns the receiver's visibility priority.

`setVisibilityPriority` (page 1716)
    Sets the receiver's visibility priority to *visibilityPriority*.

Validation

`validate` (page 1717)
    This method is called by the receiver's toolbar during validation.

`autovalidates` (page 1710)
    Returns whether the receiver is automatically validated by the toolbar.

`setAutovalidates` (page 1712)
    Sets the receiver's auto validation flag to *enabled*.

Controlling Duplicates

`allowsDuplicatesInToolbar` (page 1710)
    Returns `true` to allow dragging the receiver into the toolbar at more than one position.

Constructors

NSToolbarItem

Creates and initializes an NSToolbarItem with *itemIdentifier*, which is used by the toolbar and its delegate to identify the kind of the toolbar item.

```
public NSToolbarItem(String itemIdentifier)
```

Discussion
itemIdentifier is never seen by users and should not be localized.

Instance Methods

action

Returns the receiver's action.

```
public NSSelector action()
```

Discussion
For custom view items, this method sends `action` to the view if it responds and returns the result.

See Also
`setAction` (page 1712)

allowsDuplicatesInToolbar

Returns `true` to allow dragging the receiver into the toolbar at more than one position.

```
public boolean allowsDuplicatesInToolbar()
```

Discussion
You use this method by overriding it in a subclass to always return `true`; typically, you wouldn't call it. By default, if an item with the same identifier is already in the toolbar, dragging it in again will effectively move it to the new position.

autovalidates

Returns whether the receiver is automatically validated by the toolbar.

```
public boolean autovalidates()
```

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

See Also
`setAutovalidates` (page 1712)

image

Returns the image of the receiver.

```
public NSImage image()
```

Discussion
For an image item this method returns the result of the most recent `setImage` (page 1713). For view items, this method calls `image` on the view if it responds and returns the result.

See Also
`setImage` (page 1713)

isEnabled

Returns the receiver's enabled status.

```
public boolean isEnabled()
```

Discussion

For a view item, this method calls `isEnabled` on the view if it responds and returns the result.

See Also

`setEnabled` (page 1713)

`view` (page 1717)

itemIdentifier

Returns the receiver's identifier, which was provided in the constructor.

```
public String itemIdentifier()
```

See Also

`NSToolbarItem` (page 1709)

label

Returns the receiver's label, which normally appears in the toolbar and in the overflow menu.

```
public String label()
```

Discussion

For a discussion on labels, see "Setting a Toolbar Item's Representation".

See Also

`setLabel` (page 1713)

`menuFormRepresentation` (page 1711)

maxSize

Returns the receiver's maximum size.

```
public NSSize maxSize()
```

Discussion

See "Setting a Toolbar Item's Size" for a discussion on item sizes.

See Also

`setMaxSize` (page 1714)

menuFormRepresentation

Returns the receiver's menu form representation.

```
public NSMenuItem menuFormRepresentation()
```

Discussion

For a discussion on menu forms, see "Setting a Toolbar Item's Representation".

By default, this method returns `null`, even though there is a default menu form representation.

See Also

`setMenuFormRepresentation` (page 1714)

minSize

Returns the receiver's minimum size.

```
public NSSize minSize()
```

Discussion

See "Setting a Toolbar Item's Size" for a discussion on item sizes.

See Also

`setMinSize` (page 1714)

paletteLabel

Returns the label that appears when the receiver is in the customization palette.

```
public String paletteLabel()
```

Discussion

An item must have a palette label if the customization palette is to be used, and for most items it is reasonable to set `paletteLabel` to be the same value as `label` (page 1711). One reason for `paletteLabel` to be different from `label` (page 1711) would be if it's more descriptive; another might be if there is no `label` (page 1711).

See Also

`setPaletteLabel` (page 1714)

setAction

Sets the receiver's action to *`action`*.

```
public void setAction(NSSelector action)
```

Discussion

For a custom view item, this method calls `setAction` on the view if it responds.

See *Action Messages* for additional information on action messages.

See Also

`action` (page 1709)

`setTarget` (page 1715)

setAutovalidates

Sets the receiver's auto validation flag to *`enabled`*.

```
public void setAutovalidates(boolean resistance)
```

Discussion
By default NSToolbar automatically invokes the receiver's validate method on a regular basis. If your validate method is time consuming, you can disable auto validation on a per toolbar item basis.

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

See Also
`autovalidates` (page 1710)

setEnabled

Sets the receiver's enabled flag to *`enabled`*.

```
public void setEnabled(boolean enabled)
```

Discussion
For a custom view item, this method calls `setEnabled` on the view if it responds.

See Also
`isEnabled` (page 1710)

setImage

Sets the image of the receiver or of the view to *`image`*, if the view has already been set.

```
public void setImage(NSImage image)
```

Discussion
For a custom view item (one whose view has already been set), this method calls `setImage` on the view if it responds. If *`image`* contains multiple representations, NSToolbarItem chooses the most appropriately sized representation when displaying.

See Also
`image` (page 1710)
`view` (page 1717)

setLabel

Sets the receiver's label that appears in the toolbar to *`label`*.

```
public void setLabel(String label)
```

Discussion
The implication is that the toolbar will draw the label for the receiver, and a redraw is triggered by this method. The toolbar is in charge of the label area. It's OK for an item to have an empty label. You should make sure the length of the label is appropriate and not too long. For a discussion on labels, see "Setting a Toolbar Item's Representation".

See Also
`label` (page 1711)

`paletteLabel` (page 1712)

setMaxSize

Sets the receiver's maximum size to *`size`*.

```
public void setMaxSize(NSSize size)
```

Discussion

See "Setting a Toolbar Item's Size" for a discussion on item sizes.

See Also

`maxSize` (page 1711)

setMenuFormRepresentation

Sets the receiver's menu form to *`menuItem`*.

```
public void setMenuFormRepresentation(NSMenuItem menuItem)
```

Discussion

For a discussion on menu forms see "Setting a Toolbar Item's Representation".

See Also

`menuFormRepresentation` (page 1711)

setMinSize

Sets the receiver's minimum size to *`size`*.

```
public void setMinSize(NSSize size)
```

Discussion

See "Setting a Toolbar Item's Size" for a discussion on item sizes.

See Also

`minSize` (page 1712)

setPaletteLabel

Sets the receiver's label that appears when it is in the customization palette to *`paletteLabel`*.

```
public void setPaletteLabel(String paletteLabel)
```

Discussion

An item must have a palette label if the customization palette is to be used, and for most items it is reasonable to set `paletteLabel` (page 1712) to be the same value as `label` (page 1711). One reason for `paletteLabel` (page 1712) to be different from `label` (page 1711) would be if it's more descriptive; another might be if there is no `label` (page 1711).

See Also

`paletteLabel` (page 1712)

`setLabel` (page 1713)

setTag

Sets the receiver's tag to *`tag`*, which can be used for your own custom purpose.

```
public void setTag(int tag)
```

See Also

`tag` (page 1716)

setTarget

Sets the receiver's target.

```
public void setTarget(Object target)
```

Discussion

If *`target`* is unset, the toolbar will call `action` on the first responder that implements it.

See Also

`target` (page 1716)

`setAction` (page 1712)

`validateToolbarItem` (page 2031) (NSToolbar.ItemValidation)

setToolTip

Sets the tooltip to be used when the receiver is displayed in the toolbar to *`toolTip`*.

```
public void setToolTip(String toolTip)
```

See Also

`toolTip` (page 1716)

setView

Use this method to make the receiver into a view item.

```
public void setView(NSView view)
```

Discussion

Note that many of the set/get methods are implemented by calls forwarded to *`view`*, if it responds to it. Also, everything recursively contained in *`view`* must be archivable if the customization palette is used. For a view item to be archivable, it and all of its contents must implement `clone()` correctly. This implementation is not an issue for standard framework components, but you must be careful with subclasses of them.

See Also

`view` (page 1717)

`setMaxSize` (page 1714)
`setMinSize` (page 1714)

setVisibilityPriority

Sets the receiver's visibility priority to *visibilityPriority*.

```
public void setVisibiltyPriority(int visibilityPriority)
```

Discussion
The values for *visibilityPriority* are described in "Constants" (page 1717).

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

See Also
`visibilityPriority` (page 1717)

tag

Returns the receiver's tag, which can be used for your own custom purpose.

```
public int tag()
```

See Also
`setTag` (page 1715)

target

Returns the receiver's target.

```
public Object target()
```

See Also
`setTarget` (page 1715)

toolbar

Returns the toolbar that is using the receiver.

```
public NSToolbar toolbar()
```

toolTip

Returns the tooltip used when the receiver is displayed in the toolbar.

```
public String toolTip()
```

See Also
`setToolTip` (page 1715)

validate

This method is called by the receiver's toolbar during validation.

```
public void validate()
```

Discussion
You use this method by overriding it in a subclass; typically, you wouldn't call it. For further discussion, see "Validating Toolbar Items".

See Also
`setEnabled` (page 1713)

view

Returns the receiver's view.

```
public NSView view()
```

Discussion
Note that many of the set/get methods are implemented by calls forwarded to the NSView object referenced by this attribute, if the object responds to it.

See Also
`setView` (page 1715)

visibilityPriority

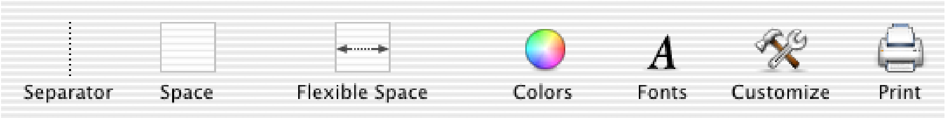
Returns the receiver's visibility priority.

```
public int visibilityPriority()
```

See Also
`setVisibilityPriority` (page 1716)

Constants

NSToolbarItem defines the following standard toolbar item identifiers, which are described in the order they are shown in the following figure:



Constant	Description
SeparatorItemIdentifier	The Separator item.
SpaceItemIdentifier	The Space item.
FlexibleSpaceItemIdentifier	The Flexible Space item.
ShowColorsItemIdentifier	The Colors item. Shows the color panel.
ShowFontsItemIdentifier	The Fonts item. Shows the font panel.
CustomizeToolbarItemIdentifier	The Customize item. Shows the customization palette.
PrintItemIdentifier	The Print item. Sends printDocument to firstResponder, but you can change this in toolbarWillAddItem (page 1705) if you need to do so.

When a toolbar does not have enough space to fit all its items, it must push some items into the overflow menu. These values allow you to suggest a priority for a toolbar item. To suggest that an item always remain visible, give it a value greater than NSToolbarItemVisibilityPriorityStandard, but less than NSToolbarItemVisibilityPriorityUser. In configurable toolbars, users can control the priority of an item and the priority is autosaved by the NSToolbar. These values are used by the setVisibilityPriority (page 1716) and visibilityPriority (page 1717) methods:

Constant	Description
VisibilityPriorityStandard	The default visibility priority. Available in Mac OS X v10.4 and later.
VisibilityPriorityLow	Items with this priority will be the first items to be pushed to the overflow menu. Available in Mac OS X v10.4 and later.
VisibilityPriorityHigh	Items with this priority are less inclined to be pushed to the overflow menu. Available in Mac OS X v10.4 and later.
VisibilityPriorityUser	Items with this priority are the last to be pushed to the overflow menu. Only the user should set items to this priority. Available in Mac OS X v10.4 and later.

NSUserDefaultsController

Inherits from	NSController : NSObject
Implements	NSCoding (NSController)
Package:	com.apple.cocoa.application
Availability	Available in Mac OS X v10.3 and later.
Companion guide	Cocoa Bindings Programming Topics

Overview

NSUserDefaultsController is a Cocoa bindings compatible controller class. Properties of the shared instance of this class can be bound to user interface elements to access and modify values stored in NSUserDefaults.

Tasks

Constructors

`NSUserDefaultsController` (page 1720)
Creates and returns an empty NSUserDefaultsController.

Obtaining the Shared Instance

`sharedUserDefaultsController` (page 1720)
Returns the shared instance of NSUserDefaultsController, creating it if necessary.

Managing User Defaults Values

`defaults` (page 1721)
Returns the instance of NSUserDefaults in use by the receiver.

`setInitialValues` (page 1723)
Sets the receiver's initial values to *`initialValues`*.

`hasUnappliedChanges` (page 1721)
Returns whether the receiver has user default values that have not been saved to NSUserDefaults.

`initialValues` (page 1722)
: Returns a dictionary containing the receiver's initial default values.

`setAppliesImmediately` (page 1723)
: Sets whether any changes made to the receiver's user default properties are saved immediately.

`appliesImmediately` (page 1721)
: Returns whether any changes made to bound user default properties are saved immediately.

`revert` (page 1722)
: Causes the receiver to discard any unsaved changes to bound user default properties, restoring their previous values.

`revertToInitialValues` (page 1722)
: Causes the receiver to discard all edits and replace the values of all the user default properties with any corresponding values in the `initialValues` (page 1722) dictionary.

`save` (page 1723)
: Saves the values of the receiver's user default properties.

Constructors

NSUserDefaultsController

Creates and returns an empty NSUserDefaultsController.

```
public NSUserDefaultsController()
```

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

Creates and returns an NSUserDefaultsController using the NSUserDefaults instance specified in *defaults* and the initial default values contained in the *initialValues* dictionary.

```
public NSUserDefaultsController(NSUserDefaults defaults, NSDictionary initialValues)
```

Discussion
If *defaults* is `null`, the receiver uses `NSUserDefaults.standardUserDefaults()`.

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

Static Methods

sharedUserDefaultsController

Returns the shared instance of NSUserDefaultsController, creating it if necessary.

```
public static Object sharedUserDefaultsController()
```

Discussion
This instance has no initial values, and uses `NSUserDefaults.standardUserDefaults()` to create the defaults. An application can get this object when an application launches and configure it as required.

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

Instance Methods

appliesImmediately

Returns whether any changes made to bound user default properties are saved immediately.

```
public boolean appliesImmediately()
```

Discussion
Default is `true`.

This property is observable using key-value observing.

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

See Also
`setAppliesImmediately` (page 1723)

defaults

Returns the instance of NSUserDefaults in use by the receiver.

```
public NSUserDefaults defaults()
```

Discussion
This property is observable using key-value observing.

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

hasUnappliedChanges

Returns whether the receiver has user default values that have not been saved to NSUserDefaults.

```
public boolean hasUnappliedChanges()
```

Discussion
This property is observable using key-value observing.

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

See Also

`appliesImmediately` (page 1721)

`setAppliesImmediately` (page 1723)

initialValues

Returns a dictionary containing the receiver's initial default values.

```
public NSDictionary initialValues()
```

Discussion

These values are used when is no value found for the bound property in `defaults` (page 1721).

This property is observable using key-value observing.

Availability

Available in Mac OS X v10.3 and later.

See Also

`setInitialValues` (page 1723)

`revertToInitialValues` (page 1722)

revert

Causes the receiver to discard any unsaved changes to bound user default properties, restoring their previous values.

```
public void revert(Object sender)
```

Discussion

The *sender* is typically the object that invoked this method.

If `appliesImmediately` (page 1721) is `true`, this method only causes any bound editors with uncommitted changes to discard their edits.

Availability

Available in Mac OS X v10.3 and later.

See Also

`revertToInitialValues` (page 1722)

revertToInitialValues

Causes the receiver to discard all edits and replace the values of all the user default properties with any corresponding values in the `initialValues` (page 1722) dictionary.

```
public void revertToInitialValues(Object sender)
```

Discussion

This effectively sets the preferences that a user can change to their "out-of-the-box" values. This method has no effect if initial values were not specified. The *sender* is typically the object that invoked this method.

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

See Also
`initialValues` (page 1722)
`revert` (page 1722)

save

Saves the values of the receiver's user default properties.

```
public void save(Object sender)
```

Discussion
This method has no effect if `appliesImmediately` (page 1721) returns `true`.

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

setAppliesImmediately

Sets whether any changes made to the receiver's user default properties are saved immediately.

```
public void setAppliesImmediately(boolean flag)
```

Discussion
The default is `true`.

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

See Also
`appliesImmediately` (page 1721)

setInitialValues

Sets the receiver's initial values to *initialValues*.

```
public void setInitialValues(NSDictionary initialValues)
```

Discussion
These values are used when a user default properties has no value in NSUserDefaults and by `revertToInitialValues` (page 1722).

The initial values must be set before loading a nib that uses the receiver, as those values may be referenced at load time. It is good practice to set the initial values–along with registering any defaults for the applications–in the initializing method of your preference dialog controller, or the application delegate.

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

See Also

`defaults` (page 1721)

`initialValues` (page 1722)

NSView

Inherits from	NSResponder : NSObject
Implements	NSCoding (NSResponder)
Package:	com.apple.cocoa.application
Companion guides	Drawing and Views Programming Topics for Cocoa Cocoa Event-Handling Guide Drag and Drop Programming Topics for Cocoa Printing Programming Topics for Cocoa

Class at a Glance

NSView is an abstract class that defines the basic drawing, event-handling, and printing architecture of an application. You typically don't interact with the NSView API directly; rather, your custom view classes inherit from NSView and override many of its methods, which are invoked automatically by the Application Kit. If you're not creating a custom view class, there are few methods you need to use.

Principal Attributes

- Event handling
- Integrated display to screen and printer
- Flexible coordinate systems
- Icon dragging

Commonly Used Methods

`frame` (page 1754)
Returns the NSView's location and size.

`bounds` (page 1743)
Returns the NSView's internal origin and size.

`setNeedsDisplay` (page 1779)
Marks the NSView as needing to be redrawn.

`window` (page 1787)
Returns the NSWindow that contains the NSView.

drawRect (page 1753)
Draws the NSView. (All subclasses must implement this method, but it's rarely invoked explicitly.)

Overview

NSView is an abstract class that provides concrete subclasses with a structure for drawing, printing, and handling events. NSViews are arranged within an NSWindow, in a nested hierarchy of subviews. A view object claims a rectangular region of its enclosing superview, is responsible for all drawing within that region, and is eligible to receive mouse events occurring in it as well. In addition to these major responsibilities, NSView handles dragging of icons and works with the NSScrollView class to support efficient scrolling.

Most of the functionality of NSView either is automatically invoked by the Application Kit, or is available in Interface Builder. Unless you're implementing a concrete subclass of NSView or working intimately with the content of the view hierarchy at runtime, you don't need to know much about this class's interface. See "Commonly Used Methods" (page 1725) for methods you might use regardless.

For more information on how NSViews handle event and action messages, see "Handling Events in Views", "Handling Mouse Events in Views", and "Handling Tracking-Rectangle and Cursor-Update Events in Views". For more information on displaying tooltips and contextual menus, see "How Contextual Menus Work" and "Tooltips".

Subclassing Notes

NSView is perhaps the most important class in the Application Kit when it comes to subclassing and inheritance. Most user-interface objects you see in a Cocoa application are objects that inherit from NSView. If you want to create an object that draws itself in a special way, or that responds to mouse clicks in a special way, you would create a custom subclass of NSView (or of a class that inherits from NSView). Subclassing NSView is such a common and important procedure that several technical documents describe how to both draw in custom subclasses and respond to events in custom subclasses. See *Drawing and Views Programming Topics for Cocoa* (especially "Drawing in a View") and *Cocoa Event-Handling Guide* (especially "Handing Events in View" and "Handling Mouse Events in Views").

Tasks

Constructors

NSView (page 1737)
Creates an NSView with a zero-sized frame rectangle.

Managing the View Hierarchy

superview (page 1782)
Returns the receiver's superview, or `null` if it has none.

`subviews` (page 1782)
: Return the receiver's immediate subviews.

`window` (page 1787)
: Returns the receiver's window object, or `null` if it has none.

`addSubview` (page 1739)
: Inserts *`aView`* among the receiver's subviews so it's displayed immediately above or below *`otherView`* according to whether *`place`* is `NSWindow.Above` or `NSWindow.Below`.

`didAddSubview` (page 1747)
: Overridden by subclasses to perform additional actions when subviews are added to the receiver.

`removeFromSuperview` (page 1768)
: Unlinks the receiver from its superview and its NSWindow, removes it from the responder chain, and invalidates its cursor rectangles.

`removeFromSuperviewWithoutNeedingDisplay` (page 1769)
: Unlinks the receiver from its superview and its NSWindow, removes it from the responder chain, but does not invalidate its cursor rectangles to cause redrawing.

`replaceSubview` (page 1769)
: Replaces *`oldView`* with *`newView`* in the receiver's subviews.

`isDescendantOf` (page 1756)
: Returns `true` if the receiver is a subview, immediate or not, of *`aView`*, or if it's identical to *`aView`*; otherwise returns `false`.

`opaqueAncestor` (page 1762)
: Returns the receiver's closest opaque ancestor (including the receiver itself).

`ancestorSharedWithView` (page 1742)
: Returns the closest ancestor shared by the receiver and *`aView`*, or `null` if there's no such object.

`viewDidMoveToSuperview` (page 1784)
: Informs the receiver that its superview has changed (possibly to `null`).

`viewDidMoveToWindow` (page 1785)
: Informs the receiver that it has been added to a new view hierarchy.

`viewWillMoveToSuperview` (page 1785)
: Informs the receiver that its superview is about to change to *`newSuperview`* (which may be `null`).

`viewWillMoveToWindow` (page 1785)
: Informs the receiver that it's being added to the view hierarchy of *`newWindow`* (which may be `null`).

`willRemoveSubview` (page 1787)
: Overridden by subclasses to perform additional actions before subviews are removed from the receiver.

Searching by Tag

`viewWithTag` (page 1786)
: Returns the receiver's nearest descendant (including itself) whose tag is *`aTag`*, or `null` if no subview has that tag.

`tag` (page 1782)
: Returns the receiver's tag, an integer that you can use to identify view objects in your application.

Modifying the Frame Rectangle

`setFrame` (page 1776)
Sets the receiver's frame rectangle to *`frameRect`*, thereby repositioning and resizing it within the coordinate system of its superview.

`frame` (page 1754)
Returns the receiver's frame rectangle, which defines its position in its superview.

`setFrameOrigin` (page 1777)
Sets the origin of the receiver's frame rectangle to *`newOrigin`*, effectively repositioning it within its superview.

`setFrameSize` (page 1777)
Sets the size of the receiver's frame rectangle to *`newSize`*, resizing it within its superview without affecting its coordinate system.

`setFrameRotation` (page 1777)
Sets the rotation of the receiver's frame rectangle to *`angle`* degrees, rotating it within its superview without affecting its coordinate system.

`frameRotation` (page 1755)
Returns the angle, in degrees, of the receiver's frame relative to its superview's coordinate system.

Modifying the Bounds Rectangle

`setBounds` (page 1773)
Sets the receiver's bounds rectangle to *`boundsRect`*.

`bounds` (page 1743)
Returns the receiver's bounds rectangle, which expresses its location and size in its own coordinate system.

`setBoundsOrigin` (page 1774)
Sets the origin of the receiver's bounds rectangle to *`newOrigin`*, effectively shifting its coordinate system so *`newOrigin`* lies at the origin of the receiver's frame rectangle.

`setBoundsSize` (page 1775)
Sets the size of the receiver's bounds rectangle to *`newSize`*, inversely scaling its coordinate system relative to its frame rectangle.

`setBoundsRotation` (page 1775)
Sets the rotation of the receiver's bounds rectangle to *`angle`* degrees.

`boundsRotation` (page 1743)
Returns the angle, in degrees, of the receiver's bounds rectangle relative to its frame rectangle.

Modifying the Coordinate System

`translateOriginToPoint` (page 1783)
Translates the receiver's coordinate system so that its origin moves to *`newOrigin`*.

`scaleUnitSquareToSize` (page 1771)
Scales the receiver's coordinate system so that the unit square scales to *`newUnitSize`*.

`rotateByAngle` (page 1770)
: Rotates the receiver's bounds rectangle by *`angle`* degrees around the origin of the coordinate system, (0.0, 0.0).

Examining Coordinate System Modifications

`isFlipped` (page 1756)
: Returns `true` if the receiver uses flipped drawing coordinates or `false` if it uses native coordinates.

`isRotatedFromBase` (page 1758)
: Returns `true` if the receiver or any of its ancestors has ever received a `setFrameRotation` (page 1777) or `setBoundsRotation` (page 1775) message; otherwise returns `false`.

`isRotatedOrScaledFromBase` (page 1758)
: Returns `true` if the receiver or any of its ancestors has ever had a nonzero frame or bounds rotation, or has been scaled from the window's base coordinate system; otherwise returns `false`.

Converting Coordinates

`convertPointFromView` (page 1744)
: Converts *`aPoint`* from the coordinate system of *`aView`* to that of the receiver.

`convertPointToView` (page 1745)
: Converts *`aPoint`* from the receiver's coordinate system to that of *`aView`*.

`convertSizeFromView` (page 1746)
: Converts *`aSize`* from *`aView`*'s coordinate system to that of the receiver.

`convertSizeToView` (page 1746)
: Converts *`aSize`* from the receiver's coordinate system to that of *`aView`*.

`convertRectFromView` (page 1745)
: Converts *`aRect`* from the coordinate system of *`aView`* to that of the receiver.

`convertRectToView` (page 1745)
: Converts *`aRect`* from the receiver's coordinate system to that of *`aView`*.

`centerScanRect` (page 1744)
: Converts the corners of *`aRect`* to lie on the center of device pixels, which is useful in compensating for rendering overscanning when the coordinate system has been scaled.

Controlling Notifications

`setPostsFrameChangedNotifications` (page 1780)
: Controls whether the receiver informs observers when its frame rectangle changes.

`postsFrameChangedNotifications` (page 1763)
: Returns `true` if the receiver posts notifications to the default notification center whenever its frame rectangle changes; returns `false` otherwise.

`setPostsBoundsChangedNotifications` (page 1779)
: Controls whether the receiver informs observers when its bounds rectangle changes.

postsBoundsChangedNotifications (page 1763)
Returns true if the receiver posts notifications to the default notification center whenever its bounds rectangle changes; returns false otherwise.

Resizing Subviews

resizeSubviewsWithOldSize (page 1770)
Informs the receivers's subviews that the receiver's bounds rectangle size has changed from *oldBoundsSize*.

resizeWithOldSuperviewSize (page 1770)
Informs the receiver that the bounds size of its superview has changed from *oldBoundsSize*.

setAutoresizesSubviews (page 1773)
Determines whether the receiver automatically resizes its subviews when its frame size changes.

autoresizesSubviews (page 1742)
Returns true if the receiver automatically resizes its subviews using resizeSubviewsWithOldSize (page 1770) whenever its frame size changes, false otherwise.

setAutoresizingMask (page 1773)
Determines how the receiver's resizeWithOldSuperviewSize (page 1770) method changes its frame rectangle.

autoresizingMask (page 1742)
Returns the receiver's autoresizing mask, which determines how it's resized by the resizeWithOldSuperviewSize (page 1770) method.

Focusing

lockFocus (page 1759)
Locks the focus on the receiver, so subsequent commands take effect in the receiver's window and coordinate system.

unlockFocus (page 1783)
Balances an earlier lockFocus (page 1759) message; restoring the focus to the previously focused view is necessary.

focusView (page 1738)
Returns the currently focused NSView object, or null if there is none.

Displaying

setNeedsDisplay (page 1779)
If *flag* is true, marks the receiver's entire bounds as needing display; if *flag* is false, marks it as not needing display.

needsDisplay (page 1760)
Returns true if the receiver needs to be displayed, as indicated using setNeedsDisplay (page 1779); returns false otherwise.

display (page 1747)
Displays the receiver and all its subviews if possible, invoking each NSView's lockFocus (page 1759), drawRect (page 1753), and unlockFocus (page 1783) methods as necessary.

displayRect (page 1749)
Acts as display (page 1747), confining drawing to *aRect*.

displayRectIgnoringOpacity (page 1749)
Acts as display (page 1747), but confining drawing to *aRect* and not backing up to the first opaque ancestor—it simply causes the receiver and its descendants to execute their drawing code.

displayIfNeeded (page 1748)
Displays the receiver and all its subviews if any part of the receiver has been marked as needing display with a setNeedsDisplay (page 1779)message.

displayIfNeededInRect (page 1748)
Acts as displayIfNeeded (page 1748), confining drawing to *aRect*.

displayIfNeededIgnoringOpacity (page 1748)
Acts as displayIfNeeded (page 1748), except that this method doesn't back up to the first opaque ancestor—it simply causes the receiver and its descendants to execute their drawing code.

displayIfNeededInRectIgnoringOpacity (page 1748)
Acts as displayIfNeeded (page 1748), but confining drawing to *aRect* and not backing up to the first opaque ancestor—it simply causes the receiver and its descendants to execute their drawing code.

isOpaque (page 1758)
Overridden by subclasses to return true if the receiver is opaque, false otherwise.

setKeyboardFocusRingNeedsDisplayInRect (page 1778)
Invalidates the area around the focus ring.

defaultFocusRingType (page 1737)

setFocusRingType (page 1776)
Sets the type of focus ring to be drawn around the receiver.

focusRingType (page 1754)
Returns the type of focus ring drawn around the receiver.

Hiding Views

setHidden (page 1778)
Sets whether the receiver is hidden.

isHidden (page 1757)
Returns whether the receiver is marked as hidden.

isHiddenOrHasHiddenAncestor (page 1757)
Returns true if the receiver is marked as hidden or has an ancestor in the view hierarchy that is marked as hidden; returns false otherwise.

Drawing

drawRect (page 1753)
Overridden by subclasses to draw the receiver's image within *aRect*.

visibleRect (page 1786)
Returns the portion of the receiver not clipped by its superviews.

`canDraw` (page 1744)
Returns `true` if drawing commands will produce any result, `false` otherwise.

`shouldDrawColor` (page 1781)
Returns `false` if the receiver is being drawn in an NSWindow (as opposed, for example, to being printed) and the NSWindow can't store color; otherwise returns `true`.

`needsToDrawRect` (page 1761)
Returns whether rectangle *`aRect`* intersects any part of the area that the receiver is being asked to draw.

`rectsBeingDrawn` (page 1766)
Returns a list of non-overlapping rectangles that define the area the receiver is being asked to draw in `drawRect` (page 1753).

`wantsDefaultClipping` (page 1787)
Returns whether the Application Kit's default clipping provided to `drawRect` (page 1753) implementations is in effect.

Managing Live Resize

`inLiveResize` (page 1756)
A convenience method, expected to be called from `drawRect` (page 1753) to make decisions about optimized drawing.

`preservesContentDuringLiveResize` (page 1764)
Returns `true` if the view supports the optimization of live resize operations by preserving content that has not moved; otherwise, returns `false`.

`rectsExposedDuringLiveResize` (page 1767)
Returns a list of rectangles indicating the newly exposed areas of the receiver.

`rectPreservedDuringLiveResize` (page 1766)
Returns the rectangle identifying the portion of your view that did not change during a live resize operation.

`viewWillStartLiveResize` (page 1785)
Informs the receiver of the start of a live resize.

`viewDidEndLiveResize` (page 1784)
Informs the receiver of the end of a live resize.

Managing a Graphics State

`allocateGState` (page 1741)
Causes the receiver to maintain a private graphics state object, which encapsulates all parameters of the graphics environment.

`gState` (page 1755)
Returns the identifier for the receiver's graphics state object, or 0 if the receiver doesn't have a graphics state object.

`setUpGState` (page 1781)
Overridden by subclasses to (re)initialize the receiver's graphics state object.

renewGState (page 1769)
Invalidates the receiver's graphics state object, if it has one, so it will be regenerated using setUpGState (page 1781) the next time the receiver is focused for drawing.

releaseGState (page 1768)
Frees the receiver's graphics state object, if it has one.

Event Handling

acceptsFirstMouse (page 1738)
Overridden by subclasses to return true if the receiver should be sent a mouseDown (page 1192) message for *theEvent*, an initial mouse-down event over the receiver in its window, false if not.

hitTest (page 1756)
Returns the farthest descendant of the receiver in the view hierarchy (including itself) that contains *aPoint*, or null if *aPoint* lies completely outside the receiver.

isMouseInRect (page 1757)
Returns true if *aRect* contains *aPoint* (which represents the hot spot of the mouse cursor), accounting for whether the receiver is flipped or not.

performKeyEquivalent (page 1763)
Implemented by subclasses to respond to key equivalents (also known as shortcuts).

performMnemonic (page 1763)
Implemented by subclasses to respond to mnemonics.

mouseDownCanMoveWindow (page 1760)
Returns true if the receiver does not need to handle a mouse down and can pass it through to the view; false if it needs to handle the mouse down.

Dragging Operations

concludeDragOperation (page 1744)

dragImage (page 1751)
Initiates a dragging operation from the receiver, allowing the user to drag arbitrary data with a specified icon into any application that has window or view objects that accept dragged data.

dragFile (page 1749)
Initiates a dragging operation from the receiver, allowing the user to drag a file icon to any application that has window or view objects that accept files.

registerForDraggedTypes (page 1768)
Registers *pboardTypes* as the pasteboard types that the receiver will accept as the destination of an image-dragging session.

draggingEntered (page 1750)

draggingExited (page 1750)
Invoked when the dragged image exits the receiver's bounds rectangle.

draggingUpdated (page 1750)

performDragOperation (page 1762)

prepareForDragOperation (page 1764)

unregisterDraggedTypes (page 1784)
Unregisters the receiver as a possible destination in a dragging session.

shouldDelayWindowOrderingForEvent (page 1781)
Overridden by subclasses to allow the user to drag images from the receiver without its window moving forward and possibly obscuring the destination and without activating the application.

dragPromisedFilesOfTypes (page 1752)
Initiates a dragging operation from the receiver, allowing the user to drag one or more promised files (or directories) into any application that has window or view objects that accept promised file data.

Managing Cursor Rectangles

addCursorRect (page 1739)
Establishes *aCursor* as the cursor to be used when the mouse pointer lies within *aRect*.

removeCursorRect (page 1768)
Completely removes a cursor rectangle from the receiver.

discardCursorRects (page 1747)
Invalidates all cursor rectangles set up using addCursorRect (page 1739).

resetCursorRects (page 1770)
Overridden by subclasses to define their default cursor rectangles.

Managing Tool Tips

setToolTip (page 1780)
Sets the tool tip text for the view to *string*.

toolTip (page 1783)
Returns the text for the view's tool tip.

Managing Tracking Rectangles

addTrackingRect (page 1739)
Establishes *aRect* as an area for tracking mouse-entered and mouse-exited events within the receiver and returns a tag that identifies the tracking rectangle in NSEvent objects and can be used to remove the tracking rectangle.

removeTrackingRect (page 1769)
Removes the tracking rectangle identified by *aTag*, which is the value returned by a previous addTrackingRect (page 1739) message.

Scrolling

`scrollPoint` (page 1772)
Scrolls the receiver's closest ancestor NSClipView so *aPoint* in the receiver lies at the origin of the NSClipView's bounds rectangle.

`scrollRectToVisible` (page 1772)
Scrolls the receiver's closest ancestor NSClipView the minimum distance needed so *aRect* in the receiver becomes visible in the NSClipView.

`autoscroll` (page 1742)
Scrolls the receiver's closest ancestor NSClipView proportionally to the distance of *theEvent* outside of it.

`adjustScroll` (page 1741)
Overridden by subclasses to modify *proposedVisibleRect*, returning the altered rectangle.

`scrollRect` (page 1772)
Copies the visible portion of the receiver's rendered image within *aRect* and lays that portion down again at *offset* from *aRect*'s origin.

`enclosingScrollView` (page 1754)
Returns the nearest ancestor NSScrollView containing the receiver (not including the receiver itself); otherwise returns `null`.

`scrollClipViewToPoint` (page 1771)
Notifies the superview of *aClipView* that *aClipView* needs to set its bounds rectangle origin to *newOrigin*.

`reflectScrolledClipView` (page 1767)
Notifies *aClipView*'s superview that either *aClipView*'s bounds rectangle or the document view's frame rectangle has changed, and that any indicators of the scroll position need to be adjusted.

Context-sensitive Menus

`menuForEvent` (page 1760)
Overridden by subclasses to return a context-sensitive pop-up menu for the mouse-down event *theEvent*.

`defaultMenu` (page 1738)
Overridden by subclasses to return the default pop-up menu for instances of the receiving class.

Managing the Key View Loop

`canBecomeKeyView` (page 1743)
Returns whether the receiver can become key view.

`needsPanelToBecomeKey` (page 1761)
Overridden by subclasses to return `true` if the receiver requires its panel, which might otherwise avoid becoming key, to become the key window so that it can handle keyboard input.

`setNextKeyView` (page 1779)
Inserts *aView* after the receiver in the key view loop of the receiver's NSWindow.

`nextKeyView` (page 1761)
Returns the view object following the receiver in the key view loop, or `null` if there is none.

`nextValidKeyView` (page 1762)
Returns the closest view object in the key view loop that follows the receiver and actually accepts first responder status, or `null` if there is none.

`previousKeyView` (page 1764)
Returns the view object preceding the receiver in the key view loop, or `null` if there is none.

`previousValidKeyView` (page 1765)
Returns the closest view object in the key view loop that precedes the receiver and actually accepts first responder status, or `null` if there is none.

Printing

`print` (page 1765)
This action method opens the Print panel, and if the user chooses an option other than canceling, prints the receiver and all its subviews to the device specified in the Print panel.

`dataWithEPSInsideRect` (page 1746)
Returns EPS data that draws the region of the receiver within *aRect*.

`dataWithPDFInsideRect` (page 1747)
Returns PDF data that draws the region of the receiver within *aRect*.

`writeEPSInsideRectToPasteboard` (page 1788)
Writes EPS data that draws the region of the receiver within *aRect* onto *pboard*.

`writePDFInsideRectToPasteboard` (page 1788)
Writes PDF data that draws the region of the receiver within *aRect* onto *pboard*.

Pagination

`heightAdjustLimit` (page 1755)
Returns the fraction (from 0.0 to 1.0) of the page that can be pushed onto the next page during automatic pagination to prevent items such as lines of text from being divided across pages.

`widthAdjustLimit` (page 1787)
Returns the fraction (from 0.0 to 1.0) of the page that can be pushed onto the next page during automatic pagination to prevent items such as small images or text columns from being divided across pages.

`adjustPageWidth` (page 1740)
Overridden by subclasses to adjust page width during automatic pagination.

`adjustPageHeight` (page 1740)
Overridden by subclasses to adjust page height during automatic pagination.

`knowsPageRange` (page 1759)
Returns `true` if the receiver handles page boundaries.

`rectForPage` (page 1765)
Implemented by subclasses to determine the portion of the receiver to be printed for the page number page.

`locationOfPrintRect` (page 1759)
Invoked by `print` (page 1765) to determine the location of *aRect*, the rectangle being printed on the physical page.

Adorning Pages in Printout

`drawPageBorderWithSize` (page 1753)
: Allows applications that use the Application Kit pagination facility to draw additional marks on each logical page, such as alignment marks or a virtual sheet border of size *borderSize*.

`drawSheetBorderWithSize` (page 1753)
: Allows applications that use the Application Kit pagination facility to draw additional marks on each printed sheet, such as crop marks or fold lines of size *borderSize*.

Writing Conforming Rendering Instructions

`endPage` (page 1754)
: Writes the end of a conforming page.

Constructors

NSView

Creates an NSView with a zero-sized frame rectangle.

```
public NSView()
```

Creates an NSView with *frameRect* as its frame rectangle.

```
public NSView(NSRect frameRect)
```

Discussion
The new view object must be inserted into the view hierarchy of an NSWindow before it can be used.

See Also
`addSubview` (page 1739)
`setFrame` (page 1776)

Static Methods

defaultFocusRingType

```
public static int defaultFocusRingType()
```

Discussion
Returns the default type of focus ring for objects of the receiver's class. If `NSGraphics.FocusRingTypeDefault` is returned from the instance method `focusRingType` (page 1754), the receiver can invoke this class method to find out what type of focus ring is the default. The receiver is free to ignore the default setting.Possible return values are listed in the "Constants" (page 727) section of NSGraphics.

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

defaultMenu

Overridden by subclasses to return the default pop-up menu for instances of the receiving class.

```
public static NSMenu defaultMenu()
```

Discussion
NSView's implementation returns `null`.

See Also
`menuForEvent` (page 1760)
`menu` (page 1192) (NSResponder)

focusView

Returns the currently focused NSView object, or `null` if there is none.

```
public static NSView focusView()
```

See Also
`lockFocus` (page 1759)
`unlockFocus` (page 1783)

Instance Methods

acceptsFirstMouse

Overridden by subclasses to return `true` if the receiver should be sent a `mouseDown` (page 1192) message for *theEvent*, an initial mouse-down event over the receiver in its window, `false` if not.

```
public boolean acceptsFirstMouse(NSEvent theEvent)
```

Discussion
The receiver can either return a value unconditionally, or use the location of *theEvent* to determine whether or not it wants the event. NSView's implementation ignores *theEvent* and returns `false`.

Override this method in a subclass to allow instances to respond to click-through. This allows the user to click on a view in an inactive window, activating the view with one click, instead of clicking first to make the window active and then clicking the view. Most view objects refuse a click-through attempt, so the event simply activates the window. Many control objects, however, such as NSButton and NSSlider, do accept them, so the user can immediately manipulate the control without having to release the mouse button.

See Also
`hitTest` (page 1756)

addCursorRect

Establishes *aCursor* as the cursor to be used when the mouse pointer lies within *aRect*.

```
public void addCursorRect(NSRect aRect, NSCursor aCursor)
```

Discussion

Cursor rectangles aren't subject to clipping by superviews, nor are they intended for use with rotated NSViews. You should explicitly confine a cursor rectangle to the NSView's visible rectangle to prevent improper behavior.

This method is intended to be invoked only by the `resetCursorRects` (page 1770) method. If invoked in any other way, the resulting cursor rectangle will be discarded the next time the NSView's cursor rectangles are rebuilt.

See Also

`removeCursorRect` (page 1768)
`discardCursorRects` (page 1747)
`resetCursorRects` (page 1770)
`visibleRect` (page 1786)

addSubview

Inserts *aView* among the receiver's subviews so it's displayed immediately above or below *otherView* according to whether *place* is `NSWindow.Above` or `NSWindow.Below`.

```
public void addSubview(NSView aView, int place, NSView otherView)
```

Discussion

If *otherView* is `null` (or isn't a subview of the receiver), *aView* is added above or below all of its new siblings. Also sets the receiver as the next responder to *aView*.

Adds *aView* to the receiver's subviews so it's displayed above its siblings.

```
public void addSubview(NSView aView)
```

Discussion

Also sets the receiver as the next responder to *aView*.

See Also

`subviews` (page 1782)
`removeFromSuperview` (page 1768)
`setNextResponder` (page 1198) (NSResponder)
`viewWillMoveToSuperview` (page 1785)
`viewWillMoveToWindow` (page 1785)

addTrackingRect

Establishes *aRect* as an area for tracking mouse-entered and mouse-exited events within the receiver and returns a tag that identifies the tracking rectangle in NSEvent objects and can be used to remove the tracking rectangle.

```
public int addTrackingRect(NSRect aRect, Object userObject, Object userData, boolean
      flag)
```

Discussion
userObject is the object that gets sent the event messages. It can be the receiver itself or some other object (such as an NSCursor or a custom drawing tool object), as long as it responds to both `mouseEntered` (page 1192) and `mouseExited` (page 1193). Your application is responsible for making sure that *userObject* is referenced elsewhere so it doesn't get garbage-collected. *userData* is supplied in the NSEvent object for each tracking event. *flag* determines which event is sent first by indicating where the cursor is assumed to be at the time this method is invoked. If *flag* is `true`, the first event will be generated when the cursor leaves *aRect*; if *flag* is `false` the first event will be generated when the cursor enters it.

Tracking rectangles provide a general mechanism that can be used to trigger actions based on the cursor location (for example, a status bar or hint field that provides information on the item the cursor lies over). To simply change the cursor over a particular area, use `addCursorRect` (page 1739). If you must use tracking rectangles to change the cursor, the NSCursor class specification describes the additional methods that must be invoked to change cursors by using tracking rectangles.

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

See Also
`removeTrackingRect` (page 1769)

adjustPageHeight

Overridden by subclasses to adjust page height during automatic pagination.

```
public float adjustPageHeight(float top, float proposedBottom, float bottomLimit)
```

Discussion
This method is invoked by `print` (page 1765) with *top* and *proposedBottom* set to the top and bottom edges of the pending page rectangle in the receiver's coordinate system. The receiver can raise the bottom edge and return the new value, allowing it to prevent items such as lines of text from being divided across pages. *bottomLimit* is the topmost value the return value can be set to, as calculated using the return value of `heightAdjustLimit` (page 1755). If this limit is exceeded, the pagination mechanism simply uses *bottomLimit* for the bottom edge.

NSView's implementation of this method propagates the message to its subviews, allowing nested views to adjust page height for their drawing as well. An NSButton or other small view, for example, will nudge the bottom edge up if necessary to prevent itself from being cut in two (thereby pushing it onto an adjacent page). Subclasses should invoke `super`'s implementation, if desired, after first making their own adjustments.

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

See Also
`adjustPageWidth` (page 1740)

adjustPageWidth

Overridden by subclasses to adjust page width during automatic pagination.

```
public native float adjustPageWidth(float left, float proposedRight, float
    rightLimit)
```

Discussion

This method is invoked by `print` (page 1765), with *`left`* and *`proposedRight`* set to the side edges of the pending page rectangle in the receiver's coordinate system. The receiver can pull in the right edge and return the new value, allowing it to prevent items such as small images or text columns from being divided across pages. *`rightLimit`* is the leftmost value the return value can be set to, as calculated using the return value of `widthAdjustLimit` (page 1787). If this limit is exceeded, the pagination mechanism simply uses *`rightLimit`* for the right edge.

NSView's implementation of this method propagates the message to its subviews, allowing nested views to adjust page width for their drawing as well. An NSButton or other small view, for example, will nudge the right edge out if necessary to prevent itself from being cut in two (thereby pushing it onto an adjacent page). Subclasses should invoke `super`'s implementation, if desired, after first making their own adjustments.

Availability

Available in Mac OS X v10.3 and later.

See Also

`adjustPageHeight` (page 1740)

adjustScroll

Overridden by subclasses to modify *`proposedVisibleRect`*, returning the altered rectangle.

```
public NSRect adjustScroll(NSRect proposedVisibleRect)
```

Discussion

NSClipView invokes this method to allow its document view to adjust its position during scrolling. For example, a custom view object that displays a table of data can adjust the origin of *`proposedVisibleRect`* so rows or columns aren't cut off by the edge of the enclosing NSClipView. NSView's implementation simply returns *`proposedVisibleRect`*.

NSClipView only invokes this method during automatic or user controlled scrolling. Its `scrollToPoint` (page 347) method doesn't invoke this method, so you can still force a scroll to an arbitrary point.

allocateGState

Causes the receiver to maintain a private graphics state object, which encapsulates all parameters of the graphics environment.

```
public void allocateGState()
```

Discussion

If you do not invoke `allocateGState`, a graphics state object is constructed from scratch each time the NSView is focused.

The receiver builds the graphics state parameters using `setUpGState` (page 1781), then automatically establishes this graphics state each time the focus is locked on it. A graphics state may improve performance for view objects that are focused often and need to set many parameters, but use of standard rendering operators is normally efficient enough.

Because graphics states occupy a fair amount of memory, they can actually degrade performance. Be sure to test application performance with and without the private graphics state before committing to its use.

See Also
`setUpGState` (page 1781)
`gState` (page 1755)
`renewGState` (page 1769)
`releaseGState` (page 1768)
`lockFocus` (page 1759)

ancestorSharedWithView

Returns the closest ancestor shared by the receiver and *aView*, or `null` if there's no such object.

```
public NSView ancestorSharedWithView(NSView aView)
```

Discussion
Returns `this` if *aView* is identical to the receiver.

See Also
`isDescendantOf` (page 1756)

autoresizesSubviews

Returns `true` if the receiver automatically resizes its subviews using `resizeSubviewsWithOldSize` (page 1770) whenever its frame size changes, `false` otherwise.

```
public boolean autoresizesSubviews()
```

See Also
`setAutoresizesSubviews` (page 1773)

autoresizingMask

Returns the receiver's autoresizing mask, which determines how it's resized by the `resizeWithOldSuperviewSize` (page 1770) method.

```
public int autoresizingMask()
```

Discussion
The autoresizing mask values are listed under the `setAutoresizingMask` (page 1773) method description. If the autoresizing mask is equal to `ViewNotSizable` (that is, if none of the options are set), then the receiver doesn't resize at all in `resizeWithOldSuperviewSize` (page 1770).

autoscroll

Scrolls the receiver's closest ancestor NSClipView proportionally to the distance of *theEvent* outside of it.

```
public boolean autoscroll(NSEvent theEvent)
```

Discussion

The location of *theEvent* should be expressed in the window's base coordinate system (which it normally is), not the receiving view object's. Returns `true` if any scrolling is performed; otherwise returns `false`.

View objects that track mouse-dragged events can use this method to scroll automatically when the cursor is dragged outside of the NSClipView. Repeated invocations of this method (with an appropriate delay) result in continual scrolling, even when the mouse doesn't move.

See Also

`autoscroll` (page 344) (NSClipView)
`scrollPoint` (page 1772)
`isDescendantOf` (page 1756)

bounds

Returns the receiver's bounds rectangle, which expresses its location and size in its own coordinate system.

```
public NSRect bounds()
```

Discussion

The bounds rectangle may be rotated; use the `boundsRotation` (page 1743) method to check this.

See Also

`frame` (page 1754)
`setBounds` (page 1773)

boundsRotation

Returns the angle, in degrees, of the receiver's bounds rectangle relative to its frame rectangle.

```
public float boundsRotation()
```

Discussion

See the `setBoundsRotation` (page 1775) method description for more information on bounds rotation.

See Also

`rotateByAngle` (page 1770)
`setBoundsRotation` (page 1775)

canBecomeKeyView

Returns whether the receiver can become key view.

```
public boolean canBecomeKeyView()
```

Availability

Available in Mac OS X v10.3 and later.

canDraw

Returns `true` if drawing commands will produce any result, `false` otherwise.

```
public boolean canDraw()
```

Discussion

Use this method when invoking a draw method directly along with `lockFocus` (page 1759) and `unlockFocus` (page 1783), bypassing the `display...` methods (which test drawing ability and perform locking for you). If this method returns `false`, you shouldn't invoke `lockFocus` (page 1759) or perform any drawing.

An NSView can draw on-screen if it is not hidden, it is attached to a view hierarchy in an NSWindow, and the NSWindow has a corresponding window device. An NSView can draw during printing if it is a descendant of the view being printed.

See Also

`setHidden` (page 1778)

centerScanRect

Converts the corners of *aRect* to lie on the center of device pixels, which is useful in compensating for rendering overscanning when the coordinate system has been scaled.

```
public NSRect centerScanRect(NSRect aRect)
```

Discussion

This method 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 receiver's coordinate system. Returns the adjusted rectangle. Note that this method does not take into account any transformations performed using NSAffineTransform or Quartz 2D routines.

See Also

`isRotatedOrScaledFromBase` (page 1758)

concludeDragOperation

```
public void concludeDragOperation(NSDraggingInfo draggingInfo)
```

Discussion

Invoked when the dragging operation is complete and the previous `performDragOperation` (page 1762) returned `true`. *draggingInfo* contains information about the operation. This method allows you to perform any tidying up that is needed, such as updating the visual representation now the dragged data has been incorporated. This message is the last message sent during a dragging session.

convertPointFromView

Converts *aPoint* from the coordinate system of *aView* to that of the receiver.

```
public NSPoint convertPointFromView(NSPoint aPoint, NSView aView)
```

Discussion

If *aView* is `null`, this method instead converts from window base coordinates. Both *aView* and the receiver must belong to the same NSWindow. Returns the converted point.

See Also

`convertRectFromView` (page 1745)
`convertSizeFromView` (page 1746)
`ancestorSharedWithView` (page 1742)
`contentView` (page 1824) (NSWindow)

convertPointToView

Converts *aPoint* from the receiver's coordinate system to that of *aView*.

```
public NSPoint convertPointToView(NSPoint aPoint, NSView aView)
```

Discussion

If *aView* is `null`, this method instead converts to window base coordinates. Both *aView* and the receiver must belong to the same NSWindow. Returns the converted point.

See Also

`convertRectToView` (page 1745)
`convertSizeToView` (page 1746)
`ancestorSharedWithView` (page 1742)
`contentView` (page 1824) (NSWindow)

convertRectFromView

Converts *aRect* from the coordinate system of *aView* to that of the receiver.

```
public NSRect convertRectFromView(NSRect aRect, NSView aView)
```

Discussion

If *aView* is `null`, this method instead converts from window base coordinates. Both *aView* and the receiver must belong to the same NSWindow. Returns the converted rectangle.

See Also

`convertPointFromView` (page 1744)
`convertSizeFromView` (page 1746)
`ancestorSharedWithView` (page 1742)
`contentView` (page 1824) (NSWindow)

convertRectToView

Converts *aRect* from the receiver's coordinate system to that of *aView*.

```
public NSRect convertRectToView(NSRect aRect, NSView aView)
```

Discussion

If *aView* is `null`, this method instead converts to window base coordinates. Both *aView* and the receiver must belong to the same NSWindow. Returns the converted rectangle.

See Also

`convertPointToView` (page 1745)
`convertSizeToView` (page 1746)
`ancestorSharedWithView` (page 1742)
`contentView` (page 1824) (NSWindow)

convertSizeFromView

Converts *aSize* from *aView*'s coordinate system to that of the receiver.

```
public NSSize convertSizeFromView(NSSize aSize, NSView aView)
```

Discussion

If *aView* is `null`, this method instead converts from window base coordinates. Both *aView* and the receiver must belong to the same NSWindow. Returns the converted size.

See Also

`convertPointFromView` (page 1744)
`convertRectFromView` (page 1745)
`ancestorSharedWithView` (page 1742)
`contentView` (page 1824) (NSWindow)

convertSizeToView

Converts *aSize* from the receiver's coordinate system to that of *aView*.

```
public NSSize convertSizeToView(NSSize aSize, NSView aView)
```

Discussion

If *aView* is `null`, this method instead converts to window base coordinates. Both *aView* and the receiver must belong to the same NSWindow. Returns the converted size.

See Also

`convertPointToView` (page 1745)
`convertRectToView` (page 1745)
`ancestorSharedWithView` (page 1742)
`contentView` (page 1824) (NSWindow)

dataWithEPSInsideRect

Returns EPS data that draws the region of the receiver within *aRect*.

```
public NSData dataWithEPSInsideRect(NSRect aRect)
```

Discussion

This data can be placed on an NSPasteboard, written to a file, or used to create an NSImage object.

See Also

writeEPSInsideRectToPasteboard (page 1788)

dataWithPDFInsideRect

Returns PDF data that draws the region of the receiver within *aRect*.

```
public NSData dataWithPDFInsideRect(NSRect aRect)
```

Discussion

This data can be placed on an NSPasteboard, written to a file, or used to create an NSImage object.

See Also

writePDFInsideRectToPasteboard (page 1788)

didAddSubview

Overridden by subclasses to perform additional actions when subviews are added to the receiver.

```
public void didAddSubview(NSView subview)
```

Discussion

Invoked by addSubview (page 1739).

discardCursorRects

Invalidates all cursor rectangles set up using addCursorRect (page 1739).

```
public void discardCursorRects()
```

Discussion

You need never invoke this method directly; it's invoked automatically before the NSView's cursor rectangles are reestablished using resetCursorRects (page 1770).

See Also

discardCursorRects (page 1828) (NSWindow)

display

Displays the receiver and all its subviews if possible, invoking each NSView's lockFocus (page 1759), drawRect (page 1753), and unlockFocus (page 1783) methods as necessary.

```
public void display()
```

Discussion

If the receiver isn't opaque, this method backs up the view hierarchy to the first opaque ancestor, calculates the portion of the opaque ancestor covered by the receiver, and begins displaying from there.

See Also

canDraw (page 1744)

opaqueAncestor (page 1762)

visibleRect (page 1786)
displayIfNeededIgnoringOpacity (page 1748)

displayIfNeeded

Displays the receiver and all its subviews if any part of the receiver has been marked as needing display with a setNeedsDisplay (page 1779)message.

```
public void displayIfNeeded()
```

Discussion

This method invokes NSView's lockFocus (page 1759), drawRect (page 1753), and unlockFocus (page 1783) methods as necessary. If the receiver isn't opaque, this method backs up the view hierarchy to the first opaque ancestor, calculates the portion of the opaque ancestor covered by the receiver, and begins displaying from there.

See Also

display (page 1747)
needsDisplay (page 1760)
displayIfNeededIgnoringOpacity (page 1748)

displayIfNeededIgnoringOpacity

Acts as displayIfNeeded (page 1748), except that this method doesn't back up to the first opaque ancestor—it simply causes the receiver and its descendants to execute their drawing code.

```
public void displayIfNeededIgnoringOpacity()
```

displayIfNeededInRect

Acts as displayIfNeeded (page 1748), confining drawing to *aRect*.

```
public void displayIfNeededInRect(NSRect aRect)
```

Discussion

The argument you pass for *aRect* should be specified in the coordinate system of the receiver.

displayIfNeededInRectIgnoringOpacity

Acts as displayIfNeeded (page 1748), but confining drawing to *aRect* and not backing up to the first opaque ancestor—it simply causes the receiver and its descendants to execute their drawing code.

```
public void displayIfNeededInRectIgnoringOpacity(NSRect aRect)
```

Discussion

The argument you pass for *aRect* should be specified in the coordinate system of the receiver.

displayRect

Acts as `display` (page 1747), confining drawing to *`aRect`*.

```
public void displayRect(NSRect aRect)
```

Discussion
The argument you pass for *`aRect`* should be specified in the coordinate system of the receiver.

displayRectIgnoringOpacity

Acts as `display` (page 1747), but confining drawing to *`aRect`* and not backing up to the first opaque ancestor—it simply causes the receiver and its descendants to execute their drawing code.

```
public void displayRectIgnoringOpacity(NSRect aRect)
```

Discussion
The argument you pass for *`aRect`* should be specified in the coordinate system of the receiver.

dragFile

Initiates a dragging operation from the receiver, allowing the user to drag a file icon to any application that has window or view objects that accept files.

```
public boolean dragFile(String fullPath, NSRect aRect, boolean slideBack, NSEvent
     theEvent)
```

Discussion
This method must be invoked only within an implementation of the `mouseDown` (page 1192) method. Returns `true` if the receiver successfully initiates the dragging operation (which doesn't necessarily mean the dragging operation concluded successfully). Otherwise returns `false`.

The dragging operation uses these arguments:

- The *`fullPath`* argument is the absolute path for the file to be dragged.
- The *`aRect`* argument describes the position of the icon in the receiver's coordinate system.
- The *`slideBack`* argument indicates whether the icon being dragged should slide back to its position in the receiver if the file isn't accepted. The icon slides back to *`aRect`* if *`slideBack`* is `true`, the file is not accepted by the dragging destination, and the user has not disabled icon animation; otherwise it simply disappears.
- The *`theEvent`* argument is the mouse-down event object from which to initiate the drag operation. In particular, its mouse location is used for the offset of the icon being dragged.

See the NSDraggingSource (page 1965), NSDraggingInfo (page 1959), and NSDraggingDestination (page 1955) interface specifications for more information on dragging operations.

See Also
`dragImage` (page 1751)
`shouldDelayWindowOrderingForEvent` (page 1781)

draggingEntered

```
public int draggingEntered(NSDraggingInfo draggingInfo)
```

Discussion
Invoked when a dragged image enters the receiver. Specifically, this method is invoked when the mouse pointer enters the receiver's bounds rectangle. *draggingInfo* contains information about the dragging operation.

Returns a value indicating which dragging operation will be performed when the image is released. In deciding which dragging operation to return, you should evaluate the overlap between both the dragging operations allowed by the source and the dragging operations and pasteboard data types the receiver supports. The returned value should be one of the following:

Option	Meaning
NSDraggingInfo.DragOperationCopy	The data represented by the image will be copied.
NSDraggingInfo.DragOperationLink	The data will be shared.
NSDraggingInfo.DragOperationGeneric	The operation will be defined by the destination.
NSDraggingInfo.DragOperationPrivate	The operation is negotiated privately between the source and the destination.
NSDraggingInfo.DragOperationEvery	Combines all the above.

If none of the operations is appropriate, returns NSDraggingInfo.DragOperationNone.

See Also
draggingUpdated (page 1750)
draggingExited (page 1750)

draggingExited

Invoked when the dragged image exits the receiver's bounds rectangle.

```
public int draggingExited(NSDraggingInfo draggingInfo)
```

Discussion
draggingInfo contains information about the dragging operation.

draggingUpdated

```
public int draggingUpdated(NSDraggingInfo draggingInfo)
```

Discussion
Invoked periodically as the dragged image is held within the receiver. The messages continue until the image is either released or dragged out of the receiver. *draggingInfo* contains information about the dragging operation. Returns one of the dragging operation options listed under draggingEntered (page 1750).

This method provides you with an opportunity to modify the dragging operation depending on the position of the mouse pointer inside of the receiver. For example, you may have several graphics or areas of text contained within the same view and wish to tailor the dragging operation, or to ignore the drag event completely, depending upon which object is underneath the mouse pointer at the time when the user releases the dragged image and `performDragOperation` (page 1762) is invoked.

You typically examine the contents of the pasteboard in `draggingEntered` (page 1750), as this method is invoked only once, rather than in this method, which is invoked multiple times.

Only one view at a time receives a sequence of `draggingUpdated` messages. If the mouse pointer is within the bounds of two overlapping views that are both valid destinations, the uppermost view receives these messages until the image is either released or dragged out.

See Also

`draggingExited` (page 1750)

`prepareForDragOperation` (page 1764)

dragImage

Initiates a dragging operation from the receiver, allowing the user to drag arbitrary data with a specified icon into any application that has window or view objects that accept dragged data.

```
public void dragImage(NSImage anImage, NSPoint imageLoc, NSSize mouseOffset, NSEvent
     theEvent, NSPasteboard pboard, Object sourceObject, boolean slideBack)
```

Discussion

This method must be invoked only within an implementation of the `mouseDown` (page 1192) or `mouseDragged` (page 1192) methods.

The dragging operation uses these arguments:

- The *anImage* argument is the NSImage to be dragged.
- The *imageLoc* argument is the location of the image's lower-left corner, in the receiver's coordinate system. It determines the placement of the dragged image under the cursor.
- The *mouseOffset* argument is the mouse's current location relative to the mouse-down location. In Mac OS X v10.4 and later, this parameter is ignored. In earlier versions of Mac OS X, this parameter is ignored when positioning the dragged image on screen but not ignored when passing the drag location to the source in `draggedImage:endedAt:operation:`.

  The argument determines the initial location of the image when dragging commences. If you initiate a dragging operation immediately on a mouse-down event, this location should be (0.0, 0.0). If you test for a mouse-dragged event first, this location should be the difference between the mouse-dragged event's location and that of the mouse-down event.
- The *theEvent* argument is the left mouse-down event that triggered the dragging operation (see below).
- The *pboard* argument holds the data to be transferred to the destination (see below).
- The *sourceObject* argument serves as the controller of the dragging operation. It must conform to the NSDraggingSource interface and is typically the receiver itself or its NSWindow.
- The *slideBack* argument determines whether the NSImage should slide back if it's rejected. The image slides back to *aPoint* if *slideBack* is `true`, the image isn't accepted by the dragging destination, and the user hasn't disabled icon animation; otherwise it simply disappears.

Before invoking this method, you must place the data to be transferred on *pboard*. To do this, get the drag pasteboard object (`NSPasteboard.DragPboard`), declare the types of the data, and then put the data on the pasteboard.

See the NSDraggingSource (page 1965), NSDraggingInfo (page 1959), and NSDraggingDestination (page 1955) interface specifications for more information on dragging operations.

See Also

`dragFile` (page 1749)

`shouldDelayWindowOrderingForEvent` (page 1781)

dragPromisedFilesOfTypes

Initiates a dragging operation from the receiver, allowing the user to drag one or more promised files (or directories) into any application that has window or view objects that accept promised file data.

```
public boolean dragPromisedFilesOfTypes(NSArray typeArray, NSRect aRect, Object
    sourceObject, boolean slideBack, NSEvent theEvent)
```

Discussion

Returns `true` if the drag operation is initiated successfully. This method must be invoked only within an implementation of the `mouseDown` (page 1192) method. As part of its implementation, this method invokes `dragImage` (page 1751).

Promised files are files that do not exist, yet, but that the drag source, *sourceObject*, promises to create at a file system location specified by the drag destination when the drag is successfully dropped.The dragging operation uses these arguments:

- The *typeArray* argument is the list of file types being promised. The array elements can consist of file extensions and HFS types encoded with the NSHFSFileTypes method `fileTypeForHFSTypeCode`. If promising a directory of files, only include the top directory in the array.
- The *aRect* argument describes the position of the icon in the receiver's coordinate system.
- The *sourceObject* argument serves as the controller of the dragging operation. It must conform to the NSDraggingSource interface, and is typically the receiver itself or its NSWindow.
- The *slideBack* argument indicates whether the icon being dragged should slide back to its position in the receiver if the file isn't accepted. The icon slides back to *aRect* if *slideBack* is `true`, the promised files are not accepted by the dragging destination, and the user has not disabled icon animation; otherwise it simply disappears.
- The *theEvent* argument is the mouse-down event object from which to initiate the drag operation. In particular, its mouse location is used for the offset of the icon being dragged.

See "Drag and Drop" for more information on dragging operations.

Availability

Available in Mac OS X v10.2 and later.

See Also

`dragImage` (page 1751)

`shouldDelayWindowOrderingForEvent` (page 1781)

drawPageBorderWithSize

Allows applications that use the Application Kit pagination facility to draw additional marks on each logical page, such as alignment marks or a virtual sheet border of size *borderSize*.

```
public void drawPageBorderWithSize(NSSize borderSize)
```

Discussion
The default implementation doesn't draw anything.

See Also
drawSheetBorderWithSize (page 1753)

drawRect

Overridden by subclasses to draw the receiver's image within *aRect*.

```
public void drawRect(NSRect aRect)
```

Discussion
The receiver can assume the focus has been locked and the coordinate transformations of its frame and bounds rectangles have been applied; all it needs to do is invoke rendering client functions. *aRect* is a rectangle defining the area that the receiver is being asked to draw. On Mac OS X version 10.2 and earlier, the Application Kit automatically clips any drawing you perform in this method to this rectangle. On Mac OS X version 10.3 and later, the Application Kit automatically clips drawing to a list of non-overlapping rectangles that more rigorously specify the area needing drawing. You can invoke the rectsBeingDrawn (page 1766) method to retrieve this list of rectangles and use them to constrain your drawing more tightly, if you wish. Moreover, the needsToDrawRect (page 1761) method gives you a convenient way to test individual objects for intersection with the rectangles in the list. See "Drawing in a View" for information and references on drawing.

The default NSView implementation does nothing. If your custom view is a direct NSView subclass you do not need to call super's implementation. Note that it is the responsibility of each subclass to totally fill *aRect* if its superclass' implementation actually draws and returns false from isOpaque (page 1758).

See Also
display (page 1747)
rectsBeingDrawn (page 1766)
isFlipped (page 1756)
needsToDrawRect (page 1761)
shouldDrawColor (page 1781)

drawSheetBorderWithSize

Allows applications that use the Application Kit pagination facility to draw additional marks on each printed sheet, such as crop marks or fold lines of size *borderSize*.

```
public void drawSheetBorderWithSize(NSSize borderSize)
```

Discussion
This method has been deprecated.

See Also

drawPageBorderWithSize (page 1753)

enclosingScrollView

Returns the nearest ancestor NSScrollView containing the receiver (not including the receiver itself); otherwise returns null.

```
public NSScrollView enclosingScrollView()
```

endPage

Writes the end of a conforming page.

```
public void endPage()
```

Discussion

This method is invoked after each page is printed. It invokes unlockFocus (page 1783). This method also generates comments for the bounding box and page fonts, if they were specified as being at the end of the page.

focusRingType

Returns the type of focus ring drawn around the receiver.

```
public native int focusRingType()
```

Discussion

Possible values are listed in the "Constants" (page 727) section of NSGraphics.You can disable a view's drawing of its focus ring by overriding this method to return NSGraphics.FocusRingTypeNone, or by invoking setFocusRingType (page 1776) with and argument of NSGraphics.FocusRingTypeNone. You should only disable the default drawing of a view's focus ring if you want it to draw its own focus ring (for example, setting the background color of the view), or if the view does not have sufficient space to display a focus ring.

Availability

Available in Mac OS X v10.3 and later.

See Also

setFocusRingType (page 1776)

frame

Returns the receiver's frame rectangle, which defines its position in its superview.

```
public NSRect frame()
```

Discussion

The frame rectangle may be rotated; use the frameRotation (page 1755) method to check this.

See Also

bounds (page 1743)

setFrame (page 1776)

frameRotation

Returns the angle, in degrees, of the receiver's frame relative to its superview's coordinate system.

```
public float frameRotation()
```

See Also

setFrameRotation (page 1777)

boundsRotation (page 1743)

gState

Returns the identifier for the receiver's graphics state object, or 0 if the receiver doesn't have a graphics state object.

```
public int gState()
```

Discussion

A view object's graphics state object is recreated from scratch whenever the view is focused, unless the allocateGState (page 1741) method has been invoked. So if the receiver hasn't been focused or hasn't received the allocateGState (page 1741) message, this method returns 0.

Although applications rarely need to use the value returned by gState (page 1755), it can be passed to the few methods that take an object identifier as a parameter.

See Also

allocateGState (page 1741)

setUpGState (page 1781)

renewGState (page 1769)

releaseGState (page 1768)

lockFocus (page 1759)

heightAdjustLimit

Returns the fraction (from 0.0 to 1.0) of the page that can be pushed onto the next page during automatic pagination to prevent items such as lines of text from being divided across pages.

```
public float heightAdjustLimit()
```

Discussion

This fraction is used to calculate the bottom edge limit for an adjustPageHeight (page 1740) message.

See Also

widthAdjustLimit (page 1787)

hitTest

Returns the farthest descendant of the receiver in the view hierarchy (including itself) that contains *aPoint*, or null if *aPoint* lies completely outside the receiver.

```
public NSView hitTest(NSPoint aPoint)
```

Discussion

aPoint is in the coordinate system of the receiver's superview, not of the receiver itself. This method ignores hidden views.

This method is used primarily by an NSWindow to determine which NSView should receive a mouse-down event. You'd rarely need invoke this method, but you might want to override it to have a view object hide mouse-down events from its subviews.

See Also

isMouseInRect (page 1757)

convertPointToView (page 1745)

setHidden (page 1778)

inLiveResize

A convenience method, expected to be called from drawRect (page 1753) to make decisions about optimized drawing.

```
public boolean inLiveResize()
```

See Also

viewDidEndLiveResize (page 1784)

viewWillStartLiveResize (page 1785)

isDescendantOf

Returns true if the receiver is a subview, immediate or not, of *aView*, or if it's identical to *aView*; otherwise returns false.

```
public boolean isDescendantOf(NSView aView)
```

See Also

superview (page 1782)

subviews (page 1782)

ancestorSharedWithView (page 1742)

isFlipped

Returns true if the receiver uses flipped drawing coordinates or false if it uses native coordinates.

```
public boolean isFlipped()
```

Discussion

NSView's implementation returns `false`; subclasses that use flipped coordinates should override this method to return `true`.

isHidden

Returns whether the receiver is marked as hidden.

```
public boolean isHidden()
```

Discussion

The return value reflects the state of the receiver only, as set in Interface Builder or through the most recent `setHidden` (page 1778) message, and does not account for the state of the receiver's ancestors in the view hierarchy, Thus this method returns `false` when the receiver is effectively hidden because it has a hidden ancestor. See `setHidden` for a discussion of the mechanics and implications of hidden views.

If you want to determine whether a view is effectively hidden, for whatever reason, send the `isHiddenOrHasHiddenAncestor` (page 1757) to the view instead.

Availability

Available in Mac OS X v10.3 and later.

isHiddenOrHasHiddenAncestor

Returns `true` if the receiver is marked as hidden or has an ancestor in the view hierarchy that is marked as hidden; returns `false` otherwise.

```
public boolean isHiddenOrHasHiddenAncestor()
```

Discussion

The return value reflects state set through the `setHidden` (page 1778) method in the receiver of one of its ancestors in the view hierarchy. It does not account for other reasons why a view might be considered hidden, such as being positioned outside its superview's bounds, not having a window, or residing in a window that is offscreen or overlapped by another window.

Availability

Available in Mac OS X v10.3 and later.

See Also

`isHidden` (page 1757)

isMouseInRect

Returns `true` if *`aRect`* contains *`aPoint`* (which represents the hot spot of the mouse cursor), accounting for whether the receiver is flipped or not.

```
public boolean isMouseInRect(NSPoint aPoint, NSRect aRect)
```

Discussion

`aPoint` and *`aRect`* must be expressed in the receiver's coordinate system.

Point-in-rectangle functions generally assume that the bottom edge of a rectangle is outside of the rectangle boundaries, while the upper edge is inside the boundaries. This method views *aRect* from the point of view of the user—that is, this method always treats the bottom edge of the rectangle as the one closest to the bottom edge of the user's screen. By making this adjustment, this function ensures consistent mouse-detection behavior from the user's perspective.

See Also

`hitTest` (page 1756)
`isFlipped` (page 1756)
`convertPointFromView` (page 1744)

isOpaque

Overridden by subclasses to return `true` if the receiver is opaque, `false` otherwise.

```
public boolean isOpaque()
```

Discussion

A view object is opaque if it completely covers its frame rectangle when drawing itself. NSView, being an abstract class, performs no drawing at all and so returns `false`.

See Also

`opaqueAncestor` (page 1762)
`displayRectIgnoringOpacity` (page 1749)
`displayIfNeededIgnoringOpacity` (page 1748)
`displayIfNeededInRectIgnoringOpacity` (page 1748)

isRotatedFromBase

Returns `true` if the receiver or any of its ancestors has ever received a `setFrameRotation` (page 1777) or `setBoundsRotation` (page 1775) message; otherwise returns `false`.

```
public boolean isRotatedFromBase()
```

Discussion

The intent of this information is to optimize drawing and coordinate calculation, not necessarily to reflect the exact state of the receiver's coordinate system, so it may not reflect the actual rotation. For example, if an NSView is rotated to 45 degrees and later back to 0, this method still returns `true`.

See Also

`frameRotation` (page 1755)
`boundsRotation` (page 1743)

isRotatedOrScaledFromBase

Returns `true` if the receiver or any of its ancestors has ever had a nonzero frame or bounds rotation, or has been scaled from the window's base coordinate system; otherwise returns `false`.

```
public boolean isRotatedOrScaledFromBase()
```

Discussion

The intent of this information is to optimize drawing and coordinate calculation, not necessarily to reflect the exact state of the receiver's coordinate system, so it may not reflect the actual rotation or scaling. For example, if an NSView is rotated to 45 degrees and later back to 0, this method still returns `true`.

See Also

`frameRotation` (page 1755)
`boundsRotation` (page 1743)
`centerScanRect` (page 1744)
`setBounds` (page 1773)
`setBoundsSize` (page 1775)
`scaleUnitSquareToSize` (page 1771)

knowsPageRange

Returns `true` if the receiver handles page boundaries.

```
public boolean knowsPageRange(NSMutableRange aRange)
```

Discussion

Returns `false` if the receiver uses NSView's default auto-pagination mechanism. If it returns `true`, the page range is returned in *aRange*. Page numbers are one-based, that is pages run from one to N.

The default implementation returns `false`. Override this method if your class handles page boundaries.

locationOfPrintRect

Invoked by `print` (page 1765) to determine the location of *aRect*, the rectangle being printed on the physical page.

```
public NSPoint locationOfPrintRect(NSRect aRect)
```

Discussion

The return value of this method is used to set the origin for *aRect*, whose size the receiver can examine in order to properly place it. Both the rectangle and the returned location are expressed in the default coordinate system of the page.

NSView's implementation places *aRect* according to the status of the NSPrintInfo object for the print job. By default it places the image in the upper-left corner of the page, but if NSPrintInfo's `isHorizontallyCentered` (page 1142) or `isVerticallyCentered` (page 1142) method returns `true`, it centers a single-page image along the appropriate axis. A multiple-page document, however, is always placed so the divided pieces can be assembled at their edges.

lockFocus

Locks the focus on the receiver, so subsequent commands take effect in the receiver's window and coordinate system.

```
public void lockFocus()
```

Discussion

If you don't use a `display...` method to draw an NSView, you must invoke `lockFocus` before invoking methods that send commands to the window server, and must balance it with an `unlockFocus` (page 1783) message when finished.

See Also

`focusView` (page 1738)

`display` (page 1747)

`drawRect` (page 1753)

menuForEvent

Overridden by subclasses to return a context-sensitive pop-up menu for the mouse-down event *theEvent*.

```
public NSMenu menuForEvent(NSEvent theEvent)
```

Discussion

The receiver can use information in the mouse event, such as its location over a particular element of the receiver, to determine what kind of menu to return. For example, a text object might display a text-editing menu when the cursor lies over text and a menu for changing graphics attributes when the cursor lies over an embedded image.

NSView's implementation returns the receiver's normal menu.

See Also

`defaultMenu` (page 1738)

`menu` (page 1192) (NSResponder)

mouseDownCanMoveWindow

Returns `true` if the receiver does not need to handle a mouse down and can pass it through to the view; `false` if it needs to handle the mouse down.

```
public boolean mouseDownCanMoveWindow()
```

Discussion

This allows iApp-type applications to properly determine the region by which a window can be moved. By default, this method returns `false` if the view is opaque; otherwise, it returns `true`. Subclasses can override this method to return a different value.

Availability

Available in Mac OS X v10.2 and later.

needsDisplay

Returns `true` if the receiver needs to be displayed, as indicated using `setNeedsDisplay` (page 1779); returns `false` otherwise.

```
public boolean needsDisplay()
```

Discussion

The `displayIfNeeded...` methods check this status to avoid unnecessary drawing, and all display methods clear this status to indicate that the view object is up to date.

needsPanelToBecomeKey

Overridden by subclasses to return `true` if the receiver requires its panel, which might otherwise avoid becoming key, to become the key window so that it can handle keyboard input.

```
public boolean needsPanelToBecomeKey()
```

Discussion

Such a subclass should also override `acceptsFirstResponder` (page 1189) to return `true`. NSView's implementation returns `false`.

See Also

`becomesKeyOnlyIfNeeded` (page 1053) (NSPanel)

needsToDrawRect

Returns whether rectangle *aRect* intersects any part of the area that the receiver is being asked to draw.

```
public boolean needsToDrawRect(NSRect aRect)
```

Discussion

You typically send this message from within a `drawRect` (page 1753) implementation. It gives you a convenient way to determine whether any part of a given graphical entity might need to be drawn. It is optimized to efficiently reject any rectangle that lies outside the bounding box of the area the receiver is being asked to draw in `drawRect`.

Availability

Available in Mac OS X v10.3 and later.

nextKeyView

Returns the view object following the receiver in the key view loop, or `null` if there is none.

```
public NSView nextKeyView()
```

Discussion

This view should, if possible, be made first responder when the user navigates forward from the receiver using keyboard interface control.

See Also

`nextValidKeyView` (page 1762)
`setNextKeyView` (page 1779)
`previousKeyView` (page 1764)
`previousValidKeyView` (page 1765)
`selectNextKeyView` (page 1852) (NSWindow)
`selectKeyViewFollowingView` (page 1852) (NSWindow)
`selectPreviousKeyView` (page 1853) (NSWindow)

selectKeyViewPrecedingView (page 1852) (NSWindow)

nextValidKeyView

Returns the closest view object in the key view loop that follows the receiver and actually accepts first responder status, or null if there is none.

```
public NSView nextValidKeyView()
```

Discussion
This method ignores hidden views when it determines the next valid key view.

See Also
nextKeyView (page 1761)
setNextKeyView (page 1779)
previousKeyView (page 1764)
previousValidKeyView (page 1765)
selectNextKeyView (page 1852) (NSWindow)
selectKeyViewFollowingView (page 1852) (NSWindow)
selectPreviousKeyView (page 1853) (NSWindow)
selectKeyViewPrecedingView (page 1852) (NSWindow)
setHidden (page 1778)

opaqueAncestor

Returns the receiver's closest opaque ancestor (including the receiver itself).

```
public NSView opaqueAncestor()
```

See Also
isOpaque (page 1758)
displayRectIgnoringOpacity (page 1749)
displayIfNeededIgnoringOpacity (page 1748)
displayIfNeededInRectIgnoringOpacity (page 1748)

performDragOperation

```
public boolean performDragOperation(NSDraggingInfo draggingInfo)
```

Discussion
Invoked after the released image has been removed from the screen and the previous prepareForDragOperation (page 1764) message has returned true. *draggingInfo* contains information about the dragging operation. This method should do the real work of importing the pasteboard data represented by the image. If the receiver accepts the data, returns true, otherwise returns false.

See Also
concludeDragOperation (page 1744)

performKeyEquivalent

Implemented by subclasses to respond to key equivalents (also known as shortcuts).

```
public boolean performKeyEquivalent(NSEvent theEvent)
```

Discussion

If the receiver's key equivalent is the same as the characters of the key-down event `theEvent`, as returned by `charactersIgnoringModifiers` (page 612), it should take the appropriate action and return `true`. Otherwise, it should return the result of invoking `super`'s implementation. NSView's implementation of this method simply passes the message down the view hierarchy (from superviews to subviews) and returns `false` if none of the receiver's subviews responds `true`.

See Also

`performMnemonic` (page 1763)

`keyDown` (page 1840) (NSWindow)

performMnemonic

Implemented by subclasses to respond to mnemonics.

```
public boolean performMnemonic(String aString)
```

Discussion

If the receiver's mnemonic is the same as the characters of the string `aString`, it should take the appropriate action and return `true`. Otherwise, it should return the result of invoking `super`'s implementation. NSView's implementation of this method simply passes the message down the view hierarchy (from superviews to subviews) and returns `false` if none of the receiver's subviews responds `true`. Mnemonics are not supported in Mac OS X.

See Also

`performKeyEquivalent` (page 1763)

`keyDown` (page 1840) (NSWindow)

postsBoundsChangedNotifications

Returns `true` if the receiver posts notifications to the default notification center whenever its bounds rectangle changes; returns `false` otherwise.

```
public boolean postsBoundsChangedNotifications()
```

Discussion

See `setPostsBoundsChangedNotifications` (page 1779) for a list of methods that result in notifications.

postsFrameChangedNotifications

Returns `true` if the receiver posts notifications to the default notification center whenever its frame rectangle changes; returns `false` otherwise.

```
public boolean postsFrameChangedNotifications()
```

Discussion

See `setPostsBoundsChangedNotifications` (page 1779) for a list of methods that result in notifications.

prepareForDragOperation

```
public boolean prepareForDragOperation(NSDraggingInfo draggingInfo)
```

Discussion

Invoked when the image is released, if the most recent `draggingEntered` (page 1750) or `draggingUpdated` (page 1750) message returned an acceptable drag-operation value. *`draggingInfo`* contains information about the dragging operation. Returns `true` if the receiver agrees to perform the drag operation and `false` if not.

See Also

`performDragOperation` (page 1762)

preservesContentDuringLiveResize

Returns `true` if the view supports the optimization of live resize operations by preserving content that has not moved; otherwise, returns `false`.

```
public boolean preservesContentDuringLiveResize()
```

Discussion

The default is `false`. If your view supports the content preservation feature, you should override this method and have your implementation return `true`.

Content preservation lets your view decide what to redraw during a live resize operation. If your view supports this feature, you should also provide a custom implementation of `setFrameSize` (page 1777) that invalidates the portions of your view that actually need to be redrawn.

For information on how to implement this feature in your views, see *Cocoa Performance Guidelines*.

Availability

Available in Mac OS X v10.4 and later.

See Also

`setFrameSize` (page 1777)

previousKeyView

Returns the view object preceding the receiver in the key view loop, or `null` if there is none.

```
public NSView previousKeyView()
```

Discussion

This view should, if possible, be made first responder when the user navigates backward from the receiver using keyboard interface control.

See Also

`previousValidKeyView` (page 1765)

nextKeyView (page 1761)

nextValidKeyView (page 1762)

setNextKeyView (page 1779)

selectNextKeyView (page 1852) (NSWindow)

selectKeyViewFollowingView (page 1852) (NSWindow)

selectPreviousKeyView (page 1853) (NSWindow)

selectKeyViewPrecedingView (page 1852) (NSWindow)

previousValidKeyView

Returns the closest view object in the key view loop that precedes the receiver and actually accepts first responder status, or `null` if there is none.

```
public NSView previousValidKeyView()
```

Discussion

This method ignores hidden views when it determines the previous valid key view.

See Also

previousKeyView (page 1764)

nextValidKeyView (page 1762)

nextKeyView (page 1761)

setNextKeyView (page 1779)

selectNextKeyView (page 1852) (NSWindow)

selectKeyViewFollowingView (page 1852) (NSWindow)

selectPreviousKeyView (page 1853) (NSWindow)

selectKeyViewPrecedingView (page 1852) (NSWindow)

setHidden (page 1778)

print

This action method opens the Print panel, and if the user chooses an option other than canceling, prints the receiver and all its subviews to the device specified in the Print panel.

```
public void print(Object sender)
```

See Also

dataWithEPSInsideRect (page 1746)

writeEPSInsideRectToPasteboard (page 1788)

rectForPage

Implemented by subclasses to determine the portion of the receiver to be printed for the page number page.

```
public NSRect rectForPage(int pageNumber)
```

Discussion

If the receiver responded true to an earlier knowsPageRange (page 1759) message, this method is invoked for each page it specified in the out parameters of that message. The receiver is later made to display this rectangle in order to generate the image for this page. Page numbers are one-based, that is pages run from one to N. This method returns NSRect.ZeroRect if *pageNumber* is outside the receiver's bounds.

If an NSView responds false to knowsPageRange (page 1759), this method isn't invoked by the printing mechanism.

See Also

adjustPageHeight (page 1740)
adjustPageWidth (page 1740)

rectPreservedDuringLiveResize

Returns the rectangle identifying the portion of your view that did not change during a live resize operation.

```
public NSRect rectPreservedDuringLiveResize()
```

Discussion

The returned rectangle is in the coordinate system of your view and reflects the space your view previously occupied. This rectangle may be smaller or the same size as your view's current bounds, depending on whether the view grew or shrunk.

If your view does not support content preservation during live resizing, the returned rectangle will be empty. To support content preservation, override preservesContentDuringLiveResize (page 1764) in your view and have your implementation return true.

Note: The window containing your view must also support content preservation. To enable support for this feature in your window, use the setPreservesContentDuringLiveResize: method of NSWindow.

Availability

Available in Mac OS X v10.4 and later.

See Also

rectsExposedDuringLiveResize (page 1767)
preservesContentDuringLiveResize (page 1764)
setPreservesContentDuringLiveResize (page 1866) (NSWindow)

rectsBeingDrawn

Returns a list of non-overlapping rectangles that define the area the receiver is being asked to draw in drawRect (page 1753).

```
public NSArray rectsBeingDrawn()
```

Discussion

An implementation of drawRect can use this information to test whether objects or regions within the view intersect with the rectangles in the list, and thereby avoid unnecessary drawing that would be completely clipped away.

The needsToDrawRect (page 1761) method gives you a convenient way to test individual objects for intersection with the area being drawn in drawRect (page 1753). However, you may want to retrieve and directly inspect the rectangle list if this is a more efficient way to perform intersection testing.

You should send this message only from within a drawRect (page 1753) implementation. The *aRect* parameter of drawRect is the rectangle enclosing the returned list of rectangles; you can use it in an initial pass to reject objects that are clearly outside the area to be drawn.

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

See Also
wantsDefaultClipping (page 1787)

rectsExposedDuringLiveResize

Returns a list of rectangles indicating the newly exposed areas of the receiver.

```
public NSArray rectsExposedDuringLiveResize()
```

Discussion
The returned rectangles are in the coordinate space of the receiver. If your view does not support content preservation during live resizing, the entire area of your view is returned. To support content preservation, override preservesContentDuringLiveResize (page 1764) in your view and have your implementation return true.

Note: The window containing your view must also support content preservation. To enable support for this feature in your window, use the setPreservesContentDuringLiveResize: method of NSWindow.

If the view decreased in both height and width, the list of returned rectangles will be empty. If the view increased in both height and width and its upper-left corner stayed anchored in the same position, the list of returned rectangles will contain a vertical and horizontal component indicating the exposed area.

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

See Also
preservesContentDuringLiveResize (page 1764)
rectPreservedDuringLiveResize (page 1766)
setPreservesContentDuringLiveResize (page 1866) (NSWindow)

reflectScrolledClipView

Notifies *aClipView*'s superview that either *aClipView*'s bounds rectangle or the document view's frame rectangle has changed, and that any indicators of the scroll position need to be adjusted.

```
public void reflectScrolledClipView(NSClipView aClipView)
```

Discussion
NSScrollView implements this method to update its NSScrollers.

registerForDraggedTypes

Registers *pboardTypes* as the pasteboard types that the receiver will accept as the destination of an image-dragging session.

```
public void registerForDraggedTypes(NSArray pboardTypes)
```

Discussion
Registering an NSView for dragged types automatically makes it a candidate destination object for a dragging session. As such, it must properly implement some or all of the NSDraggingDestination interface methods. As a convenience, NSView provides default implementations of these methods. See the NSDraggingDestination (page 1955) interface specification for details.

See Also
`unregisterDraggedTypes` (page 1784)

releaseGState

Frees the receiver's graphics state object, if it has one.

```
public void releaseGState()
```

See Also
`allocateGState` (page 1741)

removeCursorRect

Completely removes a cursor rectangle from the receiver.

```
public void removeCursorRect(NSRect aRect, NSCursor aCursor)
```

Discussion
aRect and *aCursor* must match values previously specified using `addCursorRect` (page 1739).

You should rarely need to use this method. `resetCursorRects` (page 1770), which is invoked any time cursor rectangles need to be rebuilt, should establish only the cursor rectangles needed. If you implement `resetCursorRects` (page 1770) in this way, you can then simply modify the state that `resetCursorRects` (page 1770) uses to build its cursor rectangles and then invoke NSWindow's `invalidateCursorRectsForView` (page 1836).

See Also
`discardCursorRects` (page 1747)

removeFromSuperview

Unlinks the receiver from its superview and its NSWindow, removes it from the responder chain, and invalidates its cursor rectangles.

```
public void removeFromSuperview()
```

Discussion
Never invoke this method during display.

See Also

`addSubview` (page 1739)

`removeFromSuperviewWithoutNeedingDisplay` (page 1769)

removeFromSuperviewWithoutNeedingDisplay

Unlinks the receiver from its superview and its NSWindow, removes it from the responder chain, but does not invalidate its cursor rectangles to cause redrawing.

```
public void removeFromSuperviewWithoutNeedingDisplay()
```

Discussion

Unlike its counterpart, `removeFromSuperview` (page 1768), this method can be safely invoked during display.

See Also

`addSubview` (page 1739)

removeTrackingRect

Removes the tracking rectangle identified by *aTag*, which is the value returned by a previous `addTrackingRect` (page 1739) message.

```
public void removeTrackingRect(int aTag)
```

renewGState

Invalidates the receiver's graphics state object, if it has one, so it will be regenerated using `setUpGState` (page 1781) the next time the receiver is focused for drawing.

```
public void renewGState()
```

See Also

`lockFocus` (page 1759)

replaceSubview

Replaces *oldView* with *newView* in the receiver's subviews.

```
public void replaceSubview(NSView oldView, NSView newView)
```

Discussion

Does nothing if *oldView* is not a subview of the receiver.

Neither *oldView* nor *newView* may be `null`, and the behavior is undefined if either of these parameters is `null`.

See Also

`addSubview` (page 1739)

resetCursorRects

Overridden by subclasses to define their default cursor rectangles.

```
public void resetCursorRects()
```

Discussion
A subclass's implementation must invoke `addCursorRect` (page 1739) for each cursor rectangle it wants to establish. NSView's implementation does nothing.

Application code should never invoke this method directly; it's invoked automatically as described in "Handling Tracking-Rectangle and Cursor-Update Events in Views". Use the `invalidateCursorRectsForView` (page 1836) method instead to explicitly rebuild cursor rectangles.

See Also
`visibleRect` (page 1786)

resizeSubviewsWithOldSize

Informs the receivers's subviews that the receiver's bounds rectangle size has changed from *oldBoundsSize*.

```
public void resizeSubviewsWithOldSize(NSSize oldBoundsSize)
```

Discussion
If the receiver is configured to autoresize its subviews, this method is automatically invoked by any method that changes the receiver's frame size.

NSView's implementation sends `resizeWithOldSuperviewSize` (page 1770) to the receiver's subviews with *oldBoundsSize* as the argument. You shouldn't invoke this method directly, but you can override it to define a specific retiling behavior.

See Also
`setAutoresizesSubviews` (page 1773)

resizeWithOldSuperviewSize

Informs the receiver that the bounds size of its superview has changed from *oldBoundsSize*.

```
public void resizeWithOldSuperviewSize(NSSize oldBoundsSize)
```

Discussion
This method is normally invoked automatically from `resizeSubviewsWithOldSize` (page 1770).

NSView's implementation resizes the receiver according to the autoresizing options listed under the `setAutoresizingMask` (page 1773) method description. You shouldn't invoke this method directly, but you can override it to define a specific resizing behavior.

rotateByAngle

Rotates the receiver's bounds rectangle by *angle* degrees around the origin of the coordinate system, (0.0, 0.0).

```
public void rotateByAngle(float angle)
```

Discussion

See the setBoundsRotation (page 1775) method description for more information. This method neither redisplays the receiver nor marks it as needing display. You must do this yourself with display (page 1747) or setNeedsDisplay (page 1779).

This method posts a ViewBoundsDidChangeNotification (page 1788) to the default notification center if the receiver is configured to do so.

See Also

setFrameRotation (page 1777)

setPostsBoundsChangedNotifications (page 1779)

scaleUnitSquareToSize

Scales the receiver's coordinate system so that the unit square scales to *newUnitSize*.

```
public void scaleUnitSquareToSize(NSSize newUnitSize)
```

Discussion

For example, a *newUnitSize* of (0.5, 1.0) causes the receiver's horizontal coordinates to be halved, in turn doubling the width of its bounds rectangle. Note that scaling is performed from the origin of the coordinate system, (0.0, 0.0), not the origin of the bounds rectangle; as a result, both the origin and size of the bounds rectangle are changed. The frame rectangle remains unchanged.

This method neither redisplays the receiver nor marks it as needing display. You must do this yourself with display (page 1747) or setNeedsDisplay (page 1779).

This method posts a ViewBoundsDidChangeNotification (page 1788) to the default notification center if the receiver is configured to do so.

See Also

setBoundsSize (page 1775)

setPostsBoundsChangedNotifications (page 1779)

scrollClipViewToPoint

Notifies the superview of *aClipView* that *aClipView* needs to set its bounds rectangle origin to *newOrigin*.

```
public void scrollClipViewToPoint(NSClipView aClipView, NSPoint newOrigin)
```

Discussion

The superview of *aClipView* should then send a scrollToPoint (page 347) message to *aClipView* with *newOrigin* as the argument. This mechanism is provided so the NSClipView's superview can coordinate scrolling of multiple tiled NSClipViews.

See Also

scrollToPoint (page 347) (NSClipView)

scrollPoint

Scrolls the receiver's closest ancestor NSClipView so `aPoint` in the receiver lies at the origin of the NSClipView's bounds rectangle.

```
public void scrollPoint(NSPoint aPoint)
```

See Also

`autoscroll` (page 1742)

`scrollToPoint` (page 347) (NSClipView)

`isDescendantOf` (page 1756)

scrollRect

Copies the visible portion of the receiver's rendered image within `aRect` and lays that portion down again at `offset` from `aRect`'s origin.

```
public void scrollRect(NSRect aRect, NSSize offset)
```

Discussion

This method is useful during scrolling or translation of the coordinate system to efficiently move as much of the receiver's rendered image as possible without requiring it to be redrawn, following these steps:

1. Invoke `scrollRect` (page 1772) to copy the rendered image.
2. Move the view object's origin or scroll it within its superview.
3. Calculate the newly exposed rectangles and invoke `setNeedsDisplay` (page 1779) to draw them.

You should rarely need to use this method, however. The `scrollPoint` (page 1772), `scrollRectToVisible` (page 1772), and `autoscroll` (page 1742) methods automatically perform optimized scrolling.

See Also

`setBoundsOrigin` (page 1774)

`translateOriginToPoint` (page 1783)

scrollRectToVisible

Scrolls the receiver's closest ancestor NSClipView the minimum distance needed so `aRect` in the receiver becomes visible in the NSClipView.

```
public boolean scrollRectToVisible(NSRect aRect)
```

Discussion

Returns `true` if any scrolling is performed; otherwise returns `false`.

See Also

`autoscroll` (page 1742)

`scrollToPoint` (page 347) (NSClipView)

`isDescendantOf` (page 1756)

setAutoresizesSubviews

Determines whether the receiver automatically resizes its subviews when its frame size changes.

```
public void setAutoresizesSubviews(boolean flag)
```

Discussion

If *flag* is `true`, the receiver invokes `resizeSubviewsWithOldSize` (page 1770) whenever its frame size changes; if *flag* is `false`, it doesn't. View objects do autoresize their subviews by default.

See Also

`autoresizesSubviews` (page 1742)

setAutoresizingMask

Determines how the receiver's `resizeWithOldSuperviewSize` (page 1770) method changes its frame rectangle.

```
public void setAutoresizingMask(int mask)
```

Discussion

mask can be specified by combining any of the following options using the C bitwise OR operator:

Option	Meaning
`ViewNotSizable`	The receiver cannot be resized.
`ViewMinXMargin`	The left margin between the receiver and its superview is flexible.
`ViewWidthSizable`	The receiver's width is flexible.
`ViewMaxXMargin`	The right margin between the receiver and its superview is flexible.
`ViewMinYMargin`	The bottom margin between the receiver and its superview is flexible.
`ViewHeightSizable`	The receiver's height is flexible.
`ViewMaxYMargin`	The top margin between the receiver and its superview is flexible.

Where more than one option along an axis is set, `resizeWithOldSuperviewSize` (page 1770) by default distributes the size difference as evenly as possible among the flexible portions. For example, if `ViewWidthSizable` and `ViewMaxXMargin` are set and the *superview*'s width has increased by 10.0 units, the receiver's frame and right margin are each widened by 5.0 units.

See Also

`autoresizingMask` (page 1742)

`resizeSubviewsWithOldSize` (page 1770)

`setAutoresizesSubviews` (page 1773)

setBounds

Sets the receiver's bounds rectangle to *boundsRect*.

```
public void setBounds(NSRect boundsRect)
```

Discussion

The bounds rectangle determines the origin and scale of the receiver's coordinate system within its frame rectangle. This method neither redisplays the receiver nor marks it as needing display. You must do this yourself with `display` (page 1747) or `setNeedsDisplay` (page 1779).

This method posts a `ViewBoundsDidChangeNotification` (page 1788) to the default notification center if the receiver is configured to do so.

After calling this method, NSView creates an internal transform (or appends these changes to an existing internal transform) to convert from frame coordinates to bounds coordinates in your view. As long as the width-to-height ratio of the two coordinate systems remains the same, your content appears normal. If the ratios differ, your content may appear skewed.

See Also

`bounds` (page 1743)

`setBoundsRotation` (page 1775)

`setBoundsOrigin` (page 1774)

`setBoundsSize` (page 1775)

`setFrame` (page 1776)

`setPostsBoundsChangedNotifications` (page 1779)

setBoundsOrigin

Sets the origin of the receiver's bounds rectangle to *newOrigin*, effectively shifting its coordinate system so *newOrigin* lies at the origin of the receiver's frame rectangle.

```
public void setBoundsOrigin(NSPoint newOrigin)
```

Discussion

This method neither redisplays the receiver nor marks it as needing display. You must do this yourself with `display` or `setNeedsDisplay` (page 1779).

This method posts a `ViewBoundsDidChangeNotification` (page 1788) to the default notification center if the receiver is configured to do so.

After calling this method, NSView creates an internal transform (or appends these changes to an existing internal transform) to convert from frame coordinates to bounds coordinates in your view. As long as the width-to-height ratio of the two coordinate systems remains the same, your content appears normal. If the ratios differ, your content may appear skewed.

See Also

`translateOriginToPoint` (page 1783)

`bounds` (page 1743)

`setBoundsRotation` (page 1775)

`setBounds` (page 1773)

`setBoundsSize` (page 1775)

`setPostsBoundsChangedNotifications` (page 1779)

setBoundsRotation

Sets the rotation of the receiver's bounds rectangle to *angle* degrees.

```
public void setBoundsRotation(float angle)
```

Discussion

Positive values indicate counterclockwise rotation, negative clockwise. Rotation is performed around the coordinate system origin, (0.0, 0.0), which need not coincide with that of the frame rectangle or the bounds rectangle. This method neither redisplays the receiver nor marks it as needing display. You must do this yourself with `display` (page 1747) or `setNeedsDisplay` (page 1779).

This method posts a `ViewBoundsDidChangeNotification` (page 1788) to the default notification center if the receiver is configured to do so.

Bounds rotation affects the orientation of the drawing within the view object's frame rectangle, but not the orientation of the frame rectangle itself. Also, for a rotated bounds rectangle to enclose all the visible areas of its view object—that is, to guarantee coverage over the frame rectangle—it must also contain some areas that aren't visible. This can cause unnecessary drawing to be requested, which may affect performance. It may be better in many cases to rotate the coordinate system in the `drawRect` (page 1753) method rather than use this method.

After calling this method, NSView creates an internal transform (or appends these changes to an existing internal transform) to convert from frame coordinates to bounds coordinates in your view. As long as the width-to-height ratio of the two coordinate systems remains the same, your content appears normal. If the ratios differ, your content may appear skewed.

See Also

`rotateByAngle` (page 1770)
`boundsRotation` (page 1743)
`setFrameRotation` (page 1777)
`setPostsBoundsChangedNotifications` (page 1779)

setBoundsSize

Sets the size of the receiver's bounds rectangle to *newSize*, inversely scaling its coordinate system relative to its frame rectangle.

```
public void setBoundsSize(NSSize newSize)
```

Discussion

For example, a view object with a frame size of (100.0, 100.0) and a bounds size of (200.0, 100.0) draws half as wide along the x axis. This method neither redisplays the receiver nor marks it as needing display. You must do this yourself with `display` (page 1747) or `setNeedsDisplay` (page 1779).

This method posts a `ViewBoundsDidChangeNotification` (page 1788) to the default notification center if the receiver is configured to do so.

After calling this method, NSView creates an internal transform (or appends these changes to an existing internal transform) to convert from frame coordinates to bounds coordinates in your view. As long as the width-to-height ratio of the two coordinate systems remains the same, your content appears normal. If the ratios differ, your content may appear skewed.

See Also

`bounds` (page 1743)

`setBoundsRotation` (page 1775)

`setBounds` (page 1773)

`setBoundsOrigin` (page 1774)

`setPostsBoundsChangedNotifications` (page 1779)

setFocusRingType

Sets the type of focus ring to be drawn around the receiver.

```
public native void setFocusRingType(int focusRingType)
```

Discussion

Possible values are listed in the "Constants" (page 727) section of NSGraphics.You can specify `NSGraphics.FocusRingTypeNone` to indicate you do not want your view to have a focus ring.

> **Note:** This method only sets the desired focus ring type and does not cause the view to draw the actual focus ring. You are responsible for drawing the focus ring in your view's `drawRect:` method whenever your view is made the first responder.

Availability

Available in Mac OS X v10.3 and later.

See Also

`focusRingType` (page 1754)

setFrame

Sets the receiver's frame rectangle to *frameRect*, thereby repositioning and resizing it within the coordinate system of its superview.

```
public void setFrame(NSRect frameRect)
```

Discussion

This method neither redisplays the receiver nor marks it as needing display. You must do this yourself with `display` (page 1747) or `setNeedsDisplay` (page 1779).

This method posts a`ViewFrameDidChangeNotification` (page 1789) to the default notification center if the receiver is configured to do so.

If your view does not use a custom bounds rectangle, this method also sets your view bounds to match the size of the new frame. You specify a custom bounds rectangle by calling `setBounds` (page 1773), `setBoundsOrigin` (page 1774), `setBoundsRotation` (page 1775), or `setBoundsSize` (page 1775)explicitly. Once set, NSView creates an internal transform to convert from frame coordinates to bounds coordinates. As long as the width-to-height ratio of the two coordinate systems remains the same, your content appears normal. If the ratios differ, your content may appear skewed.

See Also

`frame` (page 1754)

`setFrameRotation` (page 1777)
`setFrameOrigin` (page 1777)
`setFrameSize` (page 1777)
`setPostsFrameChangedNotifications` (page 1780)

setFrameOrigin

Sets the origin of the receiver's frame rectangle to *newOrigin*, effectively repositioning it within its superview.

```
public void setFrameOrigin(NSPoint newOrigin)
```

Discussion

This method neither redisplays the receiver nor marks it as needing display. You must do this yourself with `display` (page 1747) or `setNeedsDisplay` (page 1779).

This method posts a `ViewFrameDidChangeNotification` (page 1789) to the default notification center if the receiver is configured to do so.

See Also

`frame` (page 1754)
`setFrameSize` (page 1777)
`setFrame` (page 1776)
`setFrameRotation` (page 1777)
`setPostsFrameChangedNotifications` (page 1780)

setFrameRotation

Sets the rotation of the receiver's frame rectangle to *angle* degrees, rotating it within its superview without affecting its coordinate system.

```
public void setFrameRotation(float angle)
```

Discussion

Positive values indicate counterclockwise rotation, negative clockwise. Rotation is performed around the origin of the frame rectangle.

This method neither redisplays the receiver nor marks it as needing display. You must do this yourself with `display` (page 1747) or `setNeedsDisplay` (page 1779).

This method posts a `ViewFrameDidChangeNotification` (page 1789) to the default notification center if the receiver is configured to do so.

See Also

`frameRotation` (page 1755)
`setBoundsRotation` (page 1775)

setFrameSize

Sets the size of the receiver's frame rectangle to *newSize*, resizing it within its superview without affecting its coordinate system.

```
public void setFrameSize(NSSize newSize)
```

Discussion
This method neither redisplays the receiver nor marks it as needing display. You must do this yourself with `display` (page 1747) or `setNeedsDisplay` (page 1779).

This method posts a `ViewFrameDidChangeNotification` (page 1789) to the default notification center if the receiver is configured to do so.

In Mac OS X version 10.4 and later, you can override this method to support content preservation during live resizing. In your overridden implementation, include some conditional code to be executed only during a live resize operation. Your code must invalidate any portions of your view that need to be redrawn.

See Also
`frame` (page 1754)
`setFrameOrigin` (page 1777)
`setFrame` (page 1776)
`setFrameRotation` (page 1777)
`setPostsFrameChangedNotifications` (page 1780)

setHidden

Sets whether the receiver is hidden.

```
public void setHidden(boolean flag)
```

Discussion
A hidden view disappears from its window and does not receive input events. It remains in its superview's list of subviews, however, and participates in autoresizing as usual. The Application Kit also disables any cursor rectangle, tool-tip rectangle, or tracking rectangle associated with a hidden view. Hiding a view with subviews has the effect of hiding those subviews and any view descendents they might have. This effect is implicit and does not alter the hidden state of the receiver's descendents as reported by `isHidden` (page 1757).

Hiding the view that is the window's current first responder causes the view's next valid key view (`nextValidKeyView` (page 1762)) to become the new first responder. A hidden view remains in the `nextKeyView` (page 1761) chain of views it was previously part of, but is ignored during keyboard navigation.

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

See Also
`isHidden` (page 1757)
`isHiddenOrHasHiddenAncestor` (page 1757)

setKeyboardFocusRingNeedsDisplayInRect

Invalidates the area around the focus ring.

```
public void setKeyboardFocusRingNeedsDisplayInRect(NSRect rect)
```

Discussion

rect is the rectangle of the control or cell. *rect* will be expanded to include the focus ring for invalidation.

setNeedsDisplay

If *flag* is true, marks the receiver's entire bounds as needing display; if *flag* is false, marks it as not needing display.

```
public void setNeedsDisplay(boolean aFlag)
```

Discussion

Whenever the data or state used for drawing a view object changes, the view should be sent this message. NSViews marked as needing display are automatically redisplayed on each pass through the application's event loop. (View objects that need to redisplay before the event loop comes around can of course immediately be sent the appropriate display... method.)

Marks the region of the receiver within *invalidRect* as needing display, increasing the receiver's existing invalid region to include it.

```
public void setNeedsDisplay(NSRect invalidRect)
```

Discussion

A later displayIfNeeded... method will then perform drawing only within the invalid region. NSViews marked as needing display are automatically redisplayed on each pass through the application's event loop. (View objects that need to redisplay before the event loop comes around can of course immediately be sent the appropriate display... method.)

See Also

needsDisplay (page 1760)

setNextKeyView

Inserts *aView* after the receiver in the key view loop of the receiver's NSWindow.

```
public void setNextKeyView(NSView aView)
```

See Also

nextKeyView (page 1761)

nextValidKeyView (page 1762)

previousKeyView (page 1764)

previousValidKeyView (page 1765)

setPostsBoundsChangedNotifications

Controls whether the receiver informs observers when its bounds rectangle changes.

```
public void setPostsBoundsChangedNotifications(boolean flag)
```

Discussion

If *flag* is `true`, the receiver will post notifications to the default notification center whenever its bounds rectangle changes; if *flag* is `false` it won't. Note that if *flag* is `true` and bounds notifications are suppressed, when the bounds change notification is reenabled the view will immediately post a single such notification if its bounds changed during this time. This will happen even if there has been no net change in the view's bounds.

The following methods can result in notification posting:

`setBounds` (page 1773)
`setBoundsOrigin` (page 1774)
`setBoundsRotation` (page 1775)
`setBoundsSize` (page 1775)
`translateOriginToPoint` (page 1783)
`scaleUnitSquareToSize` (page 1771)
`rotateByAngle` (page 1770)

See Also

`postsBoundsChangedNotifications` (page 1763)

setPostsFrameChangedNotifications

Controls whether the receiver informs observers when its frame rectangle changes.

```
public void setPostsFrameChangedNotifications(boolean flag)
```

Discussion

If *flag* is `true`, the receiver will post notifications to the default notification center whenever its frame rectangle changes; if *flag* is `false` it won't.Note that if *flag* is `true` and frame notifications are suppressed, when the frame change notification is reenabled the view will immediately post a single such notification if its frame changed during this time. This will happen even if there has been no net change in the view's frame.

The following methods can result in notification posting:

`setFrame` (page 1776)
`setFrameOrigin` (page 1777)
`setFrameRotation` (page 1777)
`setFrameSize` (page 1777)

See Also

`postsFrameChangedNotifications` (page 1763)

setToolTip

Sets the tool tip text for the view to *string*.

```
public void setToolTip(String string)
```

Discussion

If *string* is `null`, cancels tool tip display for the view.

See Also

toolTip (page 1783)

setUpGState

Overridden by subclasses to (re)initialize the receiver's graphics state object.

```
public void setUpGState()
```

Discussion

This method is automatically invoked when the graphics state object created using allocateGState (page 1741) needs to be initialized. NSView's implementation does nothing. Your subclass can override it to set the current font, line width, or any other graphics state parameter except coordinate transformations and the clipping path—these are established by the frame and bounds rectangles and by methods such as scaleUnitSquareToSize (page 1771) and translateOriginToPoint (page 1783). Note that drawRect: can further transform the coordinate system and clipping path for whatever temporary effects it needs.

See Also

allocateGState (page 1741)

renewGState (page 1769)

shouldDelayWindowOrderingForEvent

Overridden by subclasses to allow the user to drag images from the receiver without its window moving forward and possibly obscuring the destination and without activating the application.

```
public boolean shouldDelayWindowOrderingForEvent(NSEvent theEvent)
```

Discussion

If this method returns true, the normal window-ordering and activation mechanism is delayed (not necessarily prevented) until the next mouse-up event. If it returns false, then normal ordering and activation occur. Never invoke this method directly; it's invoked automatically for each mouse-down event directed at the NSView.

An NSView subclass that allows dragging should implement this method to return true if *theEvent*, an initial mouse-down event, is potentially the beginning of a dragging session or of some other context where window ordering isn't appropriate. This method is invoked before a mouseDown (page 1192) message for *theEvent* is sent. NSView's implementation returns false.

If, after delaying window ordering, the receiver actually initiates a dragging session or similar operation, it should also send a preventWindowOrdering (page 117) message to NSApplication.sharedApplication(), which completely prevents the window from ordering forward and the activation from becoming active. preventWindowOrdering (page 117) is sent automatically by NSView's dragImage... and dragFile... methods.

shouldDrawColor

Returns false if the receiver is being drawn in an NSWindow (as opposed, for example, to being printed) and the NSWindow can't store color; otherwise returns true.

```
public boolean shouldDrawColor()
```

Discussion

An NSView can base its drawing behavior on the return value of this method to improve its appearance in grayscale windows.

See Also

drawRect (page 1753)

canStoreColor (page 1820) (NSWindow)

subviews

Return the receiver's immediate subviews.

```
public NSArray subviews()
```

Discussion

The order of the subviews may be considered as being back-to-front, but this does not imply invalidation and drawing behavior. The order is based on the order of the receiver's subviews as specified in the nib file from which they were unarchived or the programmatic interface for modifying the receiver's subview list. This ordering is also the reverse of the order in which hit-testing is done.

See Also

superview (page 1782)

addSubview (page 1739)

removeFromSuperview (page 1768)

superview

Returns the receiver's superview, or null if it has none.

```
public NSView superview()
```

Discussion

When applying this method iteratively or recursively, be sure to compare the returned NSView to the content view of the NSWindow to avoid proceeding out of the view hierarchy.

See Also

window (page 1787)

subviews (page 1782)

removeFromSuperview (page 1768)

tag

Returns the receiver's tag, an integer that you can use to identify view objects in your application.

```
public int tag()
```

Discussion

NSView's implementation returns –1. Subclasses can override this method to provide individual tags, possibly adding storage and a setTag method (which NSView doesn't define).

See Also

viewWithTag (page 1786)

toolTip

Returns the text for the view's tool tip.

```
public String toolTip()
```

Discussion

Returns null if the view doesn't currently display tool tip text.

See Also

setToolTip (page 1780)

translateOriginToPoint

Translates the receiver's coordinate system so that its origin moves to *newOrigin*.

```
public void translateOriginToPoint(NSPoint newOrigin)
```

Discussion

In the process, the origin of the receiver's bounds rectangle is shifted by (-*newOrigin*.x, -*newOrigin*.y). This method neither redisplays the receiver nor marks it as needing display. You must do this yourself with display (page 1747) or setNeedsDisplay (page 1779).

Note the difference between this method and setting the bounds origin. Translation effectively moves the image inside the bounds rectangle, while setting the bounds origin effectively moves the rectangle over the image. The two are in a sense inverse, although translation is cumulative, and setting the bounds origin is absolute.

This method posts a ViewBoundsDidChangeNotification (page 1788) to the default notification center if the receiver is configured to do so.

See Also

setBoundsOrigin (page 1774)

setBounds (page 1773)

setPostsBoundsChangedNotifications (page 1779)

unlockFocus

Balances an earlier lockFocus (page 1759) message; restoring the focus to the previously focused view is necessary.

```
public void unlockFocus()
```

Discussion

Throws an InvalidArgumentException if invoked on the wrong view.

See Also

allocateGState (page 1741)

unregisterDraggedTypes

Unregisters the receiver as a possible destination in a dragging session.

```
public void unregisterDraggedTypes()
```

See Also

`registerForDraggedTypes` (page 1768)

viewDidEndLiveResize

Informs the receiver of the end of a live resize.

```
public void viewDidEndLiveResize()
```

Discussion

In the simple case, a view is sent `viewWillStartLiveResize` (page 1785) before the first resize operation on the containing window and `viewDidEndLiveResize` after the last resize operation. A view that is repeatedly added and removed from a window during live resize will receive only one `viewWillStartLiveResize` (on the first time it is added to the window) and one `viewDidEndLiveResize` (when the window has completed the live resize operation). This allows a superview such as NSBrowser to add and remove its NSMatrix subviews during live resize without the NSMatrix receiving multiple calls to these methods.

A view might allocate data structures to cache-drawing information in `viewWillStartLiveResize` (page 1785) and should clean up these data structures in `viewDidEndLiveResize`. In addition, a view that does optimized drawing during live resize might want to do full drawing after `viewDidEndLiveResize`, although a view should not assume that it has a drawing context in `viewDidEndLiveResize` (since it may have been removed from the window during live resize). A view that wants to redraw itself after live resize should call `setNeedsDisplay(true)` in `viewDidEndLiveResize`.

A view subclass should call `super` from these methods.

See Also

`viewWillStartLiveResize` (page 1785)

`inLiveResize` (page 1756)

viewDidMoveToSuperview

Informs the receiver that its superview has changed (possibly to `null`).

```
public void viewDidMoveToSuperview()
```

Discussion

The default implementation does nothing; subclasses can override this method to perform whatever actions are necessary.

See Also

`viewDidMoveToWindow` (page 1785)

`viewWillMoveToSuperview` (page 1785)

`viewWillMoveToWindow` (page 1785)

viewDidMoveToWindow

Informs the receiver that it has been added to a new view hierarchy.

```
public void viewDidMoveToWindow()
```

Discussion

The default implementation does nothing; subclasses can override this method to perform whatever actions are necessary.

`window` (page 1787) may return `null` when this method is invoked, indicating that the receiver does not currently reside in any window. This occurs when the receiver has just been removed from its superview or when the receiver has just been added to a superview that does not itself have a window. Overrides of this method may choose to ignore such cases if they are not of interest.

See Also

`viewDidMoveToSuperview` (page 1784)
`viewWillMoveToSuperview` (page 1785)
`viewWillMoveToWindow` (page 1785)

viewWillMoveToSuperview

Informs the receiver that its superview is about to change to *`newSuperview`* (which may be `null`).

```
public void viewWillMoveToSuperview(NSView newSuperview)
```

Discussion

Subclasses can override this method to perform whatever actions are necessary.

See Also

`viewDidMoveToSuperview` (page 1784)
`viewDidMoveToWindow` (page 1785)
`viewWillMoveToWindow` (page 1785)

viewWillMoveToWindow

Informs the receiver that it's being added to the view hierarchy of *`newWindow`* (which may be `null`).

```
public void viewWillMoveToWindow(NSWindow newWindow)
```

Discussion

Subclasses can override this method to perform whatever actions are necessary.

See Also

`viewDidMoveToSuperview` (page 1784)
`viewDidMoveToWindow` (page 1785)
`viewWillMoveToSuperview` (page 1785)

viewWillStartLiveResize

Informs the receiver of the start of a live resize.

```
public void viewWillStartLiveResize()
```

Discussion

In the simple case, a view is sent `viewWillStartLiveResize` before the first resize operation on the containing window and `viewDidEndLiveResize` (page 1784) after the last resize operation. A view that is repeatedly added and removed from a window during live resize will receive only one `viewWillStartLiveResize` (on the first time it is added to the window) and one `viewDidEndLiveResize` (when the window has completed the live resize operation). This allows a superview such as NSBrowser to add and remove its NSMatrix subviews during live resize without the NSMatrix receiving multiple calls to these methods.

A view might allocate data structures to cache-drawing information in `viewWillStartLiveResize` and should clean up these data structures in `viewDidEndLiveResize` (page 1784). In addition, a view that does optimized drawing during live resize might want to do full drawing after `viewDidEndLiveResize`, although a view should not assume that it has a drawing context in `viewDidEndLiveResize` (since it may have been removed from the window during live resize). A view that wants to redraw itself after live resize should call `setNeedsDisplay(true)` in `viewDidEndLiveResize`.

A view subclass should call `super` from these methods.

See Also

`viewDidEndLiveResize` (page 1784)

`inLiveResize` (page 1756)

viewWithTag

Returns the receiver's nearest descendant (including itself) whose tag is *aTag*, or `null` if no subview has that tag.

```
public NSView viewWithTag(int aTag)
```

See Also

`tag` (page 1782)

visibleRect

Returns the portion of the receiver not clipped by its superviews.

```
public NSRect visibleRect()
```

Discussion

Visibility is therefore defined quite simply and doesn't account for whether other NSViews (or windows) overlap the receiver or whether the receiver has a window at all. This method returns `NSRect.ZeroRect` if the receiver is effectively hidden.

During a printing operation the visible rectangle is further clipped to the page being imaged.

See Also

`setHidden` (page 1778)

`isVisible` (page 1840) (NSWindow)

`documentVisibleRect` (page 1273) (NSScrollView)

`documentVisibleRect` (page 346) (NSClipView)

wantsDefaultClipping

Returns whether the Application Kit's default clipping provided to drawRect (page 1753) implementations is in effect.

```
public boolean wantsDefaultClipping()
```

Discussion

By default, this method returns true. Subclasses may override this method to return false if they want to suppress the default clipping. They may want to do this in situations where drawing performance is critical to avoid the cost of setting up, enforcing, and cleaning up the clip path

A view that overrides this method to refuse the default clipping must either set up whatever clipping it requires or constrain its drawing exactly to the list of rectangles returned by rectsBeingDrawn (page 1766). Failing to do so could result in corruption of other drawing in the view's window.

Availability

Available in Mac OS X v10.3 and later.

widthAdjustLimit

Returns the fraction (from 0.0 to 1.0) of the page that can be pushed onto the next page during automatic pagination to prevent items such as small images or text columns from being divided across pages.

```
public float widthAdjustLimit()
```

Discussion

This fraction is used to calculate the right edge limit for a adjustPageWidth (page 1740) message.

See Also

heightAdjustLimit (page 1755)

willRemoveSubview

Overridden by subclasses to perform additional actions before subviews are removed from the receiver.

```
public void willRemoveSubview(NSView subview)
```

Discussion

Invoked when *subview* receives a removeFromSuperview (page 1768) message or *subview* is removed from the receiver due to it being added to another view with addSubview (page 1739).

window

Returns the receiver's window object, or null if it has none.

```
public NSWindow window()
```

See Also

superview (page 1782)

writeEPSInsideRectToPasteboard

Writes EPS data that draws the region of the receiver within *aRect* onto *pboard*.

```
public void writeEPSInsideRectToPasteboard(NSRect aRect, NSPasteboard pboard)
```

See Also

dataWithEPSInsideRect (page 1746)

writePDFInsideRectToPasteboard

Writes PDF data that draws the region of the receiver within *aRect* onto *pboard*.

```
public void writePDFInsideRectToPasteboard(NSRect aRect, NSPasteboard pboard)
```

See Also

dataWithPDFInsideRect (page 1747)

Constants

NSView defines the following constants to be used when specifying a view's border:

Constant	Description
BezelBorder	A concave border that makes the view look sunken.
GrooveBorder	A thin border that looks etched around the image.
LineBorder	A black line border around the view.
NoBorder	No border.

Notifications

ViewBoundsDidChangeNotification

Posted whenever the NSView's bounds rectangle changes independently of the frame rectangle, if the NSView is configured using setPostsBoundsChangedNotifications (page 1779) to post such notifications.

The notification object is the NSView whose bounds rectangle has changed. This notification does not contain a *userInfo* dictionary.

The following methods can result in notification posting:

setBounds (page 1773)
setBoundsOrigin (page 1774)
setBoundsRotation (page 1775)
setBoundsSize (page 1775)

- `translateOriginToPoint` (page 1783)
- `scaleUnitSquareToSize` (page 1771)
- `rotateByAngle` (page 1770)

Note that the bounds rectangle resizes automatically to track the frame rectangle. Because the primary change is that of the frame rectangle, however, `setFrame` (page 1776) and `setFrameSize` (page 1777) don't result in a bounds-changed notification.

ViewFocusDidChangeNotification

Deprecated notification that was posted for an NSView and each of its descendents (recursively) whenever the frame or bounds geometry of the view changed.

In Mac OS X v10.4 and later, this notification is no longer posted. In earlier version of Mac OS X, use `NSViewBoundsDidChangeNotification` and `NSViewFrameDidChangeNotification` instead to get the same information provided by this notification.

The notification object is the NSView whose geometry changed. This notification does not contain a *`userInfo`* dictionary.

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

See Also
`ViewBoundsDidChangeNotification` (page 1788)
`ViewFrameDidChangeNotification` (page 1789)

ViewFrameDidChangeNotification

Posted whenever the NSView's frame rectangle changes, if the NSView is configured using `setPostsFrameChangedNotifications` (page 1780) to post such notifications.

The notification object is the NSView whose frame rectangle has changed. This notification does not contain a *`userInfo`* dictionary.

The following methods can result in notification posting:

- `setFrame` (page 1776)
- `setFrameOrigin` (page 1777)
- `setFrameRotation` (page 1777)
- `setFrameSize` (page 1777)

NSViewAnimation

Inherits from	NSAnimation
Package:	com.apple.cocoa.application
Availability	Available in Mac OS X v10.4 and later.
Companion guide	Drawing and Views

Overview

The NSViewAnimation class, a public subclass of NSAnimation, offers a convenient way to animate multiple views and windows. The animation effects you can achieve are limited to changes in frame location and size, and to fade-in and fade-out effects.

An NSViewAnimation object takes an array of dictionaries from which it determines the objects to animate and the effects to apply to them. Each dictionary must have a target object and, optionally, properties that specify beginning and ending frame and whether to fade in or fade out. (See "Constants" (page 1793) for further information.) Animations with NSViewAnimation are, by default, in non-blocking mode over a duration of 0.5 seconds using the ease in-out animation curve. But you can configure the animation to have any duration, curve, frame rate, and blocking mode. You may also set progress marks, assign a delegate, and implement delegation methods in order to animate view and windows concurrent with the ones specified as targets in the view-animation dictionary.

Tasks

Constructors

`NSViewAnimation` (page 1792)

Getting and Setting View-animation Dictionaries

`setViewAnimations` (page 1792)
Sets the dictionaries defining the objects to animate to *`viewAnimations`*.

`viewAnimations` (page 1793)
Returns the list of dictionaries defining the objects to animate.

Constructors

NSViewAnimation

```
public NSViewAnimation()
```

Discussion
Creates a default NSViewAnimation object. You must use the methods of this class to set the animation and view information.

```
public NSViewAnimation(double duration, int animationCurve)
```

Discussion
Creates an NSViewAnimation object and initializes it with the specified *duration* and *animationCurve* values. The *duration* parameter specifies the number of seconds over which the animation occurs; specifying a negative number raises an exception. You can change the duration later by calling the inherited `setDuration` method. The *animationCurve* parameter is a constant that describes the relative speed of the animation over its course; if it is zero, the default curve (AnimationEaseInOut) is used. See the constants defined in NSAnimation for descriptions of the possible values.

```
public NSViewAnimation(NSArray viewAnimations)
```

Discussion
Creates the NSViewAnimation object initialized with the dictionaries in *viewAnimations*. Each dictionary specifies a view or window to animate and the effect to apply. returns `null` if there was a problem initializing the object. The *viewAnimations* parameter can be `null` but you must later set the required array of dictionaries by calling `setViewAnimations` (page 1792) if you want to use the capabilities of the NSViewAnimation class. See "Constants" (page 1793) for a description of valid keys and values for dictionaries in *viewAnimations*.

Instance Methods

setViewAnimations

Sets the dictionaries defining the objects to animate to *viewAnimations*.

```
public void setViewAnimations(NSArray viewAnimations)
```

Discussion
Each dictionary in the passed-in array specifies a view or window to animate and the effect to apply. Pass in `null` to remove the current list of dictionaries. See "Constants" (page 1793) for a description of valid keys and values for dictionaries in *viewAnimations*.

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

See Also
`viewAnimations` (page 1793)

viewAnimations

Returns the list of dictionaries defining the objects to animate.

```
public NSArray viewAnimations()
```

Discussion
Each dictionary in the returned array specifies a view or window to animate and the effect to apply.

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

See Also
`setViewAnimations` (page 1792)

Constants

The following string constants are keys for the dictionaries in the array passed into `NSViewAnimation` (page 1792) and `setViewAnimations` (page 1792).

Constant (Key)	Description of value
`ViewAnimationTargetKey`	A target of the animation, which can be either an NSView object or an NSWindow object. This property is required.
`ViewAnimationStartFrameKey`	An NSRect structure encoded in an NSValue object that gives the size and location of the window or view at the start of the animation. This property is optional. If it is not specified, NSViewAnimation uses the frame of the window or view at the start of the animation.
`ViewAnimationEndFrameKey`	An NSRect structure encoded in an NSValue object that gives the size and location of the window or view at the end of the animation. This property is optional. If it is not specified, NSViewAnimation uses the frame of the window or view at the start of the animation. If the target is a view and the end frame is empty, the view is hidden at the end.
`ViewAnimationEffectKey`	Takes one of two string constants specifying fade-in or fade-out effects for the target: `ViewAnimationFadeInEffect` and `ViewAnimationFadeOutEffect`. If the target is a view and the effect is to fade out, the view is hidden at the end. If the effect is to fade in an initially hidden view and the end frame is non-empty, the view is unhidden at the end. If the target is a window, the window is ordered in or out as appropriate to the effect. This property is optional.
`ViewAnimationFadeInEffect`	Specifies a fade-in type of effect.
`ViewAnimationFadeOutEffect`	Specifies a fade-out type of effect.

NSWindow

Inherits from	NSResponder : NSObject
Implements	NSCoding (NSResponder)
Package:	com.apple.cocoa.application
Companion guide	Window Programming Guide for Cocoa

Class at a Glance

An NSWindow manages an onscreen window, coordinating the display and event handling for its NSViews. Interface Builder allows you to create and set up NSWindows, but there are many things you may wish to do programmatically as well.

Principal Attributes

- Manages a view hierarchy.
- Uses a delegate.
- Distributes events to view objects.
- Provides a field editor to view objects.

Interface Builder
Constructor
"NSWindow" (page 1813)

Commonly Used Methods

makeKeyAndOrderFront (page 1841)
Moves the NSWindow to the front and makes it the key window.

makeFirstResponder (page 1841)
Sets the first responder in the NSWindow.

fieldEditorForObject (page 1832)
Returns the shared text object for the NSWindow.

setContentView (page 1858)
Sets the root-level NSView in the NSWindow.

`representedFilename` (page 1849)
Returns the filename whose contents the NSWindow presents.

`setDocumentEdited` (page 1859)
Sets whether the NSWindow's represented file needs to be saved.

`setTitle` (page 1868)
Sets the title of the NSWindow.

`setTitleWithRepresentedFilename` (page 1868)
Sets the title of the NSWindow in a readable format for filenames.

Overview

The NSWindow class defines objects that manage and coordinate the windows an application displays on the screen. A single NSWindow object corresponds to at most one onscreen window. The two principal functions of NSWindow are to provide an area in which NSViews can be placed and to accept and distribute, to the appropriate NSViews, events the user instigates through actions with the mouse and keyboard.

Note: Although NSWindow inherits the NSCoding protocol from NSResponder, NSWindow does not support coding. Legacy support for archivers exists but its use is deprecated and may not work. Any attempt to archive or unarchive an NSWindow using a keyed coding object throws an `InvalidArgumentException`.

Tasks

Constructors

`NSWindow` (page 1813)
Creates a new NSWindow object, whose content rectangle is specified relative to the lower-left corner of the main screen.

Calculating Layout

`contentRectForFrameRect` (page 1814)
Returns the content rectangle used by an NSWindow with a frame rectangle of *`frameRect`* and a style mask of *`aStyle`*.

`frameRectForContentRect` (page 1814)
Returns the frame rectangle used by an NSWindow with a content rectangle of *`contentRect`* and a style mask of *`aStyle`*.

`minFrameWidthWithTitle` (page 1815)
Returns the minimum width an NSWindow's frame rectangle must have for it to display all of *`aTitle`*, given *`aStyle`* as its style mask.

`contentRectForFrameRect` (page 1823)
Returns the rectangle bounding the receiver's content view given the frame rectangle *`frameRect`*.

`frameRectForContentRect` (page 1835)
: Returns the receiver's frame rectangle given the rectangle bounding the content view *contentRect*.

Converting Coordinates

`convertBaseToScreen` (page 1824)
: Returns *aPoint* converted from the receiver's base coordinate system to the screen coordinate system.

`convertScreenToBase` (page 1824)
: Returns *aPoint* converted from the screen coordinate system to the receiver's base coordinate system.

`userSpaceScaleFactor` (page 1872)
: Returns the scale factor applied to the window.

Moving and Resizing

`frame` (page 1834)
: Returns the receiver's frame rectangle.

`setFrame` (page 1860)
: Sets the origin and size of the receiver's frame rectangle according to *frameRect*, thereby setting its position and size onscreen, and passes a `displayIfNeeded` (page 1829) message down the receiver's view hierarchy, thus redrawing all NSViews that need to be displayed, if *flag* is `true`.

`animationResizeTime` (page 1817)
: Subclasses can override this method to control the total time for the frame change.

`setFrameOrigin` (page 1861)
: Positions the lower-left corner of the receiver's frame rectangle at *aPoint* in screen coordinates.

`setFrameTopLeftPoint` (page 1861)
: Positions the top-left corner of the receiver's frame rectangle at *aPoint* in screen coordinates.

`setContentSize` (page 1857)
: Sets the size of the receiver's content view to *aSize*, which is expressed in the receiver's base coordinate system.

`cascadeTopLeftFromPoint` (page 1821)
: Positions the receiver's top left at *topLeftPoint*, unless *topLeftPoint* is `NSPoint.ZeroPoint` in which case the receiver is not moved except as needed to constrain to the visible screen.

`center` (page 1821)
: Sets the receiver's location to the center of the screen.

`resizeFlags` (page 1850)
: Valid only while the receiver is being resized, this method returns the flags field of the event record for the mouse-down event that initiated the resizing session.

`performZoom` (page 1847)
: This action method simulates the user clicking the zoom box by momentarily highlighting the button and then zooming the window.

`zoom` (page 1874)
: This action method toggles the size and location of the window between its standard state (provided by the application as the "best" size to display the window's data) and its user state (a new size and location the user may have set by moving or resizing the window).

`isZoomed` (page 1840)
: Returns whether the receiver is in a zoomed state.

`showsResizeIndicator` (page 1869)
: Returns whether the receiver's resize indicator is visible.

`setShowsResizeIndicator` (page 1868)
: Sets whether the receiver's resize indicator is visible to *`show`*.

`isMovableByWindowBackground` (page 1838)
: Returns `true` if the receiver is movable by clicking and dragging anywhere in its background, `false` if not.

`setMovableByWindowBackground` (page 1865)
: Sets whether the receiver is movable by clicking and dragging anywhere in its background.

`preservesContentDuringLiveResize` (page 1848)
: Returns `true` if the window tries to optimize live resize operations by preserving the content of views that have not moved; otherwise, returns `false`.

`setPreservesContentDuringLiveResize` (page 1866)
: If *`flag`* is `true`, the window optimizes live resize operations by invalidating only the view contents that changed; this is the default setting.

Constraining Window Size

`maxSize` (page 1842)
: Returns the maximum size to which the receiver's frame (including its title bar) can be sized either by the user or by the `setFrame...` methods other than `setFrame` (page 1860).

`minSize` (page 1843)
: Returns the minimum size to which the receiver's frame (including its title bar) can be sized either by the user or by the `setFrame...` methods other than `setFrame` (page 1860).

`setMaxSize` (page 1864)
: Sets the maximum size to which the receiver's frame (including its title bar) can be sized to *`aSize`*.

`setMinSize` (page 1865)
: Sets the minimum size to which the receiver's frame (including its title bar) can be sized to *`aSize`*.

`setAspectRatio` (page 1854)
: Sets the receiver's size aspect ratio to *`ratio`*, constraining the size of its frame rectangle to integral multiples of this size when the user resizes it.

`aspectRatio` (page 1817)
: Returns the receiver's size aspect ratio.

`setResizeIncrements` (page 1867)
: Restricts the user's ability to resize the receiver so the width and height change by multiples of *`increments`*`.width` and *`increments`*`.height` as the user resizes the window.

`resizeIncrements` (page 1851)
: Returns the receiver's resizing increments, which restrict the user's ability to resize it so that its width and height alter by integral multiples of `increments.width` and `increments.height` when the user resizes it.

`constrainFrameRectToScreen` (page 1822)
: Modifies and returns *`frameRect`* so that its top edge lies on *`aScreen`*.

Managing Content Size

`setContentAspectRatio` (page 1856)
: Sets the aspect ratio of the receiver's content view to *`ratio`*, constraining the dimensions of its content rectangle to integral multiples of that ratio when the user resizes it.

`contentAspectRatio` (page 1822)
: Returns the aspect ratio (height in relation to width) of the receiver's content view.

`setContentResizeIncrements` (page 1857)
: Sets the increments for both height and width by which the receiver's content view can be resized to *`increments`*.

`contentResizeIncrements` (page 1824)
: Returns the size of increments used during resizing of the receiver's content rectangle.

`setContentMaxSize` (page 1856)
: Sets the maximum size of the receiver's content view to *`size`*, which is expressed in the receiver's base coordinate system.

`contentMaxSize` (page 1823)
: Returns the maximum size of the receiver's content view.

`setContentMinSize` (page 1857)
: Sets the minimum size of the receiver's content view to *`size`*, which is expressed in the receiver's base coordinate system.

`contentMinSize` (page 1823)
: Returns the minimum size of the receiver's content view.

Saving the Frame to User Defaults

`removeFrameUsingName` (page 1815)
: Removes the frame data stored under *`name`* from the application's user defaults.

`saveFrameUsingName` (page 1851)
: Saves the receiver's frame rectangle in the user defaults system.

`setFrameUsingName` (page 1862)
: Sets the receiver's frame rectangle by reading the rectangle data stored in *`name`* from the defaults system.

`setFrameAutosaveName` (page 1860)
: Sets the name used to automatically save the receiver's frame rectangle in the defaults system to *`name`*.

`frameAutosaveName` (page 1834)
: Returns the name used to automatically save the receiver's frame rectangle data in the defaults system, as set through `setFrameAutosaveName` (page 1860).

`setFrameFromString` (page 1861)
Sets the receiver's frame rectangle from the string representation *aString*, a representation previously creating using `stringWithSavedFrame` (page 1870).

`stringWithSavedFrame` (page 1870)
Returns a string that represents the receiver's frame rectangle in a format that can be used with a later `setFrameFromString` (page 1861) message.

Ordering Windows

`orderBack` (page 1844)
This action method moves the receiver to the back of its level in the screen list, without changing either the key window or the main window.

`orderFront` (page 1844)
This action method moves the receiver to the front of its level in the screen list, without changing either the key window or the main window.

`orderFrontRegardless` (page 1845)
Moves the receiver to the front of its level, even if its application isn't active, but without changing either the key window or the main window.

`orderOut` (page 1845)
This action method takes the receiver out of the screen list.

`orderWindow` (page 1845)
Repositions the receiver's window device in the window server's screen list.

`setLevel` (page 1863)
Sets the receiver's window level to *newLevel*.

`level` (page 1841)
Returns the level of the receiver as set using `setLevel` (page 1863).

`isVisible` (page 1840)
Returns `true` if the receiver is onscreen (even if it's obscured by other windows).

Attached Windows

`addChildWindow` (page 1816)
childWin is ordered either above (`Above`) or below (`Below`) the receiver, and maintained in that relative *place* for subsequent ordering operations involving either window.

`removeChildWindow` (page 1849)
Detaches *childWin* from the receiver.

`childWindows` (page 1821)
Returns an array of the receiver's attached child windows.

`parentWindow` (page 1846)
Returns the parent window to which the receiver is attached as a child.

`setParentWindow` (page 1866)
For use by subclasses when setting the parent window in the receiver.

Making Key and Main Windows

`becomeKeyWindow` (page 1819)
Invoked automatically to inform the receiver that it has become the key window; never invoke this method directly.

`canBecomeKeyWindow` (page 1819)
Returns `true` if the receiver can become the key window, `false` if it can't.

`isKeyWindow` (page 1838)
Returns `true` if the receiver is the key window for the application, `false` if it isn't.

`makeKeyAndOrderFront` (page 1841)
This action method moves the receiver to the front of the screen list, within its level, and makes it the key window.

`makeKeyWindow` (page 1842)
Makes the receiver the key window.

`resignKeyWindow` (page 1850)
Never invoke this method; it's invoked automatically when the NSWindow resigns key window status.

`becomeMainWindow` (page 1819)
Invoked automatically to inform the receiver that it has become the main window; never invoke this method directly.

`canBecomeMainWindow` (page 1820)
Returns `true` if the receiver can become the main window, `false` if it can't.

`isMainWindow` (page 1838)
Returns `true` if the receiver is the main window for the application, `false` if it isn't.

`makeMainWindow` (page 1842)

`resignMainWindow` (page 1850)
Never invoke this method; it's invoked automatically when the NSWindow resigns main window status.

`autorecalculatesKeyViewLoop` (page 1818)
Returns `true` if the window automatically recalculates the key view loop when views are added; otherwise returns `false`.

`recalculateKeyViewLoop` (page 1848)
Marks the key view loop as dirty and in need of recalculation.

`setAutorecalculatesKeyViewLoop` (page 1855)
If *`flag`* is `true`, the window recalculates the key view loop automatically when views are added or removed.

Working with the Default Button

`defaultButtonCell` (page 1825)
Returns the button cell that performs as if clicked when the NSWindow receives a Return (or Enter) key event.

`setDefaultButtonCell` (page 1858)
Makes the key equivalent of *`aButtonCell`* the Return (or Enter) key, so when the user presses Return that button performs as if clicked.

`disableKeyEquivalentForDefaultButtonCell` (page 1828)
: Disables the default button cell's key equivalent, so it doesn't perform a click when the user presses Return (or Enter).

`enableKeyEquivalentForDefaultButtonCell` (page 1832)
: Reenables the default button cell's key equivalent, so it performs a click when the user presses Return (or Enter).

Display and Drawing

`display` (page 1829)
: Passes a display message down the receiver's view hierarchy, thus redrawing all NSViews within the receiver, including the frame view that draws the border, title bar, and other peripheral elements.

`displayIfNeeded` (page 1829)
: Passes a `displayIfNeeded` (page 1829) message down the receiver's view hierarchy, thus redrawing all NSViews that need to be displayed, including the frame view that draws the border, title bar, and other peripheral elements.

`setViewsNeedDisplay` (page 1869)
: Sets whether the receiver's views need display (`true`) or do not need display (`false`) to *`flag`*.

`viewsNeedDisplay` (page 1873)
: Returns `true` if any of the receiver's NSView's need to be displayed, `false` otherwise.

`useOptimizedDrawing` (page 1872)
: Informs the receiver whether to optimize focusing and drawing when displaying its NSViews.

`setAutodisplay` (page 1855)
: Sets whether the receiver automatically displays its views that are marked as needing it.

`isAutodisplay` (page 1837)
: Returns `true` if the receiver automatically displays its views that are marked as needing it, `false` if it doesn't.

`update` (page 1871)
: Updates the window.

`graphicsContext` (page 1835)
: Returns the graphics context associated with the receiver for the current thread.

`disableScreenUpdatesUntilFlush` (page 1828)
: Disables the receiver's screen updates until the window is flushed.

Flushing Graphics

`flushWindow` (page 1833)
: Flushes the receiver's offscreen buffer to the screen if the receiver is buffered and flushing is enabled.

`flushWindowIfNeeded` (page 1834)
: Flushes the receiver's offscreen buffer to the screen if flushing is enabled and if the last `flushWindow` (page 1833) message had no effect because flushing was disabled.

`enableFlushWindow` (page 1832)
: Reenables the `flushWindow` (page 1833) method for the receiver after it was disabled through a previous `disableFlushWindow` (page 1827) message.

`disableFlushWindow` (page 1827)
Disables the `flushWindow` (page 1833) method for the receiver.

`isFlushWindowDisabled` (page 1838)
Returns `true` if the receiver's flushing ability has been disabled; otherwise returns `false`.

Bracketing Temporary Drawing

`cacheImageInRect` (page 1819)
Stores the receiver's raster image from *`aRect`*, which is expressed in the receiver's base coordinate system.

`restoreCachedImage` (page 1851)
Splices the receiver's cached image rectangles, if any, back into its raster image (and buffer if it has one), undoing the effect of any drawing performed within those areas since they were established using `cacheImageInRect` (page 1819).

`discardCachedImage` (page 1828)
Discards all of the receiver's cached image rectangles.

Window Server Information

`windowNumber` (page 1873)
Returns the window number of the receiver's window device.

`gState` (page 1835)
Returns the graphics state object associated with the receiver.

`deviceDescription` (page 1826)
Returns a dictionary containing information about the receiver's resolution, color depth, and so on.

`setBackingType` (page 1855)
Sets the receiver's backing store type to *`backingType`*.

`backingType` (page 1818)
Returns the receiver's backing store type.

`setOneShot` (page 1865)
Sets whether the window device that the receiver manages should be freed when it's removed from the screen list (and another one created if it's returned to the screen) to *`flag`*.

`isOneShot` (page 1839)
Returns `true` if the window device the receiver manages is freed when it's removed from the screen list, `false` if not.

`defaultDepthLimit` (page 1814)
Returns the default depth limit for instances of NSWindow.

`setDepthLimit` (page 1858)
Sets the depth limit of the receiver to *`limit`*.

`depthLimit` (page 1826)
Returns the depth limit of the receiver.

`setDynamicDepthLimit` (page 1859)
Sets whether the receiver changes its depth to match the depth of the screen it's on, or the depth of the deepest screen when it spans multiple screens.

`hasDynamicDepthLimit` (page 1835)
: Returns `true` if the receiver's depth limit can change to match the depth of the screen it's on, `false` if it can't.

`canStoreColor` (page 1820)
: Returns `true` if the receiver has a depth limit that allows it to store color values, `false` if it doesn't.

Screen Information

`deepestScreen` (page 1825)
: Returns the deepest screen the receiver is on (it may be split over several screens), or `null` if the receiver is offscreen.

`screen` (page 1852)
: Returns the screen the receiver is on.

`displaysWhenScreenProfileChanges` (page 1829)
: Returns `true` if the window context should be updated when the screen profile changes or when the window moves to a different screen.

`setDisplaysWhenScreenProfileChanges` (page 1859)
: Sets whether the window context should be updated when the screen profile changes.

Working with the Responder Chain

`makeFirstResponder` (page 1841)
: Attempts to make *`aResponder`* the first responder for the receiver.

`firstResponder` (page 1833)
: Returns the receiver's first responder.

Event Handling

`currentEvent` (page 1824)
: Returns the event currently being processed by the application, by invoking NSApplication's `currentEvent` (page 110) method.

`nextEventMatchingMask` (page 1844)
: Invokes NSApplication's `nextEventMatchingMask` (page 115) method, using *`mask`* as the first argument, with an unlimited expiration, a mode of `NSApplication.EventTrackingRunLoopMode`, and a dequeue flag of `true`.

`discardEventsMatchingMask` (page 1829)
: Forwards the message to the NSApplication object, which handles it as described in the NSApplication class specification.

`postEvent` (page 1847)
: Forwards the message to the global NSApplication object, `NSApplication.sharedApplication()`.

`sendEvent` (page 1853)
: This action method dispatches mouse and keyboard events, specified by *`theEvent`*, sent to the receiver by the NSApplication object.

`tryToPerform` (page 1871)
: Dispatches action messages with *anObject* as the argument.

`keyDown` (page 1840)
: Handles *theEvent* keyboard event that may need to be interpreted as changing the key view or triggering a keyboard equivalent.

`mouseLocationOutsideOfEventStream` (page 1843)
: Returns the current location of the mouse reckoned in the receiver's base coordinate system, regardless of the current event being handled or of any events pending.

`setAcceptsMouseMovedEvents` (page 1853)
: Sets whether the receiver accepts mouse-moved events and distributes them to its responders.

`acceptsMouseMovedEvents` (page 1816)
: Returns `true` if the receiver accepts and distributes mouse-moved events, `false` if it doesn't.

`ignoresMouseEvents` (page 1836)
: Return whether the receiver is transparent to mouse events.

`setIgnoresMouseEvents` (page 1863)
: Specifies whether the receiver is transparent to mouse clicks and other mouse events, allowing overlay windows.

Working with the Field Editor

`fieldEditorForObject` (page 1832)
: Returns the receiver's field editor, creating it if needed if *createFlag* is `true`.

`endEditingForObject` (page 1832)
: Forces the field editor, which *anObject* is assumed to be using, to give up its first responder status and prepares it for its next assignment.

Keyboard Interface Control

`setInitialFirstResponder` (page 1863)
: Sets *aView* as the NSView that's made first responder (also called the key view) the first time the receiver is placed onscreen.

`initialFirstResponder` (page 1836)
: Returns the NSView that's made first responder the first time the receiver is placed onscreen.

`selectKeyViewFollowingView` (page 1852)
: Sends the NSView message `nextValidKeyView` (page 1762) to *aView*, and if that message returns an NSView, invokes `makeFirstResponder` (page 1841) with the returned NSView.

`selectKeyViewPrecedingView` (page 1852)
: Sends the NSView message `previousValidKeyView` (page 1765) to *aView*, and if that message returns an NSView, invokes `makeFirstResponder` (page 1841) with the returned NSView.

`selectNextKeyView` (page 1852)
: This action method searches for a candidate key view and, if it finds one, invokes `makeFirstResponder` (page 1841) to establish it as the first responder.

`selectPreviousKeyView` (page 1853)
: This action method searches for a candidate key view and, if it finds one, invokes `makeFirstResponder` (page 1841) to establish it as the first responder.

`keyViewSelectionDirection` (page 1840)
: Returns the direction the receiver is currently using to change the key view.

Setting the Title and Filename

`setTitle` (page 1868)
: Sets the string that appears in the receiver's title bar (if it has one) to *`aString`* and displays the title.

`setTitleWithRepresentedFilename` (page 1868)
: Sets *`path`* as the receiver's title, formatting it as a file-system path, and records *`path`* as the receiver's associated filename using `setRepresentedFilename` (page 1867).

`title` (page 1870)
: Returns either the string that appears in the title bar of the receiver, or the path to the represented file.

`setRepresentedFilename` (page 1867)
: Sets the name of the file the receiver represents to *`path`*.

`representedFilename` (page 1849)
: Returns the name of the file the receiver represents.

Marking a Window Edited

`setDocumentEdited` (page 1859)
: Sets whether the receiver's document has been edited and not saved to *`flag`*.

`isDocumentEdited` (page 1837)
: Returns `true` or `false` according to the argument supplied with the last `setDocumentEdited` (page 1859) message.

Closing the Window

`close` (page 1821)
: Removes the receiver from the screen.

`performClose` (page 1846)
: This action method simulates the user clicking the close button by momentarily highlighting the button and then closing the window.

`setReleasedWhenClosed` (page 1867)
: Sets whether the receiver is merely hidden (`false`) or hidden and then released (`true`) when it receives a `close` message.

`isReleasedWhenClosed` (page 1839)
: Returns `true` if the receiver is automatically released after being closed, `false` if it's simply removed from the screen.

Miniaturizing and Miniaturized Windows

miniaturize (page 1842)
: This action method removes the receiver from the screen list and displays the miniaturized window in the dock.

performMiniaturize (page 1847)
: This action method simulates the user clicking the miniaturize button by momentarily highlighting the button, then miniaturizing the window.

deminiaturize (page 1826)
: This action method deminiaturizes the receiver.

isMiniaturized (page 1838)
: Returns `true` if the receiver has been miniaturized, `false` if it hasn't.

setMiniwindowImage (page 1864)
: Sets the receiver's custom miniaturized window image to *anImage*.

miniwindowImage (page 1842)
: Returns the custom miniaturized window image of the receiver.

setMiniwindowTitle (page 1864)
: Sets the title of the receiver's miniaturized counterpart to *aString* and redisplays it.

miniwindowTitle (page 1843)
: Returns the title displayed in the receiver's miniaturized window.

Working with Menus

menuChanged (page 1815)
: This method does nothing; it is here for backward compatibility.

Working with the Windows Menu

setExcludedFromWindowsMenu (page 1860)
: Sets whether the receiver's title is omitted from the application's Windows menu.

isExcludedFromWindowsMenu (page 1837)
: Returns `true` if the receiver's title is omitted from the application's Windows menu, `false` if it is listed.

Working with Cursor Rectangles

areCursorRectsEnabled (page 1817)
: Returns `true` if the receiver's cursor rectangles are enabled, `false` if they're not.

enableCursorRects (page 1831)
: Reenables cursor rectangle management within the receiver after a disableCursorRects (page 1827) message.

disableCursorRects (page 1827)
: Disables all cursor rectangle management within the receiver.

`discardCursorRects` (page 1828)
  Invalidates all cursor rectangles in the receiver.

`invalidateCursorRectsForView` (page 1836)
  Marks as invalid the cursor rectangles of *aView*, an NSView in the receiver's view hierarchy, so they'll be set up again when the receiver becomes key (or immediately if the receiver is key).

`resetCursorRects` (page 1849)
  Invokes `discardCursorRects` (page 1828) to clear the receiver's cursor rectangles, then sends `resetCursorRects` (page 1849) to every NSView in the receiver's view hierarchy.

Dragging

`concludeDragOperation` (page 1822)

`dragImage` (page 1831)
  Begins a dragging session.

`draggingEntered` (page 1830)

`draggingExited` (page 1830)
  Invoked when the dragged image exits the receiver's frame rectangle.

`draggingUpdated` (page 1831)

`performDragOperation` (page 1846)

`prepareForDragOperation` (page 1847)

`registerForDraggedTypes` (page 1849)
  Registers *pboardTypes* as the pasteboard types the receiver will accept as the destination of an image-dragging session.

`unregisterDraggedTypes` (page 1871)
  Unregisters the receiver as a possible destination for dragging operations.

Controlling Behavior

`setHidesOnDeactivate` (page 1862)
  Sets whether the receiver is removed from the screen when the application is inactive.

`hidesOnDeactivate` (page 1836)
  Returns `true` if the receiver is removed from the screen when its application is deactivated, `false` if it remains onscreen.

`worksWhenModal` (page 1874)
  Returns `true` if the receiver is able to receive keyboard and mouse events even when some other window is being run modally, `false` otherwise.

`setCanHide` (page 1856)
  Sets whether the receiver can be hidden during NSApplication's `hide` (page 112) to *flag*.

`canHide` (page 1820)
Returns whether the receiver can be hidden during NSApplication’s `hide` (page 112).

Working with Display Characteristics

`setContentView` (page 1858)
Makes *aView* the receiver’s content view; the previous content view is removed from the receiver’s view hierarchy.

`contentView` (page 1824)
Returns the receivers’s content view, the highest accessible NSView object in the receiver’s view hierarchy.

`setBackgroundColor` (page 1855)
Sets the receiver’s background color to *aColor*.

`backgroundColor` (page 1818)
Returns the color of the receiver’s background.

`styleMask` (page 1870)
Returns the receiver’s style mask, indicating what kinds of control items it displays.

`setHasShadow` (page 1862)
Sets whether the receiver has a shadow to *hasShadow*.

`hasShadow` (page 1836)
Returns `true` if the window has a shadow; otherwise returns `false`.

`invalidateShadow` (page 1837)
Invalidates the window shadow so that it is recomputed based on the current window shape.

`setAlphaValue` (page 1854)
Applies *windowAlpha* to the entire window.

`alphaValue` (page 1817)
Returns the receiver’s alpha value.

`setOpaque` (page 1866)
Sets whether the receiver is opaque to *isOpaque*.

`isOpaque` (page 1839)
Returns whether the receiver is opaque.

Working with Services

`validRequestorForTypes` (page 1872)
Searches for an object that responds to a Services request by providing input of *sendType* and accepting output of *returnType*.

Printing

`print` (page 1848)
This action method runs the Print panel, and if the user chooses an option other than canceling, prints the receiver (its frame view and all subviews).

`dataWithEPSInsideRect` (page 1825)
Returns EPS data that draws the region of the receiver within *`aRect`* (expressed in the receiver's base coordinate system).

`dataWithPDFInsideRect` (page 1825)
Returns PDF data that draws the region of the receiver within *`aRect`* (expressed in the receiver's base coordinate system).

Setting the Delegate

`setDelegate` (page 1858)
Makes *`anObject`* the receiver's delegate, without retaining it.

`delegate` (page 1826)
Returns the receiver's delegate, or `null` if it doesn't have a delegate.

Getting Associated Information

`drawers` (page 1831)
Returns the collection of drawers associated with the receiver.

`setWindowController` (page 1869)
Set's the receiver's window controller to be *`windowController`*.

`windowController` (page 1873)
Returns the receiver's window controller.

Working with Sheets

`attachedSheet` (page 1818)
Returns the sheet attached to the receiver.

`isSheet` (page 1840)
Returns `true` if the receiver has ever run as a modal sheet.

Working with Toolbars

`setToolbar` (page 1869)
Sets the receiver's toolbar to *`toolbar`*.

`toolbar` (page 1871)
Returns the receiver's toolbar.

`toggleToolbarShown` (page 1871)
The action method for the "Hide Toolbar" menu item (which alternates with "Show Toolbar").

`runToolbarCustomizationPalette` (page 1851)
The action method for the "Customize Toolbar…" menu item.

Working with Title Bar Widgets

`standardWindowButtonForStyleMask` (page 1815)
Returns a new instance of the given standard *button*, sized appropriately for the *styleMask*.

`standardWindowButton` (page 1870)
Return the given standard *button* if it is in the window view hierarchy.

`setShowsToolbarButton` (page 1868)
If *flag* is `true`, the window title bar is updated to display the standard toolbar button.

`showsToolbarButton` (page 1870)
Returns `true` if the standard toolbar button is currently displayed; otherwise, returns `false`.

Managing Tool Tips

`setAllowsToolTipsWhenApplicationIsInactive` (page 1854)
Sets whether the receiver can display tool tips even when the application is in the background.

`allowsToolTipsWhenApplicationIsInactive` (page 1816)
Returns whether the receiver can display tool tips even when the application is in the background.

Working with window status

`windowDidBecomeKey` (page 1877) *delegate method*
Sent by the default notification center immediately after an NSWindow has become key.

`windowDidBecomeMain` (page 1877) *delegate method*
Sent by the default notification center immediately after an NSWindow has become main.

`windowDidResignKey` (page 1879) *delegate method*
Sent by the default notification center immediately after an NSWindow has resigned its status as key window.

`windowDidResignMain` (page 1879) *delegate method*
Sent by the default notification center immediately after an NSWindow has resigned its status as main window.

Moving and resizing windows

`windowDidChangeScreen` (page 1877) *delegate method*
Sent by the default notification center immediately after an NSWindow has changed screens.

`windowDidChangeScreenProfile` (page 1878) *delegate method*
Sent by the default notification center immediately after an NSWindow has changed screen display profiles.

`windowWillMove` (page 1881) *delegate method*
Sent by the default notification center immediately before an NSWindow is moved.

`windowDidMove` (page 1879) *delegate method*
Sent by the default notification center immediately after an NSWindow has been moved.

`windowWillResize` (page 1881) *delegate method*
Invoked when *sender* is being resized (whether by the user or through one of the `setFrame...` methods other than `setFrame` (page 1860)).

`windowDidResize` (page 1879) *delegate method*
Sent by the default notification center immediately after an NSWindow has been resized.

`windowShouldZoom` (page 1880) *delegate method*
Invoked just before *sender* is zoomed.

`windowWillUseStandardFrame` (page 1882) *delegate method*
Invoked by the `zoom` (page 1874) method while determining a frame the *sender* may be zoomed to.

Miniaturizing and closing windows

`windowWillMiniaturize` (page 1880) *delegate method*
Sent by the default notification center immediately before an NSWindow is miniaturized.

`windowDidMiniaturize` (page 1878) *delegate method*
Sent by the default notification center immediately after an NSWindow has been miniaturized.

`windowDidDeminiaturize` (page 1878) *delegate method*
Sent by the default notification center immediately after an NSWindow has been deminiaturized.

`windowShouldClose` (page 1880) *delegate method*
Invoked when the user attempts to close the window or when the NSWindow receives a `performClose` (page 1846) message.

`windowWillClose` (page 1880) *delegate method*
Sent by the default notification center immediately before an NSWindow closes.

Exposing and updating windows

`windowDidExpose` (page 1878) *delegate method*
Sent by the default notification center immediately after an NSWindow has been exposed.

`windowDidUpdate` (page 1879) *delegate method*
Sent by the default notification center immediately after an NSWindow receives an `update` (page 1871) message.

Displaying sheets

`windowWillBeginSheet` (page 1880) *delegate method*
Sent by the default notification center immediately before an NSWindow opens a sheet.

`windowDidEndSheet` (page 1878) *delegate method*
Sent by the default notification center immediately after an NSWindow closes a sheet.

`windowWillPositionSheet` (page 1881) *delegate method*
Sent to the delegate just before the animation of a sheet, giving it the opportunity to return a custom location for the attachment of the sheet (*sheet*) to the window (*window*).

Obtaining information about a window

`windowWillReturnFieldEditor` (page 1882) *delegate method*
Invoked when the field editor of *sender* is requested by *anObject*.

`windowWillReturnUndoManager` (page 1882) *delegate method*
Invoked when the undo manager for *sender* is requested.

Constructors

NSWindow

Creates a new NSWindow object, whose content rectangle is specified relative to the lower-left corner of the main screen.

```
public NSWindow()
```

Discussion
This constructor calls the following constructor with *contentRect* of (100.0, 100.0, 100.0, 100.0), *styleMask* of `TitledWindowMask`, *backingType* of `BackingStoreBuffered`, and *defer* set to `false`.

Creates a new NSWindow object, whose content rectangle is specified relative to the lower-left corner of the main screen.

```
public NSWindow(NSRect contentRect, int styleMask, int backingType, boolean defer)
```

Discussion
The *contentRect* argument specifies the location and size of the NSWindow's content area in screen coordinates. Note that the window server limits window position coordinates to ±16,000 and sizes to 10,000.

The *styleMask* argument specifies the receiver's style. Either it can be `BorderlessWindowMask`, or it can contain any of the options described in the constants section, combined using the C bitwise OR operator.

Borderless windows display none of the usual peripheral elements and are generally useful only for display or caching purposes; you should normally not need to create them. Also, note that an NSWindow's style mask should include `TitledWindowMask` if it includes any of the others.

The *backingType* argument specifies how the drawing done in the receiver is buffered by the object's window device, and possible values are described in the constants section.

The *defer* argument determines whether the window server creates a window device for the new object immediately. If *defer* is `true`, it defers creating the window until the receiver is moved onscreen. All display messages sent to the NSWindow or its NSViews are postponed until the window is created, just before it's moved onscreen. Deferring the creation of the window improves launch time and minimizes the virtual memory load on the window server.

The new NSWindow creates an instance of NSView to be its default content view. You can replace it with your own object by using the `setContentView` (page 1858) method.

Creates an NSWindow object, whose content rectangle is specified relative to the lower-left corner of *aScreen*.

```
public NSWindow(NSRect contentRect, int styleMask, int bufferingType, boolean defer,
     NSScreen aScreen)
```

Discussion
Otherwise this method is equivalent to the preceding constructor.

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.

See Also
`orderFront` (page 1844)
`setTitle` (page 1868)
`setOneShot` (page 1865)

Static Methods

contentRectForFrameRect

Returns the content rectangle used by an NSWindow with a frame rectangle of *frameRect* and a style mask of *aStyle*.

```
public static NSRect contentRectForFrameRect(NSRect frameRect, int aStyle)
```

Discussion
Both *frameRect* and the returned content rectangle are expressed in screen coordinates. See the constants section for a list of style mask values.

See Also
`"frameRectForContentRect"` (page 1814)
`contentRectForFrameRect` (page 1823)

defaultDepthLimit

Returns the default depth limit for instances of NSWindow.

```
public static int defaultDepthLimit()
```

Discussion
This limit is determined by the depth of the deepest screen level available to the window server.

See Also
`setDepthLimit` (page 1858)
`setDynamicDepthLimit` (page 1859)
`canStoreColor` (page 1820)

frameRectForContentRect

Returns the frame rectangle used by an NSWindow with a content rectangle of *contentRect* and a style mask of *aStyle*.

```
public static NSRect frameRectForContentRect(NSRect contentRect, int aStyle)
```

Discussion

Both `contentRect` and the returned frame rectangle are expressed in screen coordinates. See the constants section for a list of style mask values.

See Also

`contentRectForFrameRect` (page 1814)
`frameRectForContentRect` (page 1835)

menuChanged

This method does nothing; it is here for backward compatibility.

```
public static void menuChanged(NSMenu aMenu)
```

See Also

`menu` (page 1192) (NSResponder)

minFrameWidthWithTitle

Returns the minimum width an NSWindow's frame rectangle must have for it to display all of `aTitle`, given `aStyle` as its style mask.

```
public static float minFrameWidthWithTitle(String aTitle, int aStyle)
```

Discussion

See the constants section for a list of acceptable style mask values.

removeFrameUsingName

Removes the frame data stored under `name` from the application's user defaults.

```
public static void removeFrameUsingName(String name)
```

See Also

`setFrameUsingName` (page 1862)
`setFrameAutosaveName` (page 1860)

standardWindowButtonForStyleMask

Returns a new instance of the given standard `button`, sized appropriately for the `styleMask`.

```
public static NSButton standardWindowButtonForStyleMask(int button, int styleMask)
```

Discussion

The caller is responsible for adding the button to the view hierarchy and for setting the target to be the window.

Availability
Available in Mac OS X v10.2 and later.

See Also
`standardWindowButton` (page 1870)

Instance Methods

acceptsMouseMovedEvents

Returns `true` if the receiver accepts and distributes mouse-moved events, `false` if it doesn't.

```
public boolean acceptsMouseMovedEvents()
```

Discussion
NSWindows by default don't accept mouse-moved events.

See Also
`setAcceptsMouseMovedEvents` (page 1853)

addChildWindow

`childWin` is ordered either above (`Above`) or below (`Below`) the receiver, and maintained in that relative *`place`* for subsequent ordering operations involving either window.

```
public void addChildWindow(NSWindow childWin, int place)
```

Discussion
While this attachment is active, moving *`childWin`* will not cause the receiver to move (as in sliding a drawer in or out), but moving the receiver will cause *`childWin`* to move.

Note that you should not create cycles between parent and child windows. For example, you should not add window B as child of window A, then add window A as a child of window B.

Availability
Available in Mac OS X v10.2 and later.

See Also
`removeChildWindow` (page 1849)
`childWindows` (page 1821)
`parentWindow` (page 1846)
`setParentWindow` (page 1866)

allowsToolTipsWhenApplicationIsInactive

Returns whether the receiver can display tool tips even when the application is in the background.

```
public boolean allowsToolTipsWhenApplicationIsInactive()
```

Discussion

Default is `false`.

Availability

Available in Mac OS X v10.3 and later.

See Also

`setAllowsToolTipsWhenApplicationIsInactive` (page 1854)

alphaValue

Returns the receiver's alpha value.

```
public float alphaValue()
```

See Also

`setAlphaValue` (page 1854)

animationResizeTime

Subclasses can override this method to control the total time for the frame change.

```
public double animationResizeTime(NSRect newFrame)
```

Discussion

newFrame is the rect passed into `setFrame` (page 1860).

The default implementation uses the value from the `NSWindowResizeTime` user default as the time in seconds to resize by 150 pixels. If this value is unspecified, `NSWindowResizeTime` is 0.20 seconds (this default value may be differ in different releases of Mac OS X).

areCursorRectsEnabled

Returns `true` if the receiver's cursor rectangles are enabled, `false` if they're not.

```
public boolean areCursorRectsEnabled()
```

See Also

`enableCursorRects` (page 1831)
`addCursorRect` (page 1739) (NSView)

aspectRatio

Returns the receiver's size aspect ratio.

```
public NSSize aspectRatio()
```

Discussion

The size of the receiver's frame rectangle is constrained to integral multiples of this ratio when the user resizes it. You can set an NSWindow's size to any ratio programmatically.

See Also

`resizeIncrements` (page 1851)

`setAspectRatio` (page 1854)

`setFrame` (page 1860)

attachedSheet

Returns the sheet attached to the receiver.

```
public NSWindow attachedSheet()
```

Discussion

If the receiver does not have a sheet attached, this method returns `null`.

autorecalculatesKeyViewLoop

Returns `true` if the window automatically recalculates the key view loop when views are added; otherwise returns `false`.

```
public boolean autorecalculatesKeyViewLoop()
```

Availability

Available in Mac OS X v10.4 and later.

See Also

`recalculateKeyViewLoop` (page 1848)

`setAutorecalculatesKeyViewLoop` (page 1855)

backgroundColor

Returns the color of the receiver's background.

```
public NSColor backgroundColor()
```

See Also

`setBackgroundColor` (page 1855)

backingType

Returns the receiver's backing store type.

```
public int backingType()
```

Discussion

The possible return values are described in the constants section.

See Also

`setBackingType` (page 1855)

becomeKeyWindow

Invoked automatically to inform the receiver that it has become the key window; never invoke this method directly.

```
public void becomeKeyWindow()
```

Discussion

This method reestablishes the receiver's first responder, sends the `becomeKeyWindow` message to that object if it responds, and posts a `WindowDidBecomeKeyNotification` (page 1883) to the default notification center.

See Also

`makeKeyWindow` (page 1842)
`makeKeyAndOrderFront` (page 1841)
`becomeMainWindow` (page 1819)

becomeMainWindow

Invoked automatically to inform the receiver that it has become the main window; never invoke this method directly.

```
public void becomeMainWindow()
```

Discussion

This method posts a `WindowDidBecomeMainNotification` (page 1883) to the default notification center.

See Also

`makeMainWindow` (page 1842)
`becomeKeyWindow` (page 1819)

cacheImageInRect

Stores the receiver's raster image from *aRect*, which is expressed in the receiver's base coordinate system.

```
public void cacheImageInRect(NSRect aRect)
```

Discussion

This method allows the receiver to perform temporary drawing, such as a band around the selection as the user drags the mouse, and to quickly restore the previous image by invoking `restoreCachedImage` (page 1851) and `flushWindowIfNeeded` (page 1834). The next time the window displays, it discards its cached image rectangles. You can also explicitly use `discardCachedImage` (page 1828) to free the memory occupied by cached image rectangles. *aRect* is made integral before caching the image to avoid antialiasing artifacts.

Only the last cached rectangle is remembered and can be restored.

See Also

`display` (page 1829)

canBecomeKeyWindow

Returns `true` if the receiver can become the key window, `false` if it can't.

```
public boolean canBecomeKeyWindow()
```

Discussion
Attempts to make the receiver the key window are abandoned if this method returns `false`. NSWindow's implementation returns `true` if the receiver has a title bar or a resize bar, `false` otherwise.

See Also
`isKeyWindow` (page 1838)
`makeKeyWindow` (page 1842)

canBecomeMainWindow

Returns `true` if the receiver can become the main window, `false` if it can't.

```
public boolean canBecomeMainWindow()
```

Discussion
Attempts to make the receiver the main window are abandoned if this method returns `false`. NSWindow's implementation returns `true` if the receiver is visible, is not an NSPanel, and has a title bar or a resize mechanism. Otherwise it returns `false`.

See Also
`isMainWindow` (page 1838)
`makeMainWindow` (page 1842)

canHide

Returns whether the receiver can be hidden during NSApplication's `hide` (page 112).

```
public boolean canHide()
```

Discussion
The default is `true`.

See Also
`setCanHide` (page 1856)

canStoreColor

Returns `true` if the receiver has a depth limit that allows it to store color values, `false` if it doesn't.

```
public boolean canStoreColor()
```

See Also
`depthLimit` (page 1826)
`shouldDrawColor` (page 1781) (NSView)

cascadeTopLeftFromPoint

Positions the receiver's top left at *topLeftPoint*, unless *topLeftPoint* is NSPoint.ZeroPoint in which case the receiver is not moved except as needed to constrain to the visible screen.

```
public NSPoint cascadeTopLeftFromPoint(NSPoint topLeftPoint)
```

Discussion

Returns a point shifted from top left of the receiver that can be passed to a subsequent invocation of cascadeTopLeftFromPoint to position the next NSWindow so the title bars of both NSWindows are fully visible.

Both *topLeftPoint* and the return value are expressed in screen coordinates.

See Also

setFrameTopLeftPoint (page 1861)

center

Sets the receiver's location to the center of the screen.

```
public void center()
```

Discussion

The receiver is placed exactly in the center horizontally and somewhat above center vertically. Such a placement carries a certain visual immediacy and importance. This method doesn't put the receiver onscreen, however; use makeKeyAndOrderFront (page 1841) to do that.

You typically use this method to place an NSWindow—most likely an alert dialog—where the user can't miss it. This method is invoked automatically when an NSPanel is placed on the screen by NSApplication's runModalForWindow (page 119) method.

childWindows

Returns an array of the receiver's attached child windows.

```
public NSArray childWindows()
```

Availability

Available in Mac OS X v10.2 and later.

See Also

removeChildWindow (page 1849)

addChildWindow (page 1816)

parentWindow (page 1846)

setParentWindow (page 1866)

close

Removes the receiver from the screen.

```
public void close()
```

Discussion

A window doesn't have to be visible to receive the close message. For example, when the application terminates, it sends the close message to all windows in its window list, even those that are not currently visible.

The close method posts a `WindowWillCloseNotification` (page 1885) to the default notification center.

The close method differs in two important ways from the `performClose` (page 1846) method:

- It does not attempt to send a `windowShouldClose` (page 1880) message to the receiver or its delegate.
- It does not simulate the user clicking the close button by momentarily highlighting the button.

Use `performClose` (page 1846) if you need these features.

See Also

`orderOut` (page 1845)

concludeDragOperation

```
public void concludeDragOperation(NSDraggingInfo draggingInfo)
```

Discussion

Invoked when the dragging operation is complete and the previous `performDragOperation` (page 1846) returned `true`. *draggingInfo* contains details about the dragging operation. This method allows you to perform any tidying up that is needed, such as updating the visual representation now the dragged data has been incorporated. This is the last message sent during a dragging session.

constrainFrameRectToScreen

Modifies and returns *frameRect* so that its top edge lies on *aScreen*.

```
public NSRect constrainFrameRectToScreen(NSRect frameRect, NSScreen aScreen)
```

Discussion

If the receiver is resizable and the receiver's height is greater than the screen height, the rectangle's height is adjusted to fit within the screen as well. The rectangle's width and horizontal location are unaffected. You shouldn't need to invoke this method yourself; it's invoked automatically (and the modified frame is used to locate and set the size of the receiver) whenever a titled NSWindow is placed onscreen and whenever its size is changed.

Subclasses can override this method to prevent their instances from being constrained or to constrain them differently.

contentAspectRatio

Returns the aspect ratio (height in relation to width) of the receiver's content view.

```
public NSSize contentAspectRatio()
```

Availability

Available in Mac OS X v10.3 and later.

See Also
`setContentAspectRatio` (page 1856)

contentMaxSize

Returns the maximum size of the receiver's content view.

```
public NSSize contentMaxSize()
```

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

See Also
`setContentMaxSize` (page 1856)
`contentMinSize` (page 1823)

contentMinSize

Returns the minimum size of the receiver's content view.

```
public NSSize contentMinSize()
```

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

See Also
`setContentMinSize` (page 1857)
`contentMaxSize` (page 1823)

contentRectForFrameRect

Returns the rectangle bounding the receiver's content view given the frame rectangle *frameRect*.

```
public NSRect contentRectForFrameRect(NSRect frameRect)
```

Discussion
Both *frameRect* and the returned content rectangle are expressed in screen coordinates. The receiver uses its current style mask in computing the content rectangle. See the constants sectionfor a list of style mask values. The main advantage of this instance-method counterpart to `contentRectForFrameRect` (page 1814) is that it allows you to take toolbars into account when converting between content and frame rectangles. (The toolbar is not included in the content rectangle.)

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

See Also
`frameRectForContentRect` (page 1835)
`"contentRectForFrameRect"` (page 1814)

contentResizeIncrements

Returns the size of increments used during resizing of the receiver's content rectangle.

```
public NSSize contentResizeIncrements()
```

Discussion

Resizing of the content rectangle is constrained to integral multiples of the returned height and width.

Availability

Available in Mac OS X v10.3 and later.

See Also

`setContentResizeIncrements` (page 1857)

contentView

Returns the receivers's content view, the highest accessible NSView object in the receiver's view hierarchy.

```
public NSView contentView()
```

See Also

`setContentView` (page 1858)

convertBaseToScreen

Returns *aPoint* converted from the receiver's base coordinate system to the screen coordinate system.

```
public NSPoint convertBaseToScreen(NSPoint aPoint)
```

See Also

`convertScreenToBase` (page 1824)

`convertPointToView` (page 1745) (NSView)

convertScreenToBase

Returns *aPoint* converted from the screen coordinate system to the receiver's base coordinate system.

```
public NSPoint convertScreenToBase(NSPoint aPoint)
```

See Also

`convertBaseToScreen` (page 1824)

`convertRectFromView` (page 1745) (NSView)

currentEvent

Returns the event currently being processed by the application, by invoking NSApplication's `currentEvent` (page 110) method.

```
public NSEvent currentEvent()
```

dataWithEPSInsideRect

Returns EPS data that draws the region of the receiver within *aRect* (expressed in the receiver's base coordinate system).

```
public NSData dataWithEPSInsideRect(NSRect aRect)
```

Discussion

This data can be placed on a pasteboard, written to a file, or used to create an NSImage object.

See Also

dataWithEPSInsideRect (page 1746) (NSView)

writeEPSInsideRectToPasteboard (page 1788) (NSView)

dataWithPDFInsideRect

Returns PDF data that draws the region of the receiver within *aRect* (expressed in the receiver's base coordinate system).

```
public NSData dataWithPDFInsideRect(NSRect aRect)
```

Discussion

This data can be placed on a pasteboard, written to a file, or used to create an NSImage object.

See Also

dataWithPDFInsideRect (page 1747) (NSView)

writePDFInsideRectToPasteboard (page 1788) (NSView)

deepestScreen

Returns the deepest screen the receiver is on (it may be split over several screens), or `null` if the receiver is offscreen.

```
public NSScreen deepestScreen()
```

See Also

screen (page 1852)

defaultButtonCell

Returns the button cell that performs as if clicked when the NSWindow receives a Return (or Enter) key event.

```
public NSButtonCell defaultButtonCell()
```

Discussion

This cell draws itself as the focal element for keyboard interface control, unless another button cell is focused on, in which case the default button cell temporarily draws itself as normal and disables its key equivalent.

The window receives a Return key event if no responder in its responder chain claims it, or if the user presses the Control key along with the Return key.

See Also

`setDefaultButtonCell` (page 1858)
`disableKeyEquivalentForDefaultButtonCell` (page 1828)
`enableKeyEquivalentForDefaultButtonCell` (page 1832)

delegate

Returns the receiver's delegate, or `null` if it doesn't have a delegate.

```
public Object delegate()
```

See Also

`setDelegate` (page 1858)

deminiaturize

This action method deminiaturizes the receiver.

```
public void deminiaturize(Object sender)
```

Discussion

Invoke this method to programmatically deminiaturize a miniaturized window in the dock.

See Also

`miniaturize` (page 1842)
`styleMask` (page 1870)

depthLimit

Returns the depth limit of the receiver.

```
public int depthLimit()
```

See Also

`"defaultDepthLimit"` (page 1814)
`setDepthLimit` (page 1858)
`setDynamicDepthLimit` (page 1859)

deviceDescription

Returns a dictionary containing information about the receiver's resolution, color depth, and so on.

```
public NSDictionary deviceDescription()
```

Discussion

This information is useful for tuning images and colors to the window's display capabilities. The contents of the dictionary are:

Dictionary Key	Value
`NSGraphics.DeviceResolution`	An NSSize that describes the receiver's raster resolution in dots per inch (dpi).
`NSGraphics.DeviceColorSpaceName`	A String giving the name of the receiver's color space. See the constants section in NSGraphics for a list of possible values.
`NSGraphics.DeviceBitsPerSample`	A Number containing an integer that gives the bit depth of the receiver's raster image (2-bit, 8-bit, and so forth).
`NSGraphics.DeviceIsScreen`	A result of `true`, indicating that the receiver displays on the screen.
`NSGraphics.DeviceSize`	An NSSize that gives the size of the receiver's frame rectangle.

See Also

`deviceDescription` (page 1249) (NSScreen)

`bestRepresentationForDevice` (page 755) (NSImage)

`colorUsingColorSpaceName` (page 372) (NSColor)

disableCursorRects

Disables all cursor rectangle management within the receiver.

```
public void disableCursorRects()
```

Discussion

Use this method when you need to do some special cursor manipulation and you don't want the Application Kit interfering.

See Also

`enableCursorRects` (page 1831)

disableFlushWindow

Disables the `flushWindow` (page 1833) method for the receiver.

```
public void disableFlushWindow()
```

Discussion

If the receiver is buffered, disabling `flushWindow` (page 1833) prevents drawing from being automatically flushed by NSView's `display...` methods from the receiver's backing store to the screen. This method permits several NSViews to be drawn before the results are shown to the user.

Flushing should be disabled only temporarily, while the NSWindow's display is being updated. Each `disableFlushWindow` (page 1827) message must be paired with a subsequent `enableFlushWindow` (page 1832) message. Invocations of these methods can be nested; flushing isn't reenabled until the last (unnested) `enableFlushWindow` (page 1832) message is sent.

disableKeyEquivalentForDefaultButtonCell

Disables the default button cell's key equivalent, so it doesn't perform a click when the user presses Return (or Enter).

```
public void disableKeyEquivalentForDefaultButtonCell()
```

Discussion

See the method description for `defaultButtonCell` (page 1825) for more information.

See Also

`enableKeyEquivalentForDefaultButtonCell` (page 1832)

disableScreenUpdatesUntilFlush

Disables the receiver's screen updates until the window is flushed.

```
public void disableScreenUpdatesUntilFlush()
```

Discussion

This method can be invoked to synchronize hardware surface flushes with the window's flushes. The receiver immediately disables screen updates using the function `NSDisableScreenUpdates` and re-enables screen updates when the window flushes. Sending this message multiple times during a window update cycle has no effect.

Availability

Available in Mac OS X v10.4 and later.

discardCachedImage

Discards all of the receiver's cached image rectangles.

```
public void discardCachedImage()
```

Discussion

An NSWindow automatically discards its cached image rectangles when it displays.

See Also

`cacheImageInRect` (page 1819)
`restoreCachedImage` (page 1851)
`display` (page 1829)

discardCursorRects

Invalidates all cursor rectangles in the receiver.

```
public void discardCursorRects()
```

Discussion

This method is invoked by `resetCursorRects` (page 1849) to clear out existing cursor rectangles before resetting them. You shouldn't invoke it in the code you write, but might want to override it to change its behavior.

discardEventsMatchingMask

Forwards the message to the NSApplication object, which handles it as described in the NSApplication class specification.

```
public void discardEventsMatchingMask(int mask, NSEvent lastEvent)
```

display

Passes a display message down the receiver's view hierarchy, thus redrawing all NSViews within the receiver, including the frame view that draws the border, title bar, and other peripheral elements.

```
public void display()
```

Discussion
You rarely need to invoke this method. NSWindows normally record which of their NSViews need display and display them automatically on each pass through the event loop.

See Also
`display` (page 1747) (NSView)
`displayIfNeeded` (page 1829)
`isAutodisplay` (page 1837)

displayIfNeeded

Passes a `displayIfNeeded` (page 1829) message down the receiver's view hierarchy, thus redrawing all NSViews that need to be displayed, including the frame view that draws the border, title bar, and other peripheral elements.

```
public void displayIfNeeded()
```

Discussion
This method is useful when you want to modify some number of NSViews and then display only the ones that were modified.

You rarely need to invoke this method. NSWindows normally record which of their NSViews need display and display them automatically on each pass through the event loop.

See Also
`display` (page 1829)
`displayIfNeeded` (page 1748) (NSView)
`setNeedsDisplay` (page 1779) (NSView)
`isAutodisplay` (page 1837)

displaysWhenScreenProfileChanges

Returns `true` if the window context should be updated when the screen profile changes or when the window moves to a different screen.

```
public boolean displaysWhenScreenProfileChanges()
```

Discussion

Returns `false` (the default value) if the window context should stay the same despite profile changes.

Availability

Available in Mac OS X v10.4 and later.

See Also

`setDisplaysWhenScreenProfileChanges` (page 1859)

draggingEntered

```
public int draggingEntered(NSDraggingInfo draggingInfo)
```

Discussion

Invoked when a dragged image enters the receiver. Specifically, this method is invoked when the mouse pointer enters the receiver's frame rectangle. *draggingInfo* contains details about the dragging operation.

Returns a value indicating which dragging operation will be performed when the image is released. In deciding which dragging operation to return, you should evaluate the overlap between both the dragging operations allowed by the source and the dragging operations and pasteboard data types the receiver supports. The returned value should be one of the following:

Option	Meaning
`NSDraggingInfo.DragOperationCopy`	The data represented by the image will be copied.
`NSDraggingInfo.DragOperationLink`	The data will be shared.
`NSDraggingInfo.DragOperationGeneric`	The operation will be defined by the destination.
`NSDraggingInfo.DragOperationPrivate`	The operation is negotiated privately between the source and the destination.
`NSDraggingInfo.DragOperationEvery`	Combines all the above.

If none of the operations is appropriate, returns `NSDraggingInfo.DragOperationNone`.

See Also

`draggingUpdated` (page 1831)

`draggingExited` (page 1830)

draggingExited

Invoked when the dragged image exits the receiver's frame rectangle.

```
public void draggingExited(NSDraggingInfo draggingInfo)
```

Discussion

draggingInfo contains details about the dragging operation.

draggingUpdated

```
public int draggingUpdated(NSDraggingInfo draggingInfo)
```

Discussion
Invoked periodically as the dragged image is held within the receiver. The messages continue until the image is either released or dragged out of the receiver. *draggingInfo* contains details about the dragging operation. Returns one of the dragging operation options listed under draggingEntered (page 1830).

This method provides you with an opportunity to modify the dragging operation depending on the position of the mouse pointer inside of the receiver. For example, you may have several graphics or areas of text contained within the same view and wish to tailor the dragging operation or to ignore the drag event completely, depending upon which object is underneath the mouse pointer at the time when the user releases the dragged image and performDragOperation (page 1846) is invoked.

You typically examine the contents of the pasteboard in draggingEntered (page 1830), as this method is invoked only once, rather than in this method, which is invoked multiple times.

Only one window at a time receives a sequence of draggingUpdated messages. If the mouse pointer is within the bounds of two overlapping windows that are both valid destinations, the uppermost window receives these messages until the image is either released or dragged out.

See Also
draggingExited (page 1830)
prepareForDragOperation (page 1847)

dragImage

Begins a dragging session.

```
public void dragImage(NSImage anImage, NSPoint aPoint, NSSize initialOffset, NSEvent
     theEvent, NSPasteboard pboard, Object sourceObject, boolean slideBack)
```

Discussion
This method should be invoked only from within a view's implementation of the mouseDown (page 1192) or mouseDragged (page 1192) methods (which overrides the version defined in NSResponder). Essentially the same as NSView's method of the same name, except that *aPoint* is given in the NSWindow's base coordinate system. See the description of NSView's dragImage (page 1751) for more information.

drawers

Returns the collection of drawers associated with the receiver.

```
public NSArray drawers()
```

enableCursorRects

Reenables cursor rectangle management within the receiver after a disableCursorRects (page 1827) message.

```
public void enableCursorRects()
```

enableFlushWindow

Reenables the `flushWindow` (page 1833) method for the receiver after it was disabled through a previous `disableFlushWindow` (page 1827) message.

```
public void enableFlushWindow()
```

enableKeyEquivalentForDefaultButtonCell

Reenables the default button cell's key equivalent, so it performs a click when the user presses Return (or Enter).

```
public void enableKeyEquivalentForDefaultButtonCell()
```

Discussion

See the method description for `defaultButtonCell` (page 1825) for more information.

See Also

`disableKeyEquivalentForDefaultButtonCell` (page 1828)

endEditingForObject

Forces the field editor, which *anObject* is assumed to be using, to give up its first responder status and prepares it for its next assignment.

```
public void endEditingForObject(Object anObject)
```

Discussion

If the field editor is the first responder, it's made to resign that status even if its `resignFirstResponder` (page 1196) method returns `false`. This registration forces the field editor to send a `textDidEndEditing` (page 1532) message to its delegate. The field editor is then removed from the view hierarchy, its delegate is set to `null`, and it's emptied of any text it may contain.

This method is typically invoked by the object using the field editor when it's finished. Other objects normally change the first responder by simply using `makeFirstResponder` (page 1841), which allows a field editor or other object to retain its first responder status if, for example, the user has entered an invalid value. The `endEditingForObject` (page 1832) method should be used only as a last resort if the field editor refuses to resign first responder status. Even in this case, you should always allow the field editor a chance to validate its text and take whatever other action it needs first. You can do this by first trying to make the NSWindow the first responder.

See Also

`fieldEditorForObject` (page 1832)

`windowWillReturnFieldEditor` (page 1882)

fieldEditorForObject

Returns the receiver's field editor, creating it if needed if *createFlag* is `true`.

```
public NSText fieldEditorForObject(boolean createFlag, Object anObject)
```

Discussion

Returns `null` if *`createFlag`* is `false` and the field editor doesn't exist. *`anObject`* is used to allow the receiver's delegate to substitute another object in place of the field editor, as described below. The field editor may be in use by some view object, so be sure to properly dissociate it from that object before actually using it yourself (the appropriate way to do this is illustrated in the description of `endEditingForObject` (page 1832)). Once you retrieve the field editor, you can insert it in the view hierarchy, set a delegate to interpret text events, and have it perform whatever editing is needed. Then, when it sends a `textDidEndEditing` (page 1532) message to the delegate, you can get its text to display or store and remove the field editor using `endEditingForObject` (page 1832).

The field editor is a single NSTextView object that is shared among all the controls in a window for light text-editing needs. It is automatically instantiated when needed, and it can be used however your application sees fit. Typically, the field editor is used by simple text-bearing objects—for example, an NSTextField object uses its window'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 layout (for this you should create a dedicated NSTextView object).

A freshly created NSWindow doesn't have a field editor. After a field editor has been created for an NSWindow, the *`createFlag`* argument is ignored. By passing `false` for *`createFlag`* and testing the return value, however, you can predicate an action on the existence of the field editor.

The receiver's delegate can substitute a custom editor in place of the NSWindow's field editor by implementing `windowWillReturnFieldEditor` (page 1882). The receiver sends this message to its delegate with itself and *`anObject`* as the arguments, and if the return value is not `null` the NSWindow returns that object instead of its field editor. However, note the following:

- If the NSWindow's delegate is identical to *`anObject`*, `windowWillReturnFieldEditor` (page 1882) isn't sent.
- The object returned by the delegate method, though it may become first responder, does not become the NSWindow's field editor. Other objects continue to use the NSWindow's established field editor.

firstResponder

Returns the receiver's first responder.

```
public NSResponder firstResponder()
```

See Also

`makeFirstResponder` (page 1841)
`acceptsFirstResponder` (page 1189) (NSResponder)

flushWindow

Flushes the receiver's offscreen buffer to the screen if the receiver is buffered and flushing is enabled.

```
public void flushWindow()
```

Discussion

Does nothing for other display devices, such as a printer. This method is automatically invoked by NSWindow's and NSView's `display` (page 1829) and `displayIfNeeded` (page 1829) methods.

See Also

flushWindowIfNeeded (page 1834)

disableFlushWindow (page 1827)

enableFlushWindow (page 1832)

flushWindowIfNeeded

Flushes the receiver's offscreen buffer to the screen if flushing is enabled and if the last flushWindow (page 1833) message had no effect because flushing was disabled.

```
public void flushWindowIfNeeded()
```

Discussion

To avoid unnecessary flushing, use this method rather than flushWindow (page 1833) to flush an NSWindow after flushing has been reenabled.

See Also

flushWindow (page 1833)

disableFlushWindow (page 1827)

enableFlushWindow (page 1832)

frame

Returns the receiver's frame rectangle.

```
public NSRect frame()
```

Discussion

The frame rectangle is always reckoned in the screen coordinate system.

See Also

screen (page 1852)

deepestScreen (page 1825)

frameAutosaveName

Returns the name used to automatically save the receiver's frame rectangle data in the defaults system, as set through setFrameAutosaveName (page 1860).

```
public String frameAutosaveName()
```

Discussion

If the receiver has an autosave name, its frame data is written whenever the frame rectangle changes.

See Also

setFrameUsingName (page 1862)

frameRectForContentRect

Returns the receiver's frame rectangle given the rectangle bounding the content view *contentRect*.

```
public NSRect frameRectForContentRect(NSRect contentRect)
```

Discussion

Both *contentRect* and the returned frame rectangle are expressed in screen coordinates. The receiver uses its current style mask in computing the frame rectangle. See the constants section for a list of style mask values. The major advantage of this instance-method counterpart to frameRectForContentRect (page 1814) is that it allows you to take toolbars into account when converting between content and frame rectangles. (The toolbar is included in the frame rectangle but not the content rectangle.)

Availability

Available in Mac OS X v10.3 and later.

See Also

contentRectForFrameRect (page 1823)

"frameRectForContentRect" (page 1814)

graphicsContext

Returns the graphics context associated with the receiver for the current thread.

```
public NSGraphicsContext graphicsContext()
```

Availability

Available in Mac OS X v10.4 and later.

gState

Returns the graphics state object associated with the receiver.

```
public int gState()
```

Discussion

This graphics state is used by default for all NSViews in the receiver's view hierarchy, but individual NSViews can be made to use their own with the NSView method allocateGState (page 1741).

hasDynamicDepthLimit

Returns `true` if the receiver's depth limit can change to match the depth of the screen it's on, `false` if it can't.

```
public boolean hasDynamicDepthLimit()
```

See Also

setDynamicDepthLimit (page 1859)

hasShadow

Returns `true` if the window has a shadow; otherwise returns `false`.

```
public boolean hasShadow()
```

See Also

`setHasShadow` (page 1862)
`invalidateShadow` (page 1837)

hidesOnDeactivate

Returns `true` if the receiver is removed from the screen when its application is deactivated, `false` if it remains onscreen.

```
public boolean hidesOnDeactivate()
```

See Also

`setHidesOnDeactivate` (page 1862)

ignoresMouseEvents

Return whether the receiver is transparent to mouse events.

```
public boolean ignoresMouseEvents()
```

Availability

Available in Mac OS X v10.2 and later.

See Also

`setIgnoresMouseEvents` (page 1863)

initialFirstResponder

Returns the NSView that's made first responder the first time the receiver is placed onscreen.

```
public NSView initialFirstResponder()
```

See Also

`setInitialFirstResponder` (page 1863)
`setNextKeyView` (page 1779) (NSView)

invalidateCursorRectsForView

Marks as invalid the cursor rectangles of *aView*, an NSView in the receiver's view hierarchy, so they'll be set up again when the receiver becomes key (or immediately if the receiver is key).

```
public void invalidateCursorRectsForView(NSView aView)
```

See Also

resetCursorRects (page 1849)

resetCursorRects (page 1770) (NSView)

invalidateShadow

Invalidates the window shadow so that it is recomputed based on the current window shape.

```
public void invalidateShadow()
```

Availability

Available in Mac OS X v10.2 and later.

See Also

hasShadow (page 1836)

setHasShadow (page 1862)

isAutodisplay

Returns true if the receiver automatically displays its views that are marked as needing it, false if it doesn't.

```
public boolean isAutodisplay()
```

Discussion

Automatic display typically occurs on each pass through the event loop.

See Also

setAutodisplay (page 1855)

displayIfNeeded (page 1829)

setNeedsDisplay (page 1779) (NSView)

isDocumentEdited

Returns true or false according to the argument supplied with the last setDocumentEdited (page 1859) message.

```
public boolean isDocumentEdited()
```

isExcludedFromWindowsMenu

Returns true if the receiver's title is omitted from the application's Windows menu, false if it is listed.

```
public boolean isExcludedFromWindowsMenu()
```

See Also

setExcludedFromWindowsMenu (page 1860)

isFlushWindowDisabled

Returns `true` if the receiver's flushing ability has been disabled; otherwise returns `false`.

```
public boolean isFlushWindowDisabled()
```

See Also
`disableFlushWindow` (page 1827)
`enableFlushWindow` (page 1832)

isKeyWindow

Returns `true` if the receiver is the key window for the application, `false` if it isn't.

```
public boolean isKeyWindow()
```

See Also
`isMainWindow` (page 1838)
`makeKeyWindow` (page 1842)

isMainWindow

Returns `true` if the receiver is the main window for the application, `false` if it isn't.

```
public boolean isMainWindow()
```

See Also
`isKeyWindow` (page 1838)
`makeMainWindow` (page 1842)

isMiniaturized

Returns `true` if the receiver has been miniaturized, `false` if it hasn't.

```
public boolean isMiniaturized()
```

Discussion
A miniaturized window is removed from the screen and replaced by a image, icon, or button that represents it, called the counterpart.

See Also
`miniaturize` (page 1842)

isMovableByWindowBackground

Returns `true` if the receiver is movable by clicking and dragging anywhere in its background, `false` if not.

```
public boolean isMovableByWindowBackground()
```

Discussion

A window with a style mask of `NSTexturedBackgroundWindowMask` returns `true` by default. Sheets and drawers cannot be movable by window background.

Availability

Available in Mac OS X v10.2 and later.

See Also

`setMovableByWindowBackground` (page 1865)

isOneShot

Returns `true` if the window device the receiver manages is freed when it's removed from the screen list, `false` if not.

```
public boolean isOneShot()
```

Discussion

The default is `false`.

See Also

`setOneShot` (page 1865)

isOpaque

Returns whether the receiver is opaque.

```
public boolean isOpaque()
```

Discussion

The default is `true`.

See Also

`setOpaque` (page 1866)

isReleasedWhenClosed

Returns `true` if the receiver is automatically released after being closed, `false` if it's simply removed from the screen.

```
public boolean isReleasedWhenClosed()
```

Discussion

The default for NSWindow is `true`; the default for NSPanel is `false`. Release when closed, however, is ignored for windows owned by window controllers.

See Also

`setReleasedWhenClosed` (page 1867)

isSheet

Returns `true` if the receiver has ever run as a modal sheet.

```
public boolean isSheet()
```

Discussion
Sheets are created using the NSPanel subclass.

isVisible

Returns `true` if the receiver is onscreen (even if it's obscured by other windows).

```
public boolean isVisible()
```

See Also
`visibleRect` (page 1786) (NSView)

isZoomed

Returns whether the receiver is in a zoomed state.

```
public boolean isZoomed()
```

See Also
`zoom` (page 1874)

keyDown

Handles *theEvent* keyboard event that may need to be interpreted as changing the key view or triggering a keyboard equivalent.

```
public void keyDown(NSEvent theEvent)
```

See Also
`selectNextKeyView` (page 1852)
`nextKeyView` (page 1761) (NSView)
`performMnemonic` (page 1763) (NSView)

keyViewSelectionDirection

Returns the direction the receiver is currently using to change the key view.

```
public int keyViewSelectionDirection()
```

Discussion
This direction can be one of the values described in the constants section

See Also
`selectNextKeyView` (page 1852)
`selectPreviousKeyView` (page 1853)

level

Returns the level of the receiver as set using `setLevel` (page 1863).

```
public int level()
```

makeFirstResponder

Attempts to make *aResponder* the first responder for the receiver.

```
public boolean makeFirstResponder(NSResponder aResponder)
```

Discussion

If *aResponder* isn't already the first responder, this method first sends a `resignFirstResponder` (page 1196) message to the object that is. If that object refuses to resign, it remains the first responder, and this method immediately returns `false`. If it returns `true`, this method sends a `becomeFirstResponder` (page 1190) message to *aResponder*. If *aResponder* accepts first responder status, this method returns `true`. If it refuses, this method returns `false`, and the NSWindow becomes first responder.

If *aResponder* is `null`, this method still sends `resignFirstResponder` (page 1196) to the current first responder. If the current first responder refuses to resign, it remains the first responder and this method immediately returns `false`. If the current first responder returns `true` from `resignFirstResponder`, the receiver is made its own first responder and this method returns `true`.

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* is typically an NSView in the receiver's view hierarchy. If this method is called explicitly, first send `acceptsFirstResponder` (page 1189) to *aResponder*, and do not call `makeFirstResponder` if `acceptsFirstResponder` returns `false`.

Use `setInitialFirstResponder` (page 1863) to the set the first responder to be used when the window is brought onscreen for the first time.

See Also

`becomeFirstResponder` (page 1190) (NSResponder)
`resignFirstResponder` (page 1196) (NSResponder)

makeKeyAndOrderFront

This action method moves the receiver to the front of the screen list, within its level, and makes it the key window.

```
public void makeKeyAndOrderFront(Object sender)
```

See Also

`orderFront` (page 1844)
`orderBack` (page 1844)
`orderOut` (page 1845)
`orderWindow` (page 1845)
`setLevel` (page 1863)

makeKeyWindow

Makes the receiver the key window.

```
public void makeKeyWindow()
```

See Also

makeMainWindow (page 1842)

becomeKeyWindow (page 1819)

isKeyWindow (page 1838)

makeMainWindow

```
public void makeMainWindow()
```

Discussion

Makes the receiver the main window.

See Also

makeKeyWindow (page 1842)

becomeMainWindow (page 1819)

isMainWindow (page 1838)

maxSize

Returns the maximum size to which the receiver's frame (including its title bar) can be sized either by the user or by the setFrame... methods other than setFrame (page 1860).

```
public NSSize maxSize()
```

See Also

setMaxSize (page 1864)

minSize (page 1843)

aspectRatio (page 1817)

resizeIncrements (page 1851)

miniaturize

This action method removes the receiver from the screen list and displays the miniaturized window in the dock.

```
public void miniaturize(Object sender)
```

See Also

deminiaturize (page 1826)

miniwindowImage

Returns the custom miniaturized window image of the receiver.

```
public NSImage miniwindowImage()
```

Discussion

The miniaturized window image is the image displayed in the dock when the window is minimized. If you did not assign a custom image to the window, this method returns `null`.

See Also

`setMiniwindowImage` (page 1864)

`miniwindowTitle` (page 1843)

miniwindowTitle

Returns the title displayed in the receiver’s miniaturized window.

```
public String miniwindowTitle()
```

See Also

`setMiniwindowTitle` (page 1864)

`miniwindowImage` (page 1842)

minSize

Returns the minimum size to which the receiver’s frame (including its title bar) can be sized either by the user or by the `setFrame...` methods other than `setFrame` (page 1860).

```
public NSSize minSize()
```

See Also

`setMinSize` (page 1865)

`maxSize` (page 1842)

`aspectRatio` (page 1817)

`resizeIncrements` (page 1851)

mouseLocationOutsideOfEventStream

Returns the current location of the mouse reckoned in the receiver’s base coordinate system, regardless of the current event being handled or of any events pending.

```
public NSPoint mouseLocationOutsideOfEventStream()
```

Discussion

For the same information in screen coordinates, use NSEvent's `"mouseLocation"` (page 609).

See Also

`currentEvent` (page 110) (NSApplication)

nextEventMatchingMask

Invokes NSApplication's `nextEventMatchingMask` (page 115) method, using *`mask`* as the first argument, with an unlimited expiration, a mode of `NSApplication.EventTrackingRunLoopMode`, and a dequeue flag of `true`.

```
public NSEvent nextEventMatchingMask(int mask)
```

Discussion

See the method description in the NSApplication class specification for more information.

Forwards the message to the global NSApplication object, `NSApplication.sharedApplication()`.

```
public NSEvent nextEventMatchingMask(int mask, NSDate expirationDate, String mode,
     boolean dequeue)
```

Discussion

See the method description in the NSApplication class specification for more information.

orderBack

This action method moves the receiver to the back of its level in the screen list, without changing either the key window or the main window.

```
public void orderBack(Object sender)
```

See Also

`orderFront` (page 1844)
`orderOut` (page 1845)
`orderWindow` (page 1845)
`makeKeyAndOrderFront` (page 1841)
`level` (page 1841)

orderFront

This action method moves the receiver to the front of its level in the screen list, without changing either the key window or the main window.

```
public void orderFront(Object sender)
```

See Also

`orderBack` (page 1844)
`orderOut` (page 1845)
`orderWindow` (page 1845)
`makeKeyAndOrderFront` (page 1841)
`level` (page 1841)

orderFrontRegardless

Moves the receiver to the front of its level, even if its application isn't active, but without changing either the key window or the main window.

```
public void orderFrontRegardless()
```

Discussion

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 in such a way that an active application (with the key window) is using another application to display data.

See Also

`orderFront` (page 1844)

`level` (page 1841)

orderOut

This action method takes the receiver out of the screen list.

```
public void orderOut(Object sender)
```

Discussion

If the receiver is the key or main window, the NSWindow immediately behind it is made key or main in its place. Calling the `orderOut` (page 1845) method causes the receiver to be removed from the screen, but does not cause it to be released.

See Also

`orderFront` (page 1844)

`orderBack` (page 1844)

`orderWindow` (page 1845)

`setReleasedWhenClosed` (page 1867)

orderWindow

Repositions the receiver's window device in the window server's screen list.

```
public void orderWindow(int place, int otherWindowNumber)
```

Discussion

If *`place`* is `Out`, the receiver is removed from the screen list and *`otherWindowNumber`* is ignored. If it's `Above`, the receiver is ordered immediately in front of the window whose window number is *`otherWindowNumber`*. Similarly, if place is `Below`, the receiver is placed immediately behind the window represented by *`otherWindowNumber`*. If *`otherWindowNumber`* is 0, the receiver is placed in front of or behind all other windows in its level.

See Also

`orderFront` (page 1844)

`orderBack` (page 1844)

`orderOut` (page 1845)

`makeKeyAndOrderFront` (page 1841)

level (page 1841)
windowNumber (page 1873)

parentWindow

Returns the parent window to which the receiver is attached as a child.

```
public NSWindow parentWindow()
```

Availability
Available in Mac OS X v10.2 and later.

See Also
removeChildWindow (page 1849)
childWindows (page 1821)
addChildWindow (page 1816)
setParentWindow (page 1866)

performClose

This action method simulates the user clicking the close button by momentarily highlighting the button and then closing the window.

```
public void performClose(Object sender)
```

Discussion
If the receiver's delegate or the receiver itself implements windowShouldClose (page 1880), then that message is sent with the receiver as the argument. (Only one such message is sent; if both the delegate and the NSWindow implement the method, only the delegate receives the message.) If the windowShouldClose (page 1880) method returns false, the window isn't closed. If it returns true, or if it isn't implemented, performClose (page 1846) invokes the close (page 1821) method to close the window.

If the receiver doesn't have a close button or can't be closed (for example, if the delegate replies false to a windowShouldClose (page 1880) message), the system beeps.

See Also
styleMask (page 1870)
performMiniaturize (page 1847)

performDragOperation

```
public boolean performDragOperation(NSDraggingInfo aDraggingInfo)
```

Discussion
Invoked after the released image has been removed from the screen and the previous prepareForDragOperation (page 1847) message has returned true. *aDraggingInfo* contains details about the dragging operation. This method should do the real work of importing the pasteboard data represented by the image. If the receiver accepts the data, return true, otherwise return false.

See Also
`concludeDragOperation` (page 1822)

performMiniaturize

This action method simulates the user clicking the miniaturize button by momentarily highlighting the button, then miniaturizing the window.

```
public void performMiniaturize(Object sender)
```

Discussion
If the receiver doesn't have a miniaturize button or can't be miniaturized for some reason, the system beeps.

See Also
`close` (page 1821)
`styleMask` (page 1870)
`performClose` (page 1846)

performZoom

This action method simulates the user clicking the zoom box by momentarily highlighting the button and then zooming the window.

```
public void performZoom(Object sender)
```

Discussion
If the receiver doesn't have a zoom box or can't be zoomed for some reason, the system beeps.

See Also
`styleMask` (page 1870)
`zoom` (page 1874)

postEvent

Forwards the message to the global NSApplication object, `NSApplication.sharedApplication()`.

```
public void postEvent(NSEvent anEvent, boolean flag)
```

Discussion
See "NSApplication" (page 93) for details.

prepareForDragOperation

```
public boolean prepareForDragOperation(NSDraggingInfo aDraggingInfo)
```

Discussion
Invoked when the image is released, if the most recent `draggingEntered` (page 1830) or `draggingUpdated` (page 1831) message returned an acceptable drag-operation value. *`aDraggingInfo`* contains details about the dragging operation. Returns `true` if the receiver agrees to perform the drag operation and `false` if not.

See Also

`performDragOperation` (page 1846)

preservesContentDuringLiveResize

Returns `true` if the window tries to optimize live resize operations by preserving the content of views that have not moved; otherwise, returns `false`.

```
public boolean preservesContentDuringLiveResize()
```

Discussion

When enabled, the window redraws only those views that moved (or do not support this optimization) during a live resize operation.

See `preservesContentDuringLiveResize` (page 1848) in NSView for additional information on how to support this optimization.

Availability

Available in Mac OS X v10.4 and later.

See Also

`setPreservesContentDuringLiveResize` (page 1866)
`preservesContentDuringLiveResize` (page 1764) (NSView)

print

This action method runs the Print panel, and if the user chooses an option other than canceling, prints the receiver (its frame view and all subviews).

```
public void print(Object sender)
```

recalculateKeyViewLoop

Marks the key view loop as dirty and in need of recalculation.

```
public void recalculateKeyViewLoop()
```

Discussion

The key view loop is actually recalculated the next time someone requests the next or previous key view of the window. The recalculated loop is based on the geometric order of the views in the window.

If you do not want to maintain the key view loop of your window manually, you can use this method to do it for you. When it is first loaded, NSWindow calls this method automatically if your window does not have a key view loop already established. If you add or remove views later, you can call this method manually from your code to update the window's key view loop. You can also call `setAutorecalculatesKeyViewLoop` (page 1855) to have the window recalculate the loop automatically.

Availability

Available in Mac OS X v10.4 and later.

See Also

selectKeyViewFollowingView (page 1852)

selectKeyViewPrecedingView (page 1852)

setAutorecalculatesKeyViewLoop (page 1855)

registerForDraggedTypes

Registers *pboardTypes* as the pasteboard types the receiver will accept as the destination of an image-dragging session.

```
public void registerForDraggedTypes(NSArray pboardTypes)
```

Discussion

Registering an NSWindow for dragged types automatically makes it a candidate destination object for a dragging session. As such, it must properly implement some or all of the NSDraggingDestination interface methods. As a convenience, NSWindow provides default implementations of these methods. See the "NSDraggingDestination" (page 1955) interface specification for details.

See Also

unregisterDraggedTypes (page 1871)

removeChildWindow

Detaches *childWin* from the receiver.

```
public void removeChildWindow(NSWindow childWin)
```

Availability

Available in Mac OS X v10.2 and later.

See Also

addChildWindow (page 1816)

childWindows (page 1821)

parentWindow (page 1846)

setParentWindow (page 1866)

representedFilename

Returns the name of the file the receiver represents.

```
public String representedFilename()
```

See Also

setRepresentedFilename (page 1867)

resetCursorRects

Invokes discardCursorRects (page 1828) to clear the receiver's cursor rectangles, then sends resetCursorRects (page 1849) to every NSView in the receiver's view hierarchy.

```
public void resetCursorRects()
```

Discussion

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` (page 1836).

resignKeyWindow

Never invoke this method; it's invoked automatically when the NSWindow resigns key window status.

```
public void resignKeyWindow()
```

Discussion

This method sends `resignKeyWindow` (page 1850) to the receiver's first responder, sends `windowDidResignKey` (page 1879) to the receiver's delegate, and posts a `WindowDidResignKeyNotification` (page 1884) to the default notification center.

See Also

`becomeKeyWindow` (page 1819)

`resignMainWindow` (page 1850)

resignMainWindow

Never invoke this method; it's invoked automatically when the NSWindow resigns main window status.

```
public void resignMainWindow()
```

Discussion

This method sends `windowDidResignMain` (page 1879) to the receiver's delegate and posts a `WindowDidResignMainNotification` (page 1884) to the default notification center.

See Also

`becomeMainWindow` (page 1819)

`resignKeyWindow` (page 1850)

resizeFlags

Valid only while the receiver is being resized, this method returns the flags field of the event record for the mouse-down event that initiated the resizing session.

```
public int resizeFlags()
```

Discussion

The integer encodes, as a mask, which of the modifier keys was held down when the event occurred. The flags are listed in NSEvent's `modifierFlags` (page 616) method description. You can use this method to constrain the direction or amount of resizing. Because of its limited validity, this method should only be invoked from within an implementation of the delegate method `windowWillResize` (page 1881).

resizeIncrements

Returns the receiver's resizing increments, which restrict the user's ability to resize it so that its width and height alter by integral multiples of `increments.width` and `increments.height` when the user resizes it.

```
public NSSize resizeIncrements()
```

Discussion
These amounts are whole number values, 1.0 or greater. You can set an NSWindow's size to any value programmatically.

See Also
`setResizeIncrements` (page 1867)
`setAspectRatio` (page 1854)
`setFrame` (page 1860)

restoreCachedImage

Splices the receiver's cached image rectangles, if any, back into its raster image (and buffer if it has one), undoing the effect of any drawing performed within those areas since they were established using `cacheImageInRect` (page 1819).

```
public void restoreCachedImage()
```

Discussion
You must invoke `flushWindow` (page 1833) after this method to guarantee proper redisplay. An NSWindow automatically discards its cached image rectangles when it displays.

See Also
`discardCachedImage` (page 1828)
`display` (page 1829)

runToolbarCustomizationPalette

The action method for the "Customize Toolbar…" menu item.

```
public void runToolbarCustomizationPalette(Object sender)
```

Discussion
See the NSToolbar (page 1693) class description for additional information.

saveFrameUsingName

Saves the receiver's frame rectangle in the user defaults system.

```
public void saveFrameUsingName(String name)
```

Discussion
With the companion method `setFrameUsingName` (page 1862), you can save and reset an NSWindow's frame over various launchings of an application. The default is owned by the application and stored under the name "NSWindow Frame *name*." See the NSUserDefaults class specification for more information.

See Also
`stringWithSavedFrame` (page 1870)

screen

Returns the screen the receiver is on.

```
public NSScreen screen()
```

Discussion
If the receiver is partly on one screen and partly on another, the screen where most of it lies is returned. Returns null if the receiver is completely offscreen.

See Also
`deepestScreen` (page 1825)

selectKeyViewFollowingView

Sends the NSView message `nextValidKeyView` (page 1762) to *aView*, and if that message returns an NSView, invokes `makeFirstResponder` (page 1841) with the returned NSView.

```
public void selectKeyViewFollowingView(NSView aView)
```

See Also
`selectKeyViewPrecedingView` (page 1852)

selectKeyViewPrecedingView

Sends the NSView message `previousValidKeyView` (page 1765) to *aView*, and if that message returns an NSView, invokes `makeFirstResponder` (page 1841) with the returned NSView.

```
public void selectKeyViewPrecedingView(NSView aView)
```

See Also
`selectKeyViewFollowingView` (page 1852)

selectNextKeyView

This action method searches for a candidate key view and, if it finds one, invokes `makeFirstResponder` (page 1841) to establish it as the first responder.

```
public void selectNextKeyView(Object sender)
```

Discussion
The candidate is one of the following (searched for in this order):

- The current first responder's next valid key view, as returned by NSView's `nextValidKeyView` (page 1762) method
- The object designated as the receiver's initial first responder (using `setInitialFirstResponder` (page 1863)) if it returns `true` to an `acceptsFirstResponder` (page 1189) message

- Otherwise, the initial first responder's next valid key view, which may end up being `null`

See Also
`selectPreviousKeyView` (page 1853)
`selectKeyViewFollowingView` (page 1852)

selectPreviousKeyView

This action method searches for a candidate key view and, if it finds one, invokes `makeFirstResponder` (page 1841) to establish it as the first responder.

```
public void selectPreviousKeyView(Object sender)
```

Discussion
The candidate is one of the following (searched for in this order):

- The current first responder's previous valid key view, as returned by NSView's `previousValidKeyView` (page 1765) method
- The object designated as the receiver's initial first responder (using `setInitialFirstResponder` (page 1863)) if it returns `true` to an `acceptsFirstResponder` (page 1189) message
- Otherwise, the initial first responder's previous valid key view, which may end up being `null`

See Also
`selectNextKeyView` (page 1852)
`selectKeyViewPrecedingView` (page 1852)

sendEvent

This action method dispatches mouse and keyboard events, specified by *`theEvent`*, sent to the receiver by the NSApplication object.

```
public void sendEvent(NSEvent theEvent)
```

Discussion
Never invoke this method directly. A right mouse-down event in a window of an inactive application is not delivered to NSWindow. It is instead delivered to NSApplications's `sendEvent` (page 121) with a window number of 0.

setAcceptsMouseMovedEvents

Sets whether the receiver accepts mouse-moved events and distributes them to its responders.

```
public void setAcceptsMouseMovedEvents(boolean flag)
```

Discussion
If *`flag`* is `true` it does accept them; if *`flag`* is `false` it doesn't. NSWindows don't accept mouse-moved events by default.

See Also

acceptsMouseMovedEvents (page 1816)

setAllowsToolTipsWhenApplicationIsInactive

Sets whether the receiver can display tool tips even when the application is in the background.

```
public void setAllowsToolTipsWhenApplicationIsInactive(boolean allowWhenInactive)
```

Discussion

Default is false. The message does not take effect until the receiver changes to an active state. Note that enabling tool tips in an inactive application will cause the application to do work any time the mouse passes over the window, thus degrading system performance.

Availability

Available in Mac OS X v10.3 and later.

See Also

allowsToolTipsWhenApplicationIsInactive (page 1816)

setAlphaValue

Applies *windowAlpha* to the entire window.

```
public void setAlphaValue(float windowAlpha)
```

See Also

alphaValue (page 1817)

setAspectRatio

Sets the receiver's size aspect ratio to *ratio*, constraining the size of its frame rectangle to integral multiples of this size when the user resizes it.

```
public void setAspectRatio(NSSize ratio)
```

Discussion

An NSWindow's aspect ratio and its resize increments are mutually exclusive attributes. In fact, setting one attribute cancels the setting of the other. For example, to cancel an established aspect ratio setting for an NSWindow, you send the NSWindow object a setResizeIncrements (page 1867) message with the width and height set to 1.0.

The setContentAspectRatio (page 1856) method takes precedence over this method.

See Also

aspectRatio (page 1817)

setFrame (page 1860)

setAutodisplay

Sets whether the receiver automatically displays its views that are marked as needing it.

```
public void setAutodisplay(boolean flag)
```

Discussion

If *flag* is true, views are automatically displayed as needed, typically on each pass through the event loop. If *flag* is false, the receiver or its views must be explicitly displayed.

See Also

isAutodisplay (page 1837)

displayIfNeeded (page 1829)

displayIfNeeded (page 1748) (NSView)

setAutorecalculatesKeyViewLoop

If *flag* is true, the window recalculates the key view loop automatically when views are added or removed.

```
public void setAutorecalculatesKeyViewLoop(boolean flag)
```

Discussion

If *flag* is false, the client code must update the key view loop manually or call recalculateKeyViewLoop (page 1848) to have the window recalculate it.

Availability

Available in Mac OS X v10.4 and later.

See Also

autorecalculatesKeyViewLoop (page 1818)

recalculateKeyViewLoop (page 1848)

setBackgroundColor

Sets the receiver's background color to *aColor*.

```
public void setBackgroundColor(NSColor aColor)
```

See Also

backgroundColor (page 1818)

setBackingType

Sets the receiver's backing store type to *backingType*.

```
public void setBackingType(int backingType)
```

Discussion

The valid backing store types are described in the constants section.

This method can only be used to switch a buffered NSWindow to retained or vice versa; you can't change the backing type to or from nonretained after initializing an NSWindow (an error is generated if you attempt to do so).

See Also
`backingType` (page 1818)

setCanHide

Sets whether the receiver can be hidden during NSApplication's `hide` (page 112) to *`flag`*.

```
public void setCanHide(boolean flag)
```

See Also
`canHide` (page 1820)

setContentAspectRatio

Sets the aspect ratio of the receiver's content view to *`ratio`*, constraining the dimensions of its content rectangle to integral multiples of that ratio when the user resizes it.

```
public void setContentAspectRatio(NSSize ratio)
```

Discussion
You can set a window's content view to any size programmatically, regardless of its aspect ratio. This method takes precedence over `setAspectRatio` (page 1854).

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

See Also
`contentAspectRatio` (page 1822)

setContentMaxSize

Sets the maximum size of the receiver's content view to *`size`*, which is expressed in the receiver's base coordinate system.

```
public void setContentMaxSize(NSSize size)
```

Discussion
The minimum size constraint is enforced for resizing by the user as well as for the `setContentSize` (page 1857) method and the `setFrame...` methods other than `setFrame` (page 1860). This method takes precedence over `setMaxSize` (page 1864).

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

See Also
`contentMaxSize` (page 1823)
`setContentMinSize` (page 1857)

setContentMinSize

Sets the minimum size of the receiver's content view to *size*, which is expressed in the receiver's base coordinate system.

```
public void setContentMinSize(NSSize size)
```

Discussion

The minimum size constraint is enforced for resizing by the user as well as for the `setContentSize` (page 1857) method and the `setFrame...` methods other than `setFrame` (page 1860). This method takes precedence over `setMinSize` (page 1865).

Availability

Available in Mac OS X v10.3 and later.

See Also

`contentMinSize` (page 1823)

`setContentMaxSize` (page 1856)

setContentResizeIncrements

Sets the increments for both height and width by which the receiver's content view can be resized to *increments*.

```
public void setContentResizeIncrements(NSSize increments)
```

Discussion

The *increments* value constrains the width and height of the content rectangle to change by multiples of `increments.width` and `increments.height` when the user resizes the window. You can set a window's size to any width and height programmatically. This method takes precedence over `setResizeIncrements` (page 1867).

Availability

Available in Mac OS X v10.3 and later.

See Also

`contentResizeIncrements` (page 1824)

setContentSize

Sets the size of the receiver's content view to *aSize*, which is expressed in the receiver's base coordinate system.

```
public void setContentSize(NSSize aSize)
```

Discussion

This size in turn alters the size of the NSWindow itself. Note that the window server limits window sizes to 10,000; if necessary, be sure to limit *aSize* relative to the frame rectangle.

See Also

`setFrame` (page 1860)

`contentRectForFrameRect` (page 1814)

`"frameRectForContentRect"` (page 1814)

setContentView

Makes *aView* the receiver's content view; the previous content view is removed from the receiver's view hierarchy.

```
public void setContentView(NSView aView)
```

Discussion
The receiver retains the new content view and owns it thereafter. *aView* is resized to fit precisely within the content area of the NSWindow. You can modify the content view's coordinate system through its bounds rectangle, but can't alter its frame rectangle (that is, its size or location) directly.

See Also
`contentView` (page 1824)
`setContentSize` (page 1857)

setDefaultButtonCell

Makes the key equivalent of *aButtonCell* the Return (or Enter) key, so when the user presses Return that button performs as if clicked.

```
public void setDefaultButtonCell(NSButtonCell aButtonCell)
```

Discussion
See the method description for `defaultButtonCell` (page 1825) for more information.

See Also
`disableKeyEquivalentForDefaultButtonCell` (page 1828)
`enableKeyEquivalentForDefaultButtonCell` (page 1832)

setDelegate

Makes *anObject* the receiver's delegate, without retaining it.

```
public void setDelegate(Object anObject)
```

Discussion
An NSWindow's delegate is inserted in the responder chain after the NSWindow itself and is informed of various actions by the NSWindow through delegation messages.

See Also
`delegate` (page 1826)
`tryToPerform` (page 1871)
`sendActionToTargetFromSender` (page 121) (NSApplication)

setDepthLimit

Sets the depth limit of the receiver to *limit*.

```
public void setDepthLimit(int limit)
```

Discussion

Passing a value of 0 for *limit* sets the depth limit to the receiver's default depth limit; using a value of 0 can be useful for reverting an NSWindow to its initial depth.

See Also

depthLimit (page 1826)
"defaultDepthLimit" (page 1814)
setDynamicDepthLimit (page 1859)

setDisplaysWhenScreenProfileChanges

Sets whether the window context should be updated when the screen profile changes.

```
public void setDisplaysWhenScreenProfileChanges(boolean flag)
```

Discussion

If *flag* is `false`, the screen profile information for the window context never changes. This is the default setting. If *flag* is `true`, the window context may be changed in the following situations:

- A majority of the window is moved to a different screen whose profile is different than the previous screen.
- The ColorSync profile of the current screen changes.

After the window context is updated, the window is told to display itself. If you need to update offscreen caches for the window, you should register to receive the WindowDidChangeScreenProfileNotification.

Availability

Available in Mac OS X v10.4 and later.

See Also

displaysWhenScreenProfileChanges (page 1829)

setDocumentEdited

Sets whether the receiver's document has been edited and not saved to *flag*.

```
public void setDocumentEdited(boolean flag)
```

Discussion

NSWindows are in the "not edited" state by default.

You should invoke this method with an argument of `true` every time the NSWindow's document changes in such a way that it needs to be saved and with an argument of `false` every time it gets saved. Then, before closing the NSWindow you can use isDocumentEdited (page 1837) to determine whether to allow the user a chance to save the document.

setDynamicDepthLimit

Sets whether the receiver changes its depth to match the depth of the screen it's on, or the depth of the deepest screen when it spans multiple screens.

```
public void setDynamicDepthLimit(boolean flag)
```

Discussion

If *flag* is `true`, the depth limit depends on which screen the receiver is on. If *flag* is `false`, the receiver uses either its preset depth limit or the default depth limit. A different, and nondynamic, depth limit can be set with the `setDepthLimit` (page 1858) method.

See Also

`hasDynamicDepthLimit` (page 1835)

`"defaultDepthLimit"` (page 1814)

setExcludedFromWindowsMenu

Sets whether the receiver's title is omitted from the application's Windows menu.

```
public void setExcludedFromWindowsMenu(boolean flag)
```

Discussion

If *flag* is `true` it's omitted; if *flag* is `false`, it's listed when it or its miniaturized window is onscreen. The default is `false`.

See Also

`isExcludedFromWindowsMenu` (page 1837)

setFrame

Sets the origin and size of the receiver's frame rectangle according to *frameRect*, thereby setting its position and size onscreen, and passes a `displayIfNeeded` (page 1829) message down the receiver's view hierarchy, thus redrawing all NSViews that need to be displayed, if *flag* is `true`.

```
public void setFrame(NSRect frameRect, boolean flag)
```

Discussion

Note that the window server limits window position coordinates to ±16,000 and sizes to 10,000.

See Also

`frame` (page 1834)

`setFrameFromString` (page 1861)

`setFrameOrigin` (page 1861)

`setFrameTopLeftPoint` (page 1861)

`setFrameUsingName` (page 1862)

If *animationFlag* is `false`, equivalent to the preceding version of this method. Otherwise, if *animationFlag* is `true`, this method performs a smooth resize of the window, where the total time for the resize is specified by `animationResizeTime` (page 1817).

```
public void setFrame(NSRect frameRect, boolean displayFlag, boolean animationFlag)
```

setFrameAutosaveName

Sets the name used to automatically save the receiver's frame rectangle in the defaults system to *name*.

```
public boolean setFrameAutosaveName(String name)
```

Discussion
If `name` isn't the empty string (""), the receiver's frame is saved as a user default (as described in `saveFrameUsingName` (page 1851)) each time the frame changes. Returns `true` if the name is set successfully, `false` if it's being used as an autosave name by another NSWindow in the application (in which case the receiver's old name remains in effect).

If there is a frame rectangle previously stored for `name` in the user defaults, the receiver's frame is set to this frame rectangle. That is, when you call `setFrameAutosaveName` with a previously used `name`, the window picks up the previously saved setting.

Keep in mind that a window controller may change the window's position when it displays it if window cascading is turned on. To preclude the window controller from changing a window's position from the one saved in the defaults system, you must send `setShouldCascadeWindows:NO` to the window controller.

See Also
`"removeFrameUsingName"` (page 1815)
`stringWithSavedFrame` (page 1870)
`setFrameFromString` (page 1861)
`setFrameUsingName` (page 1862)

setFrameFromString

Sets the receiver's frame rectangle from the string representation `aString`, a representation previously creating using `stringWithSavedFrame` (page 1870).

```
public void setFrameFromString(String aString)
```

Discussion
The frame is constrained according to the receiver's minimum and maximum size settings. This method causes a `windowWillResize` (page 1881) message to be sent to the delegate.

setFrameOrigin

Positions the lower-left corner of the receiver's frame rectangle at `aPoint` in screen coordinates.

```
public void setFrameOrigin(NSPoint aPoint)
```

Discussion
Note that the window server limits window position coordinates to ±16,000.

See Also
`setFrame` (page 1860)
`setFrameTopLeftPoint` (page 1861)

setFrameTopLeftPoint

Positions the top-left corner of the receiver's frame rectangle at `aPoint` in screen coordinates.

```
public void setFrameTopLeftPoint(NSPoint aPoint)
```

Discussion

Note that the window server limits window position coordinates to ±16,000; if necessary, adjust *aPoint* relative to the window's lower-left corner to account for this limit.

See Also

`cascadeTopLeftFromPoint` (page 1821)
`setFrame` (page 1860)
`setFrameOrigin` (page 1861)

setFrameUsingName

Sets the receiver's frame rectangle by reading the rectangle data stored in *name* from the defaults system.

```
public boolean setFrameUsingName(String name)
```

Discussion

The frame is constrained according to the receiver's minimum and maximum size settings. This method causes a `windowWillResize` (page 1881) message to be sent to the delegate. Returns `true` if *name* is read and the frame is set successfully; otherwise returns `false`.

See Also

`setFrameAutosaveName` (page 1860)
`"removeFrameUsingName"` (page 1815)
`stringWithSavedFrame` (page 1870)
`setFrameFromString` (page 1861)

Sets the receiver's frame rectangle by reading the rectangle data stored in *name* from the defaults system. Send this method with *force* set to `true` to use the preceding version of this method on a nonresizable window.

```
public boolean setFrameUsingName(String name, boolean force)
```

setHasShadow

Sets whether the receiver has a shadow to *hasShadow*.

```
public void setHasShadow(boolean hasShadow)
```

Discussion

If the shadow setting changes, the window shadow is invalidated, forcing the window shadow to be recomputed.

See Also

`hasShadow` (page 1836)
`invalidateShadow` (page 1837)

setHidesOnDeactivate

Sets whether the receiver is removed from the screen when the application is inactive.

```
public void setHidesOnDeactivate(boolean flag)
```

Discussion

If *flag* is `true`, the receiver is hidden (taken out of the screen list) when the application stops being the active application. If *flag* is `false`, the receiver stays onscreen. The default for NSWindow is `false`; the default for NSPanel is `true`.

See Also

`hidesOnDeactivate` (page 1836)

setIgnoresMouseEvents

Specifies whether the receiver is transparent to mouse clicks and other mouse events, allowing overlay windows.

```
public void setIgnoresMouseEvents(boolean flag)
```

Availability

Available in Mac OS X v10.2 and later.

See Also

`ignoresMouseEvents` (page 1836)

setInitialFirstResponder

Sets *aView* as the NSView that's made first responder (also called the key view) the first time the receiver is placed onscreen.

```
public void setInitialFirstResponder(NSView aView)
```

See Also

`initialFirstResponder` (page 1836)

setLevel

Sets the receiver's window level to *newLevel*.

```
public void setLevel(int newLevel)
```

Discussion

Some useful predefined values, ordered from lowest to highest, are described in the constants section.

Each level in the list groups windows within it in front of those in all preceding groups. Floating windows, for example, appear in front of all normal-level windows. When a window enters a new level, it's ordered in front of all its peers in that level.

The constant `TornOffMenuWindowLevel` is preferable to its synonym, `SubmenuWindowLevel`.

See Also

`level` (page 1841)

`orderWindow` (page 1845)

`orderFront` (page 1844)

`orderBack` (page 1844)

setMaxSize

Sets the maximum size to which the receiver's frame (including its title bar) can be sized to *aSize*.

```
public void setMaxSize(NSSize aSize)
```

Discussion

The maximum size constraint is enforced for resizing by the user as well as for the `setFrame...` methods other than `setFrame` (page 1860). Note that the window server limits window sizes to 10,000.

The default maximum size of a window is `{FLT_MAX, FLT_MAX}`. Once the maximum size of a window has been set, there is no way to reset it other than specifying this default maximum size.

The setContentMaxSize (page 1856) method takes precedence over this method.

See Also

`maxSize` (page 1842)

`setMinSize` (page 1865)

`setAspectRatio` (page 1854)

`setResizeIncrements` (page 1867)

setMiniwindowImage

Sets the receiver's custom miniaturized window image to *anImage*.

```
public void setMiniwindowImage(NSImage anImage)
```

Discussion

When the user minimizes the window, the dock displays *anImage* in the corresponding dock tile, scaling it as needed to fit in the tile. If you do not specify a custom image using this method, the dock creates one for you automatically.

You can also call this method as needed to change the miniaturized window image. Typically, you would specify a custom image immediately prior to a window being minimized—when the system posts an `WindowWillMiniaturizeNotification` (page 1885). You can call this method while the window is minimized to update the current image in the dock. However, this method is not recommended for creating complex animations in the dock.

Support for custom images is disabled by default. To enable support, set the `AppleDockIconEnabled` key to `true` when first registering your application's user defaults. You must set this key prior to calling the `init` method of NSApplication, which reads the current value of the key.

See Also

`miniwindowImage` (page 1842)

`isMiniaturized` (page 1838)

setMiniwindowTitle

Sets the title of the receiver's miniaturized counterpart to *aString* and redisplays it.

```
public void setMiniwindowTitle(String aString)
```

Discussion

A miniaturized window's title normally reflects that of its full-size counterpart, abbreviated to fit if necessary. Although this method allows you to set the miniaturized window's title explicitly, changing the full-size NSWindow's title (through `setTitle` (page 1868) or `setTitleWithRepresentedFilename` (page 1868)) automatically changes the miniaturized window's title as well.

See Also

`miniwindowTitle` (page 1843)

setMinSize

Sets the minimum size to which the receiver's frame (including its title bar) can be sized to *aSize*.

```
public void setMinSize(NSSize aSize)
```

Discussion

The minimum size constraint is enforced for resizing by the user as well as for the `setFrame...` methods other than `setFrame` (page 1860).

The `setContentMinSize` (page 1857) takes precedence over this method.

See Also

`minSize` (page 1843)

`setMaxSize` (page 1864)

`setAspectRatio` (page 1854)

`setResizeIncrements` (page 1867)

setMovableByWindowBackground

Sets whether the receiver is movable by clicking and dragging anywhere in its background.

```
public void setMovableByWindowBackground(boolean flag)
```

Discussion

A window with a style mask of `TexturedBackgroundWindowMask` get set to `true` by default. Sheets and drawers cannot be movable by window background.

Availability

Available in Mac OS X v10.2 and later.

See Also

`isMovableByWindowBackground` (page 1838)

setOneShot

Sets whether the window device that the receiver manages should be freed when it's removed from the screen list (and another one created if it's returned to the screen) to *flag*.

```
public void setOneShot(boolean flag)
```

Discussion

Freeing the window device when it's removed from the screen list can result in memory savings and performance improvement for NSWindows that don't take long to display. It's particularly appropriate for NSWindows the user might use once or twice but not display continually. The default is `false`.

See Also

`isOneShot` (page 1839)

setOpaque

Sets whether the receiver is opaque to *isOpaque*.

```
public void setOpaque(boolean isOpaque)
```

See Also

`isOpaque` (page 1839)

setParentWindow

For use by subclasses when setting the parent window in the receiver.

```
public void setParentWindow(NSWindow window)
```

Discussion

You should call super if overriding.

Availability

Available in Mac OS X v10.2 and later.

See Also

`removeChildWindow` (page 1849)

`childWindows` (page 1821)

`parentWindow` (page 1846)

`addChildWindow` (page 1816)

setPreservesContentDuringLiveResize

If *flag* is `true`, the window optimizes live resize operations by invalidating only the view contents that changed; this is the default setting.

```
public void setPreservesContentDuringLiveResize(boolean flag)
```

Discussion

If *flag* is `false`, this optimization is disabled for the window and all of its contained views.

You might consider disabling this optimization for the window if none of the window's contained views can take advantage of it. Disabling the optimization for the window prevents it from checking each view to see if the optimization is supported.

Availability

Available in Mac OS X v10.4 and later.

See Also

preservesContentDuringLiveResize (page 1848)

setReleasedWhenClosed

Sets whether the receiver is merely hidden (false) or hidden and then released (true) when it receives a close message.

```
public void setReleasedWhenClosed(boolean flag)
```

Discussion

The default for NSWindow is true; the default for NSPanel is false. Release when closed, however, is ignored for windows owned by window controllers.

See Also

close (page 1821)

isReleasedWhenClosed (page 1839)

setRepresentedFilename

Sets the name of the file the receiver represents to *path*.

```
public void setRepresentedFilename(String path)
```

See Also

representedFilename (page 1849)

setTitleWithRepresentedFilename (page 1868)

setResizeIncrements

Restricts the user's ability to resize the receiver so the width and height change by multiples of *increments*.width and *increments*.height as the user resizes the window.

```
public void setResizeIncrements(NSSize increments)
```

Discussion

The width and height increments should be whole numbers, 1.0 or greater. Whatever the current resize increments, you can set an NSWindow's size to any height and width programmatically.

Resize increments and aspect ratio are mutually exclusive attributes. For more information, see setAspectRatio (page 1854).

The setContentResizeIncrements (page 1857) method takes precedence over this method.

See Also

resizeIncrements (page 1851)

setFrame (page 1860)

setShowsResizeIndicator

Sets whether the receiver's resize indicator is visible to *show*.

```
public void setShowsResizeIndicator(boolean show)
```

Discussion
This method does not affect whether the receiver is resizable.

See Also
`showsResizeIndicator` (page 1869)

setShowsToolbarButton

If *flag* is `true`, the window title bar is updated to display the standard toolbar button.

```
public void setShowsToolbarButton(boolean flag)
```

Discussion
If *flag* is `false`, the button is not displayed. If the window does not have a toolbar, this method has no effect.

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

See Also
`showsToolbarButton` (page 1870)

setTitle

Sets the string that appears in the receiver's title bar (if it has one) to *aString* and displays the title.

```
public void setTitle(String aString)
```

Discussion
Also sets the title of the receiver's miniaturized window.

See Also
`title` (page 1870)
`setTitleWithRepresentedFilename` (page 1868)
`setMiniwindowTitle` (page 1864)

setTitleWithRepresentedFilename

Sets *path* as the receiver's title, formatting it as a file-system path, and records *path* as the receiver's associated filename using `setRepresentedFilename` (page 1867).

```
public void setTitleWithRepresentedFilename(String path)
```

Discussion
The filename—not the pathname—is displayed in the receiver's title bar.

This method also sets the title bar of the receiver's minimized window.

See Also
`title` (page 1870)
`setTitle` (page 1868)
`setMiniwindowTitle` (page 1864)

setToolbar

Sets the receiver's toolbar to *toolbar*.

```
public void setToolbar(NSToolbar toolbar)
```

Discussion
See the NSToolbar (page 1693) class description for additional information.

See Also
`toolbar` (page 1871)

setViewsNeedDisplay

Sets whether the receiver's views need display (`true`) or do not need display (`false`) to *flag*.

```
public void setViewsNeedDisplay(boolean flag)
```

Discussion
You should rarely need to invoke this method; NSView's `setNeedsDisplay` (page 1779) and similar methods invoke it automatically.

See Also
`viewsNeedDisplay` (page 1873)

setWindowController

Set's the receiver's window controller to be *windowController*.

```
public void setWindowController(NSWindowController windowController)
```

See Also
`windowController` (page 1873)

showsResizeIndicator

Returns whether the receiver's resize indicator is visible.

```
public boolean showsResizeIndicator()
```

See Also
`setShowsResizeIndicator` (page 1868)

showsToolbarButton

Returns `true` if the standard toolbar button is currently displayed; otherwise, returns `false`.

```
public boolean showsToolbarButton()
```

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

See Also
`setShowsToolbarButton` (page 1868)

standardWindowButton

Return the given standard *button* if it is in the window view hierarchy.

```
public NSButton standardWindowButton(int button)
```

Availability
Available in Mac OS X v10.2 and later.

See Also
`"standardWindowButtonForStyleMask"` (page 1815)

stringWithSavedFrame

Returns a string that represents the receiver's frame rectangle in a format that can be used with a later `setFrameFromString` (page 1861) message.

```
public String stringWithSavedFrame()
```

styleMask

Returns the receiver's style mask, indicating what kinds of control items it displays.

```
public int styleMask()
```

Discussion
See the information about the style mask in the constants section. An NSWindow's style is set when the object is initialized. Once set, it can't be changed.

title

Returns either the string that appears in the title bar of the receiver, or the path to the represented file.

```
public String title()
```

Discussion
If the title has been set using `setTitleWithRepresentedFilename` (page 1868), then this method returns the file's path, as described in `setTitleWithRepresentedFilename`.

See Also

`setTitle` (page 1868)

`setTitleWithRepresentedFilename` (page 1868)

toggleToolbarShown

The action method for the "Hide Toolbar" menu item (which alternates with "Show Toolbar").

```
public void toggleToolbarShown(Object sender)
```

Discussion

See the NSToolbar (page 1693) class description for additional information.

toolbar

Returns the receiver's toolbar.

```
public NSToolbar toolbar()
```

Discussion

See the NSToolbar (page 1693) class description for additional information.

See Also

`setToolbar` (page 1869)

tryToPerform

Dispatches action messages with *anObject* as the argument.

```
public boolean tryToPerform(NSSelector anAction, Object anObject)
```

Discussion

The receiver tries to perform the method *anAction* using its inherited NSResponder method `tryToPerform` (page 1199). If the receiver doesn't perform *anAction*, the delegate is given the opportunity to perform it. If either the receiver or its delegate accepts *anAction*, this method returns `true`; otherwise it returns `false`.

unregisterDraggedTypes

Unregisters the receiver as a possible destination for dragging operations.

```
public void unregisterDraggedTypes()
```

See Also

`registerForDraggedTypes` (page 1849)

update

Updates the window.

```
public void update()
```

Discussion

The default implementation of this method does nothing more than post a `WindowDidUpdateNotification` (page 1885) to the default notification center. A subclass can override 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.

An NSWindow is automatically sent an `update` (page 1871) message on every pass through the event loop and before it's displayed onscreen. You can manually cause an `update` (page 1871) message to be sent to all visible NSWindows through NSApplication's `updateWindows` (page 127) method.

See Also

`setWindowsNeedUpdate` (page 123) (NSApplication)

`updateWindows` (page 127) (NSApplication)

useOptimizedDrawing

Informs the receiver whether to optimize focusing and drawing when displaying its NSViews.

```
public void useOptimizedDrawing(boolean flag)
```

Discussion

The optimizations may prevent sibling subviews from being displayed in the correct order—which matters only if the subviews overlap. You should always set *flag* to `true` if there are no overlapping subviews within the NSWindow. The default is `false`.

userSpaceScaleFactor

Returns the scale factor applied to the window.

```
public float userSpaceScaleFactor()
```

Discussion

Clients can multiply view coordinates by the returned scale factor to get a set of new coordinates that are scaled to the resolution of the target screen. For example, if the scale factor is 1.25 and the view frame size is 80 x 80, the actual size of the view frame is 100 x 100 pixels on the target screen.

Availability

Available in Mac OS X v10.4 and later.

validRequestorForTypes

Searches for an object that responds to a Services request by providing input of *sendType* and accepting output of *returnType*.

```
public Object validRequestorForTypes(String sendType, String returnType)
```

Discussion

Returns that object, or `null` if none is found.

Messages to perform this method are initiated by the Services menu. It's part of the mechanism that passes `validRequestorForTypes` messages up the responder chain.

This method works by forwarding the message to the receiver's delegate if it responds (and provided it isn't an NSResponder with its own next responder). If the delegate doesn't respond to the message or returns `null` when sent it, this method forwards the message to the NSApplication object. If the NSApplication object returns `null`, this method also returns `null`. Otherwise this method returns the object returned by the delegate or the NSApplication object.

See Also

`validRequestorForTypes` (page 1200) (NSResponder)
`validRequestorForTypes` (page 128) (NSApplication)

viewsNeedDisplay

Returns `true` if any of the receiver's NSView's need to be displayed, `false` otherwise.

```
public boolean viewsNeedDisplay()
```

See Also

`setViewsNeedDisplay` (page 1869)

windowController

Returns the receiver's window controller.

```
public Object windowController()
```

See Also

`setWindowController` (page 1869)

windowNumber

Returns the window number of the receiver's window device.

```
public int windowNumber()
```

Discussion

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. This number can be used to identify the window device with the `orderWindow` (page 1845) method.

If the receiver doesn't have a window device, the value returned will be equal to or less than 0.

See Also

`setOneShot` (page 1865)

worksWhenModal

Returns `true` if the receiver is able to receive keyboard and mouse events even when some other window is being run modally, `false` otherwise.

```
public boolean worksWhenModal()
```

Discussion

NSWindow's implementation of this method returns `false`. Only subclasses of NSPanel should override this default.

See Also

`setWorksWhenModal` (page 1054) (NSPanel)

zoom

This action method toggles the size and location of the window between its standard state (provided by the application as the "best" size to display the window's data) and its user state (a new size and location the user may have set by moving or resizing the window).

```
public void zoom(Object sender)
```

Discussion

For more information on the standard and user states, see `windowWillUseStandardFrame` (page 1882).

The `zoom` method is typically invoked after a user clicks the window's zoom box but may also be invoked programmatically from the `performZoom` (page 1847) method. It performs the following steps:

1. Invokes the `windowWillUseStandardFrame` (page 1882) method, if the delegate or the window class implements it, to obtain a "best fit" frame for the window. If neither the delegate nor the window class implements the method, uses a default frame that nearly fills the current screen, which is defined to be the screen containing the largest part of the window's current frame.

2. Adjusts the resulting frame, if necessary, to fit on the current screen.

3. Compares the resulting frame to the current frame to determine whether the window's standard frame is currently displayed. If the current frame is within a few pixels of the standard frame in size and location, it is considered a match.

4. Determines a new frame. If the window is currently in the standard state, the new frame represents the user state, saved during a previous zoom. If the window is currently in the user state, the new frame represents the standard state, computed in step 1 above. If there is no saved user state because there has been no previous zoom, the size and location of the window do not change.

5. Determines whether the window currently allows zooming. By default, zooming is allowed. If the window's delegate implements the `windowShouldZoom` (page 1880) method, `zoom` invokes that method. If the delegate doesn't implement the method but the window does, `zoom` invokes the window's version. `windowShouldZoom` returns `false` if zooming is not currently allowed.

6. If the window currently allows zooming, sets the new frame.

See Also

`isZoomed` (page 1840)

Constants

These constants specify the presence of a title and various buttons in an NSWindow's border. Either it can be BorderlessWindowMask, or it can contain any of the following options, combined using the C bitwise OR operator:

Constant	Description
BorderlessWindowMask	The NSWindow displays none of the usual peripheral elements. Useful only for display or caching purposes.
TitledWindowMask	The NSWindow displays a title bar.
ClosableWindowMask	The NSWindow displays a close button.
MiniaturizableWindowMask	The NSWindow displays a miniaturize button.
ResizableWindowMask	The NSWindow displays a resize control.
TexturedBackgroundWindowMask	The NSWindow displays with a metal-textured background. Additionally, the NSWindow may be moved by clicking and dragging anywhere in the window background. A bordered window with this mask gets rounded bottom corners.

These constants specify how the drawing done in a window is buffered by the window device:

Constant	Description
Buffered	The NSWindow renders all drawing into a display buffer and then flushes it to the screen.
Retained	The NSWindow uses a buffer, but draws directly to the screen where possible and to the buffer for obscured portions.
NonRetained	The NSWindow draws directly to the screen without using any buffer.

These constants specify the window's level. The stacking of levels takes precedence over the stacking of windows within each level. That is, even the bottom window in a level will obscure even the top window of the next level down. Levels are listed in order from lowest to highest. These constants are mapped (using #define) to corresponding elements in the window level enum in Core Graphics.

Constant	Description
NormalWindowLevel	The default level for NSWindow objects.
FloatingWindowLevel	Useful for floating palettes.
SubmenuWindowLevel	Reserved for submenus. Synonymous with TornOffMenuWindowLevel, which is preferred.
TornOffMenuWindowLevel	The level for a torn-off menu. Synonymous with SubmenuWindowLevel.

Constant	Description
`ModalPanelWindowLevel`	The level for a modal panel.
`MainMenuWindowLevel`	Reserved for the application's main menu.
`StatusWindowLevel`	The level for a status window.
`PopUpMenuWindowLevel`	The level for a pop-up menu.
`ScreenSaverWindowLevel`	The level for a screen saver.

These constants let you specify how a window is ordered relative to another window. For more information, see `orderWindow` (page 1845).

Constant	Description
`Above`	Moves the window above the indicated window.
`Below`	Moves the window below the indicated window.
`Out`	Moves the window off the screen.

These constants specify the direction a window is currently using to change the key view. They're used by `keyViewSelectionDirection` (page 1840).

Constant	Description
`DirectSelection`	The receiver isn't traversing the key view loop.
`SelectingNext`	The receiver is proceeding to the next valid key view.
`SelectingPrevious`	The receiver is proceeding to the previous valid key view.

These constants provide a way to access standard title bar widgets:

Constant	Description
`WindowCloseButton`	The close button.
`WindowMiniaturizeButton`	The miniaturize button.
`WindowZoomButton`	The zoom button.
`WindowToolbarButton`	The toolbar button.
`WindowDocumentIconButton`	The document icon button.

This constant provides a way to manage scaling factors:

Constant	Description
`UnscaledWindowMask`	Specifies that the window is created without any scaling factors applied. The client is responsible for all scaling operations in the window. Such a window returns `1.0` from its `userSpaceScaleFactor` method. Available in Mac OS X v10.4 and later.

This constant controls the look of a window and its toolbar:

Constant	Description
`UnifiedTitleAndToolbarWindowMask`	Specifies a window whose toolbar and titlebar are rendered on a single continuous background. Available in Mac OS X v10.4 and later.

Delegate Methods

windowDidBecomeKey

Sent by the default notification center immediately after an NSWindow has become key.

```
public abstract void windowDidBecomeKey(NSNotification aNotification)
```

Discussion

aNotification is always `WindowDidBecomeKeyNotification` (page 1883). You can retrieve the NSWindow object in question by sending `object` to *aNotification*.

windowDidBecomeMain

Sent by the default notification center immediately after an NSWindow has become main.

```
public abstract void windowDidBecomeMain(NSNotification aNotification)
```

Discussion

aNotification is always `WindowDidBecomeMainNotification` (page 1883). You can retrieve the NSWindow object in question by sending `object` to *aNotification*.

windowDidChangeScreen

Sent by the default notification center immediately after an NSWindow has changed screens.

```
public abstract void windowDidChangeScreen(NSNotification aNotification)
```

Discussion

aNotification is always `WindowDidChangeScreenNotification` (page 1883). You can retrieve the NSWindow object in question by sending `object` to *aNotification*.

windowDidChangeScreenProfile

Sent by the default notification center immediately after an NSWindow has changed screen display profiles.

```
public abstract void windowDidChangeScreenProfile(NSNotification aNotification)
```

Discussion
aNotification is always WindowDidChangeScreenProfileNotification (page 1883). You can retrieve the NSWindow object in question by sending object to *aNotification*.

windowDidDeminiaturize

Sent by the default notification center immediately after an NSWindow has been deminiaturized.

```
public abstract void windowDidDeminiaturize(NSNotification aNotification)
```

Discussion
aNotification is always WindowDidDeminiaturizeNotification (page 1884). You can retrieve the NSWindow object in question by sending object to *aNotification*.

windowDidEndSheet

Sent by the default notification center immediately after an NSWindow closes a sheet.

```
public abstract void windowDidEndSheet(NSNotification aNotification)
```

Discussion
aNotification is always WindowDidEndSheetNotification (page 1884). You can retrieve the NSWindow object in question by sending object to *aNotification*.

windowDidExpose

Sent by the default notification center immediately after an NSWindow has been exposed.

```
public abstract void windowDidExpose(NSNotification aNotification)
```

Discussion
aNotification is always WindowDidExposeNotification (page 1884). You can retrieve the NSWindow object in question by sending object to *aNotification*.

windowDidMiniaturize

Sent by the default notification center immediately after an NSWindow has been miniaturized.

```
public abstract void windowDidMiniaturize(NSNotification aNotification)
```

Discussion
aNotification is always WindowDidMiniaturizeNotification (page 1884). You can retrieve the NSWindow object in question by sending object to *aNotification*.

windowDidMove

Sent by the default notification center immediately after an NSWindow has been moved.

```
public abstract void windowDidMove(NSNotification aNotification)
```

Discussion

aNotification is always WindowDidMoveNotification (page 1884). You can retrieve the NSWindow object in question by sending object to *aNotification*.

windowDidResignKey

Sent by the default notification center immediately after an NSWindow has resigned its status as key window.

```
public abstract void windowDidResignKey(NSNotification)
```

Discussion

aNotification is always WindowDidResignKeyNotification (page 1884). You can retrieve the NSWindow object in question by sending object to *aNotification*.

windowDidResignMain

Sent by the default notification center immediately after an NSWindow has resigned its status as main window.

```
public abstract void windowDidResignMain(NSNotification aNotification)
```

Discussion

aNotification is always WindowDidResignMainNotification (page 1884). You can retrieve the NSWindow object in question by sending object to *aNotification*.

windowDidResize

Sent by the default notification center immediately after an NSWindow has been resized.

```
public abstract void windowDidResize(NSNotification aNotification)
```

Discussion

aNotification is always WindowDidResizeNotification (page 1885). You can retrieve the NSWindow object in question by sending object to *aNotification*.

windowDidUpdate

Sent by the default notification center immediately after an NSWindow receives an update (page 1871) message.

```
public abstract void windowDidUpdate(NSNotification aNotification)
```

Discussion

aNotification is always WindowDidUpdateNotification (page 1885). You can retrieve the NSWindow object in question by sending object to *aNotification*.

windowShouldClose

Invoked when the user attempts to close the window or when the NSWindow receives a `performClose` (page 1846) message.

```
public abstract boolean windowShouldClose(Object sender)
```

Discussion

The delegate can return `false` to prevent *sender* from closing.

This method may not always be called. Specifically, this method is not called when a user quits an application. Additional information on application termination can be found in Graceful Application Termination.

windowShouldZoom

Invoked just before *sender* is zoomed.

```
public abstract boolean windowShouldZoom(NSWindow sender, NSRect newFrame)
```

Discussion

Zooming will change the frame of *sender* to *newFrame*. The delegate can return `false` to prevent *sender* from zooming.

See Also

`windowWillUseStandardFrame` (page 1882)

windowWillBeginSheet

Sent by the default notification center immediately before an NSWindow opens a sheet.

```
public abstract void windowWillBeginSheet(NSNotification aNotification)
```

Discussion

aNotification is always `WindowWillBeginSheetNotification` (page 1885). You can retrieve the NSWindow object in question by sending `object` to *aNotification*.

windowWillClose

Sent by the default notification center immediately before an NSWindow closes.

```
public abstract void windowWillClose(NSNotification aNotification)
```

Discussion

aNotification is always `WindowWillCloseNotification` (page 1885). You can retrieve the NSWindow object in question by sending `object` to *aNotification*.

windowWillMiniaturize

Sent by the default notification center immediately before an NSWindow is miniaturized.

```
public abstract void windowWillMiniaturize(NSNotification aNotification)
```

Discussion

aNotification is always `WindowWillMiniaturizeNotification` (page 1885). You can retrieve the NSWindow object in question by sending `object` to *aNotification*.

windowWillMove

Sent by the default notification center immediately before an NSWindow is moved.

```
public abstract void windowWillMove(NSNotification aNotification)
```

Discussion

aNotification is always `WindowWillMoveNotification` (page 1885). You can retrieve the NSWindow object in question by sending `object` to *aNotification*.

windowWillPositionSheet

Sent to the delegate just before the animation of a sheet, giving it the opportunity to return a custom location for the attachment of the sheet (*sheet*) to the window (*window*).

```
public abstract NSRect windowWillPositionSheet(NSWindow window, NSWindow sheet,
    NSRect rect)
```

Discussion

This method is also invoked whenever the user resizes *window* while *sheet* is attached. The default sheet location, passed in *rect*, is just under the title bar of the window, aligned with the left and right edges of the window.

This method is useful in many situations. If your window has a toolbar, for example, you can specify a location for the sheet that is just below it. If you want the sheet associated with a certain control or view, you could position the sheet so that it appears to originate from the object (though genie animation) or is positioned next to it.

Neither the *rect* parameter nor the returned NSRect define the boundary of the sheet. They indicate where the top-left edge of the sheet is attached to the window. The origin is expressed in window coordinates; the default `origin.y` value is the height of the content view and the default `origin.x` value is zero. The `size.width` value indicates the width and behavior of the initial animation; if `size.width` is narrower than the sheet, the sheet genies out from the specified location, and if `size.width` is wider than the sheet, the sheet slides out. You cannot affect the size of the sheet through the `size.width` and `size.height` fields. It is recommended that you specify zero for the `size.height` value as this field may have additional meaning in a future release.

Availability

Available in Mac OS X v10.3 and later.

windowWillResize

Invoked when *sender* is being resized (whether by the user or through one of the `setFrame...` methods other than `setFrame` (page 1860)).

```
public abstract NSSize windowWillResize(NSWindow sender, NSSize proposedFrameSize)
```

Discussion

proposedFrameSize contains the size (in screen coordinates) the sender will be resized to. To resize to a different size, simply return the desired size from this method; to avoid resizing, return the current size. The NSWindow's minimum and maximum size constraints have already been applied when this method is invoked.

While the user is resizing an NSWindow, the delegate is sent a series of `windowWillResize` messages as the NSWindow's outline is dragged. The NSWindow's outline is displayed at the constrained size as set by this method.

windowWillReturnFieldEditor

Invoked when the field editor of *sender* is requested by *anObject*.

```
public abstract Object windowWillReturnFieldEditor(NSWindow sender, Object anObject)
```

Discussion

If the delegate's implementation of this method returns an object other than `null`, the NSWindow substitutes it for the field editor and returns it to *anObject*.

This method may be called multiple times while a control is first responder. Therefore, you must return the same field editor object for the control while the control is being edited.

See Also

`fieldEditorForObject` (page 1832)

windowWillReturnUndoManager

Invoked when the undo manager for *sender* is requested.

```
public abstract NSUndoManager windowWillReturnUndoManager(NSWindow sender)
```

Discussion

Returns the appropriate undo manager. If this method is not implemented, the NSWindow creates an NSUndoManager for the window.

windowWillUseStandardFrame

Invoked by the `zoom` (page 1874) method while determining a frame the *sender* may be zoomed to.

```
public abstract NSRect windowWillUseStandardFrame(NSWindow sender, NSRect
    defaultFrame)
```

Discussion

Returns the standard frame (described below) for a window. The *defaultFrame* parameter passed in is the size of the current screen, which is the screen containing the largest part of the window's current frame, possibly reduced on the top, bottom, left, or right, depending on the current interface style. The frame is reduced on the top to leave room for the menu bar.

The standard frame for a window should supply the size and location that are "best" for the type of information shown in the window, taking into account the available display or displays. For example, the best width for a window that displays a word-processing document is the width of a page or the width of the display, whichever is smaller. The best height can be determined similarly. On return from this method, the `zoom` (page 1874) method modifies the returned standard frame, if necessary, to fit on the current screen.

To customize the standard state, you implement `windowWillUseStandardFrame` in the class of the window's delegate or, if necessary, in a window subclass. Your version should return a suitable standard frame, based on the currently displayed data or other factors.

See Also

`windowShouldZoom` (page 1880)

`zoom` (page 1874)

Notifications

WindowDidBecomeKeyNotification

Posted whenever an NSWindow becomes the key window.

The notification object is the NSWindow that has become key. This notification does not contain a *`userInfo`* dictionary.

WindowDidBecomeMainNotification

Posted whenever an NSWindow becomes the main window.

The notification object is the NSWindow that has become main. This notification does not contain a *`userInfo`* dictionary.

WindowDidChangeScreenNotification

Posted whenever a portion of an NSWindow's frame moves onto or off of a screen.

The notification object is the NSWindow that has changed screens. This notification does not contain a *`userInfo`* dictionary.

This notification is not sent in Mac OS X versions earlier than 10.4.

WindowDidChangeScreenProfileNotification

Posted whenever the display profile for the screen containing the window changes.

This notification is sent only if the window returns `true` from `displaysWhenScreenProfileChanges` (page 1829). This notification may be sent when a majority of the window is moved to a different screen (whose profile is also different from the previous screen) or when the ColorSync profile for the current screen changes.

The notification object is the NSWindow whose profile changed. This notification does not contain a *`userInfo`* dictionary.

Availability

Available in Mac OS X v10.4 and later.

WindowDidDeminiaturizeNotification

Posted whenever an NSWindow is deminiaturized.

The notification object is the NSWindow that has been deminiaturized. This notification does not contain a `userInfo` dictionary.

WindowDidEndSheetNotification

Posted whenever an NSWindow closes an attached sheet.

The notification object is the NSWindow that contained the sheet. This notification does not contain a `userInfo` dictionary.

WindowDidExposeNotification

Posted whenever a portion of a nonretained NSWindow is exposed, whether by being ordered in front of other windows or by other windows being removed from in front of it.

The notification object is the NSWindow that has been exposed. The `userInfo` dictionary contains the following information:

Key	Value
`"NSExposedRect"`	The rectangle that has been exposed (NSRect).

WindowDidMiniaturizeNotification

Posted whenever an NSWindow is miniaturized.

The notification object is the NSWindow that has been miniaturized. This notification does not contain a `userInfo` dictionary.

WindowDidMoveNotification

Posted whenever an NSWindow is moved.

The notification object is the NSWindow that has moved. This notification does not contain a `userInfo` dictionary.

WindowDidResignKeyNotification

Posted whenever an NSWindow resigns its status as key window.

The notification object is the NSWindow that has resigned its key window status. This notification does not contain a `userInfo` dictionary.

WindowDidResignMainNotification

Posted whenever an NSWindow resigns its status as main window.

The notification object is the NSWindow that has resigned its main window status. This notification does not contain a `userInfo` dictionary.

WindowDidResizeNotification

Posted whenever an NSWindow's size changes.

The notification object is the NSWindow whose size has changed. This notification does not contain a `userInfo` dictionary.

WindowDidUpdateNotification

Posted whenever an NSWindow receives an `update` (page 1871) message.

The notification object is the NSWindow that received the `update` (page 1871) message. This notification does not contain a `userInfo` dictionary.

WindowWillBeginSheetNotification

Posted whenever an NSWindow is about to open a sheet.

The notification object is the NSWindow that is about to open the sheet. This notification does not contain a `userInfo` dictionary.

WindowWillCloseNotification

Posted whenever an NSWindow is about to close.

The notification object is the NSWindow that is about to close. This notification does not contain a `userInfo` dictionary.

WindowWillMiniaturizeNotification

Posted whenever an NSWindow is about to be miniaturized.

The notification object is the NSWindow that is about to be miniaturized. This notification does not contain a `userInfo` dictionary.

WindowWillMoveNotification

Posted whenever an NSWindow is about to move.

The notification object is the NSWindow that is about to move. This notification does not contain a `userInfo` dictionary.

NSWindowController

Inherits from	NSResponder : NSObject
Implements	NSCoding
Package:	com.apple.cocoa.application
Companion guide	Document-Based Applications Overview

Overview

An NSWindowController object manages a window, usually a window stored in a nib file. This management entails:

- Loading and displaying the window
- Closing the window when appropriate
- Customizing the window's title
- Storing the window's frame (size and location) in the defaults database
- Cascading the window in relation to other document windows of the application

An NSWindowController can manage a window by itself or as a role player in the Application Kit's document-based architecture, which also includes NSDocument and NSDocumentController objects. In this architecture, an NSWindowController is created and managed by a "document" (an instance of an NSDocument subclass) and, in turn, keeps a reference to the document.

The relationship between an NSWindowController (or, simply, a window controller) and a nib file is important. Although a window controller can manage a programmatically created window, it usually manages a window in a nib file. The nib file can contain other top-level objects, including other windows, but the window controller's responsibility is this primary window. The window controller is usually the owner of the nib file, even when it is part of a document-based application. Regardless of who is the file's owner, the window controller is responsible for freeing all top-level objects in the nib file it loads.

For simple documents—that is, documents with only one nib file containing a window—you need do little directly with NSWindowController. The Application Kit will create one for you. However, if the default window controller is not sufficient, you can create a custom subclass of NSWindowController. For documents with multiple windows or panels, your document must create separate instances of NSWindowController (or of custom subclasses of NSWindowController), one for each window or panel. An example is a CAD application that has different windows for side, top, and front views of drawn objects. What you do in your NSDocument subclass determines whether the default NSWindowController or separately created and configured NSWindowController objects are used.

Subclassing NSWindowController

You should create a subclass of NSWindowController when you want to augment the default behavior, such as to give the window a custom title or to perform some setup tasks before the window is loaded. In your class's initialization method, be sure to invoke on `super` one of the constructors. Which one depends on whether the window object originates in a nib file or is programmatically created.

Three NSWindowController methods are most commonly overridden:

Method Name	Description
`windowWillLoad` (page 1897)	Override to perform tasks before the window nib file is loaded.
`windowDidLoad` (page 1895)	Override to perform tasks after the window nib file is loaded.
`windowTitleForDocumentDisplayName` (page 1896)	Override to customize the window title.

You can also override `loadWindow` (page 1892) to get different nib-searching or nib-loading behavior, although there is usually no need to do this.

Tasks

Constructors

`NSWindowController` (page 1890)
Creates an NSWindowController object with no window object to manage.

Loading and Display the Window

`loadWindow` (page 1892)
Loads the receiver's window from the nib file.

`showWindow` (page 1894)
This action method displays the window associated with the receiver.

`isWindowLoaded` (page 1892)
Returns whether the nib file containing the receiver's window has been loaded.

`window` (page 1895)
Returns the window owned by the receiver or `null` if there isn't one.

`setWindow` (page 1893)
Sets the window controller's window to *`aWindow`*.

`windowDidLoad` (page 1895)
Allows subclasses of NSWindowController to perform any required tasks after the window owned by the receiver has been loaded.

`windowWillLoad` (page 1897)
Allows subclasses of NSWindowController to perform any required tasks before the window owned by the receiver is loaded.

Setting and Getting the Document

`setDocument` (page 1892)
Sets the document associated with the window managed by the receiver.

`document` (page 1891)
Returns the NSDocument object associated with the receiver or `null` if there is none.

`setDocumentEdited` (page 1893)
Sets the document edited flag for the window controller to *`flag`*.

Closing the Window

`close` (page 1891)
Closes the window if it was loaded.

`shouldCloseDocument` (page 1894)
Returns whether the receiver closes the associated document when the window it manages is closed (`true`) or whether the document is closed only when the last remaining window of the document is closed (`false`).

`setShouldCloseDocument` (page 1893)
Sets whether the receiver should close the associated document when the window it manages is closed (*`flag`* is `true`) or whether to close the document only when the last document window has been closed (*`flag`* is `false`).

Getting Nib File Information

`owner` (page 1892)
Returns the owner of the nib file containing the window managed by the receiver.

`windowNibName` (page 1896)
Returns the name of the nib file that stores the window associated with the receiver.

`windowNibPath` (page 1896)
Returns the full path of the nib file that stores the window associated with the receiver.

Setting and Getting Window Attributes

`setShouldCascadeWindows` (page 1893)
Sets whether the window, when it is displayed, should cascade in relation to other document windows (that is, have a slightly offset location so that the title bars of previously displayed windows are still visible) to *`flag`*.

`shouldCascadeWindows` (page 1894)
Returns whether the window will cascade in relation to other document windows when it is displayed.

`setWindowFrameAutosaveName` (page 1894)
Sets the name under which the window's frame (its size and location on the screen) is saved in the defaults database.

`synchronizeWindowTitleWithDocumentName` (page 1895)
Synchronizes the displayed window title and the represented filename with the information in the associated document.

`windowFrameAutosaveName` (page 1896)
Returns the name under which the frame rectangle of the window owned by the receiver is stored in the defaults database.

`windowTitleForDocumentDisplayName` (page 1896)
Returns *displayName* by default.

Constructors

NSWindowController

Creates an NSWindowController object with no window object to manage.

```
public NSWindowController()
```

Discussion
The default initialization turns on cascading, sets the `shouldCloseDocument` (page 1894) flag to `false`, and sets the window frame autosave name to an empty string.

Creates an NSWindowController object initialized with *window*, the window object to manage.

```
public NSWindowController(NSWindow window)
```

Discussion
The *window* argument can be `null`. The default initialization turns on cascading, sets the `shouldCloseDocument` (page 1894) flag to `false`, and sets the window frame autosave name to an empty string. As a side effect, the created window controller is added as an observer of the `WindowWillCloseNotification` (page 1885)s posted by that window object (which is handled by a private method). If you make the window controller a delegate of the window, you can implement NSWindow's `windowShouldClose` (page 1880) delegate method.

Creates an NSWindowController object initialized with *windowNibName*, the name of the nib file (minus the "`.nib`" extension) that archives the receiver's window.

```
public NSWindowController(String windowNibName)
```

Discussion
The *windowNibName* argument cannot be `null`. The default initialization turns on cascading, sets the `shouldCloseDocument` (page 1894) flag to `false`, and sets the autosave name for the window's frame to an empty string.

Creates an NSWindowController object initialized with *windowNibName* and *owner*.

```
public NSWindowController(String windowNibName, Object owner)
```

Discussion

Neither *windowNibName* nor *owner* can be `null`. The *windowNibName* argument is the name of the nib file (minus the “`.nib`” extension) that archives the receiver’s window. The *owner* argument is the nib file’s owner. The default initialization turns on cascading, sets the `shouldCloseDocument` (page 1894) flag to `false`, and sets the autosave name for the window’s frame to an empty string.

Returns an NSWindowController object initialized with *windowNibNameOrPath* and *owner*. Neither *windowNibName* nor *owner* can be `null`

```
public NSWindowController(String windowNibNameOrPath, Object owner, boolean
    isFullPath)
```

Discussion

If *isFullPath* is `true`, *windowNibNameOrPath* should be the full path to the nib file that archives the receiver’s window. Use this option if your nib file is at a fixed location (which is not inside either the file’s owner’s class’s bundle or in the application’s main bundle). If *isFullPath* is `false`, *windowNibNameOrPath* is the name of the nib file (minus the “`.nib`” extension) that archives the receiver’s window.

The *owner* argument is the nib file’s owner. The default initialization turns on cascading, sets the `shouldCloseDocument` (page 1894) flag to `false`, and sets the autosave name for the window’s frame to an empty string.

Instance Methods

close

Closes the window if it was loaded.

```
public void close()
```

Discussion

Because this method closes the window without asking the user for confirmation, you usually do not invoke it when the Close menu command is chosen. Instead invoke NSWindow’s `performClose` (page 1846) on the receiver’s window. See “Window Closing Behavior” for an overview of deallocation behavior when a window is closed.

See Also

`shouldCloseDocument` (page 1894)

`setShouldCloseDocument` (page 1893)

document

Returns the NSDocument object associated with the receiver or `null` if there is none.

```
public NSDocument document()
```

Discussion

When a window controller is added to an NSDocument’s list of window controllers, the document sets the window controller’s document with `setDocument`. The Application Kit uses this outlet to access the document for relevant next-responder messages.

See Also

`setDocument` (page 1892)

isWindowLoaded

Returns whether the nib file containing the receiver's window has been loaded.

```
public boolean isWindowLoaded()
```

See Also

`loadWindow` (page 1892)

`window` (page 1895)

`windowDidLoad` (page 1895)

`windowWillLoad` (page 1897)

loadWindow

Loads the receiver's window from the nib file.

```
public void loadWindow()
```

Discussion

You should never directly invoke this method. Instead, invoke `window` (page 1895) so the `windowDidLoad` (page 1895) and `windowWillLoad` (page 1897) methods are invoked. Subclasses can override this method if the way it finds and loads the window is not adequate. It uses NSBundle's `bundleForClass` method to get the bundle, using the class of the nib file owner as argument. It then locates the nib file within the bundle and, if successful, loads it; if unsuccessful, it tries to find the nib file in the main bundle.

See Also

`isWindowLoaded` (page 1892)

owner

Returns the owner of the nib file containing the window managed by the receiver.

```
public Object owner()
```

Discussion

This owner is usually `this`, but can be the receiver's document (an instance of an NSDocument subclass) or some other object.

See Also

`windowNibName` (page 1896)

setDocument

Sets the document associated with the window managed by the receiver.

```
public void setDocument(NSDocument document)
```

Discussion

document is an instance of an NSDocument subclass that represents and manages the data displayed and captured in the window. Documents automatically call this method when they add a window controller to their list of window controllers; if you are using a subclass of NSDocument, you should not call it directly.

See Also

`document` (page 1891)

setDocumentEdited

Sets the document edited flag for the window controller to *flag*.

```
public void setDocumentEdited(boolean flag)
```

Discussion

The window controller uses this flag to control whether its associated window shows up as dirty. You should not call this method directly for window controllers with an associated NSDocument; NSDocument calls this method on its window controllers as needed.

setShouldCascadeWindows

Sets whether the window, when it is displayed, should cascade in relation to other document windows (that is, have a slightly offset location so that the title bars of previously displayed windows are still visible) to *flag*.

```
public void setShouldCascadeWindows(boolean flag)
```

Discussion

The default is `true`.

See Also

`shouldCascadeWindows` (page 1894)

setShouldCloseDocument

Sets whether the receiver should close the associated document when the window it manages is closed (*flag* is `true`) or whether to close the document only when the last document window has been closed (*flag* is `false`).

```
public void setShouldCloseDocument(boolean flag)
```

Discussion

The default is `false`.

See Also

`shouldCloseDocument` (page 1894)

setWindow

Sets the window controller's window to *aWindow*.

```
public void setWindow(NSWindow aWindow)
```

Discussion
This method releases the old window and any associated top-level objects in its nib file and assumes ownership of the new window. You should generally create a new window controller for a new window and release the old window controller instead of using this method.

setWindowFrameAutosaveName

Sets the name under which the window's frame (its size and location on the screen) is saved in the defaults database.

```
public void setWindowFrameAutosaveName(String name)
```

Discussion
By default, *name* is an empty string, causing no information to be stored in the defaults database.

See Also
windowFrameAutosaveName (page 1896)
setFrameAutosaveName (page 1860) (NSWindow)

shouldCascadeWindows

Returns whether the window will cascade in relation to other document windows when it is displayed.

```
public boolean shouldCascadeWindows()
```

Discussion
The default is true.

See Also
setShouldCascadeWindows (page 1893)

shouldCloseDocument

Returns whether the receiver closes the associated document when the window it manages is closed (true) or whether the document is closed only when the last remaining window of the document is closed (false).

```
public boolean shouldCloseDocument()
```

Discussion
The default is false.

See Also
setShouldCloseDocument (page 1893)

showWindow

This action method displays the window associated with the receiver.

```
public void showWindow(Object sender)
```

Discussion

If the window is an NSPanel object and has its `becomesKeyOnlyIfNeeded` (page 1053) flag set to `true`, the window is displayed in front of all other windows but is not made key; otherwise it is displayed in front and is made key. This method is useful for menu actions.

See Also

`makeKeyAndOrderFront` (page 1841) (NSWindow)

`orderFront` (page 1844) (NSWindow)

synchronizeWindowTitleWithDocumentName

Synchronizes the displayed window title and the represented filename with the information in the associated document.

```
public void synchronizeWindowTitleWithDocumentName()
```

Discussion

Does nothing if the window controller has no associated document or loaded window. This method queries the window controller's document to get the document's display name and full filename path, then calls `windowTitleForDocumentDisplayName` (page 1896) to get the display name to show in the window title.

window

Returns the window owned by the receiver or `null` if there isn't one.

```
public NSWindow window()
```

Discussion

If the window has not yet been loaded, it attempts to load the window's nib file using `loadWindow` (page 1892). Before it loads the window, it invokes `windowWillLoad` (page 1897) in subclass implementations, and if the NSWindowController has a document, it invokes the NSDocument's corresponding method windowControllerWillLoadNib (if implemented). After loading the window, it invokes `windowDidLoad` (page 1895) and, if there is a document, the NSDocument method windowControllerDidLoadNib (if implemented).

See Also

`windowControllerWillLoadNib` (page 550) (NSDocument)

windowDidLoad

Allows subclasses of NSWindowController to perform any required tasks after the window owned by the receiver has been loaded.

```
public void windowDidLoad()
```

Discussion

The default implementation does nothing.

See Also

`loadWindow` (page 1892)

`window` (page 1895)

`windowWillLoad` (page 1897)

windowFrameAutosaveName

Returns the name under which the frame rectangle of the window owned by the receiver is stored in the defaults database.

```
public String windowFrameAutosaveName()
```

See Also
`setWindowFrameAutosaveName` (page 1894)

windowNibName

Returns the name of the nib file that stores the window associated with the receiver.

```
public String windowNibName()
```

Discussion
If the nib path was passed to the constructor, `windowNibName` returns the last path component with the "`.nib`" extension stripped off. If the nib name was passed, `windowNibName` returns the name without the "`.nib`" extension.

See Also
`owner` (page 1892)

windowNibPath

Returns the full path of the nib file that stores the window associated with the receiver.

```
public String windowNibPath()
```

Discussion
If the nib path was passed to the constructor, the path is just returned. If the nib name was passed, `windowNibPath` locates the nib in the file's owner's class' bundle or in the application's main bundle and returns the full path (or `null` if it cannot be located). Subclasses can override this to augment the search behavior, but probably ought to call `super` first.

windowTitleForDocumentDisplayName

Returns *displayName* by default.

```
public String windowTitleForDocumentDisplayName(String displayName)
```

Discussion
The display name, which is generally maintained by the associated NSDocument, is the last path component under which the document file is saved. Subclasses can override this method to customize the window title. For example, a CAD application could append "-Top" or "-Side," depending on the view displayed by the window.

See Also
`synchronizeWindowTitleWithDocumentName` (page 1895)

windowWillLoad

Allows subclasses of NSWindowController to perform any required tasks before the window owned by the receiver is loaded.

```
public void windowWillLoad()
```

Discussion
The default implementation does nothing.

See Also
`loadWindow` (page 1892)
`window` (page 1895)
`windowDidLoad` (page 1895)

NSWorkspace

Inherits from	NSObject
Package:	com.apple.cocoa.application
Companion guide	Workspace Services Programming Topics

Overview

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

There is one shared NSWorkspace object per application. You use the static method `sharedWorkspace` (page 1902) to access it.

Tasks

Constructors

`NSWorkspace` (page 1902)
Creates an empty NSWorkspace.

Accessing the Shared NSWorkspace

`sharedWorkspace` (page 1902)
Returns the shared NSWorkspace instance.

Accessing the NSWorkspace Notification Center

`notificationCenter` (page 1908)
Returns the notification center for workspace notifications.

Opening Files

`openFile` (page 1908)
: Opens the file specified by *`fullPath`* using the *`appName`* application.

`openTempFile` (page 1909)
: Opens the temporary file specified by *`fullPath`* using the default application for its type.

`openURL` (page 1909)
: Opens the location specified by *`url`*; returns `true` if the location was successfully opened, `false` otherwise.

Manipulating Applications

`launchApplication` (page 1905)
: Launches the application *`appName`*.

`hideOtherApplications` (page 1904)
: Hides all applications other than the sender.

Manipulating Files

`applicationForFile` (page 1903)
: Returns the full path to the application that the system would use to open the document *`pathToFile`*.

`performFileOperation` (page 1910)
: Performs a file operation on a set of files in a particular directory.

`selectFile` (page 1910)
: Selects the file specified by *`fullPath`*.

Requesting Information

`iconForFile` (page 1904)
: Returns an NSImage with the icon for the single file specified by *`fullPath`*, with an initial size of 32 pixels by 32 pixels.

`iconForFileType` (page 1905)
: Returns an NSImage with the icon for the file type specified by *`fileType`*, with an initial size of 32 pixels by 32 pixels.

`iconForFiles` (page 1905)
: Returns an NSImage with the icon for the files specified in *`fullPaths`*, an array of Strings.

`fullPathForApplication` (page 1904)
: Returns the full path for the application *`appName`*, or `null` if *`appName`* isn't in one of the normal places.

`isFilePackageAtPath` (page 1905)
: Determines whether *`fullPath`* is a file package.

`activeApplication` (page 1903)
: Returns a dictionary with information about the current active application.

`launchedApplications` (page 1906)
Returns an array of dictionaries, one entry for each running application.

Requesting Additional Time Before Logout

`extendPowerOffBy` (page 1904)
Requests *`requested`* milliseconds more time before the power goes off or the user logs out.

Tracking Changes to the File System

`noteFileSystemChanged` (page 1907)
Informs NSWorkspace that the file system has changed.

`noteFileSystemChangedAtPath` (page 1907)
Informs NSWorkspace that the file system specified by *`path`* has changed.

`fileSystemChanged` (page 1904)
Returns `true` if a change to the file system has been registered with a `noteFileSystemChanged` (page 1907) message since the last `fileSystemChanged` (page 1904) message; `false` otherwise.

Updating Registered Services and File Types

`findApplications` (page 1904)
Examines all applications in the normal places (`/Network/Applications`, `/Applications`, `/Developer/Applications`) and updates the records of registered services and file types.

Tracking Changes to the Defaults Database

`noteUserDefaultsChanged` (page 1908)
Informs NSWorkspace that the defaults database has changed.

`userDefaultsChanged` (page 1911)
Returns whether a change to the defaults database has been registered with a `noteUserDefaultsChanged` (page 1908) message since the last `userDefaultsChanged` (page 1911) message.

Tracking Status Changes for Applications and Devices

`mountedRemovableMedia` (page 1906)
Returns an NSArray of Strings containing the full pathnames of all currently mounted removable disks.

`mountNewRemovableMedia` (page 1907)
Polls 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 an NSArray of Strings containing full pathnames to all newly mounted disks.

`mountedLocalVolumePaths` (page 1906)
: Returns an array containing the mount points of all local volumes, not just the removable ones returned by `mountedRemovableMedia` (page 1906).

`checkForRemovableMedia` (page 1903)
: Polls the system's drives for any disks that have been inserted but not yet mounted.

Unmounting a Device

`unmountAndEjectDeviceAtPath` (page 1910)
: Unmounts and ejects the device at *`path`*.

Working with Bundles

`absolutePathForAppBundleWithIdentifier` (page 1902)
: Returns the absolute file-system path of an application bundle.

`openURLs` (page 1909)
: Opens one or more files from an array of URLs.

Constructors

NSWorkspace

Creates an empty NSWorkspace.

```
public NSWorkspace()
```

Static Methods

sharedWorkspace

Returns the shared NSWorkspace instance.

```
public static NSWorkspace sharedWorkspace()
```

Instance Methods

absolutePathForAppBundleWithIdentifier

Returns the absolute file-system path of an application bundle.

```
public String absolutePathForAppBundleWithIdentifier(String bundleIdentifier)
```

Discussion
The *`bundleIdentifier`* parameter identifies the desired application and corresponds to the value from the `CFBundleIdentifier` key in the application's `Info.plist` file. For example, the bundle identifier of the TextEdit application is `com.apple.TextEdit`.

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

activeApplication

Returns a dictionary with information about the current active application.

```
public NSDictionary activeApplication()
```

Discussion
The dictionary contains as many of the keys described in the constants section as are available.

Availability
Available in Mac OS X v10.2 and later.

See Also
`launchedApplications` (page 1906)

applicationForFile

Returns the full path to the application that the system would use to open the document *`pathToFile`*.

```
public String applicationForFile(String pathToFile)
```

Discussion
Returns `null` if the file cannot be found or if the file is of an unknown type.

checkForRemovableMedia

Polls the system's drives for any disks that have been inserted but not yet mounted.

```
public void checkForRemovableMedia()
```

Discussion
`checkForRemovableMedia` (page 1903) doesn't wait until such disks are mounted; instead, it requests that the disk be mounted asynchronously and returns immediately. Currently has no effect.

See Also
`mountNewRemovableMedia` (page 1907)
`mountedRemovableMedia` (page 1906)

extendPowerOffBy

Requests *requested* milliseconds more time before the power goes off or the user logs out.

```
public int extendPowerOffBy(int requested)
```

Discussion
Returns the number of additional milliseconds granted. Currently unimplemented.

fileSystemChanged

Returns `true` if a change to the file system has been registered with a `noteFileSystemChanged` (page 1907) message since the last `fileSystemChanged` (page 1904) message; `false` otherwise.

```
public boolean fileSystemChanged()
```

Discussion
Currently always returns `false`.

findApplications

Examines all applications in the normal places (`/Network/Applications`, `/Applications`, `/Developer/Applications`) and updates the records of registered services and file types.

```
public void findApplications()
```

fullPathForApplication

Returns the full path for the application *appName*, or `null` if *appName* isn't in one of the normal places.

```
public String fullPathForApplication(String appName)
```

hideOtherApplications

Hides all applications other than the sender.

```
public void hideOtherApplications()
```

Discussion
The user can hide all applications except the current one by Command-Option-clicking on an application's Dock icon.

iconForFile

Returns an NSImage with the icon for the single file specified by *fullPath*, with an initial size of 32 pixels by 32 pixels.

```
public NSImage iconForFile(String fullPath)
```

See Also

`iconForFileType` (page 1905)

`iconForFiles` (page 1905)

iconForFiles

Returns an NSImage with the icon for the files specified in *fullPaths*, an array of Strings.

```
public NSImage iconForFiles(NSArray fullPaths)
```

Discussion

If *fullPaths* specifies one file, its icon is returned. If *fullPaths* specifies more than one file, an icon representing the multiple selection is returned.

See Also

`iconForFile` (page 1904)

`iconForFileType` (page 1905)

iconForFileType

Returns an NSImage with the icon for the file type specified by *fileType*, with an initial size of 32 pixels by 32 pixels.

```
public NSImage iconForFileType(String fileType)
```

Discussion

fileType may be either a filename extension or an encoded HFS file type.

See Also

`iconForFile` (page 1904)

`iconForFiles` (page 1905)

isFilePackageAtPath

Determines whether *fullPath* is a file package.

```
public boolean isFilePackageAtPath(String fullPath)
```

Discussion

Returns `false` if *fullPath* does not exist or is not a directory.

launchApplication

Launches the application *appName*.

```
public boolean launchApplication(String appName, boolean showIcon, boolean
    autoLaunch)
```

Discussion

If *showIcon* is `false`, the application's icon won't be placed on the screen. (The icon still exists, though.) If *autolaunch* is `true`, the autolaunch default will be set as though the application were autolaunched at startup. This method is provided to enable daemon-like applications that lack a normal user interface. Its use is not generally encouraged.

Returns `true` if the application is successfully launched or already running, and `false` if it can't be launched.

Before this method begins, it posts an `WorkspaceWillLaunchApplicationNotification` (page 1915) to the NSWorkspace's notification center. When the operation is complete, it posts an `WorkspaceDidLaunchApplicationNotification` (page 1913).

Launches the application *appName*.

```
public boolean launchApplication(String appName)
```

Discussion

appName need not be specified with a full path and, in the case of an application wrapper, may be specified with or without the `.app` extension, as described in "Use of .app Extension". Returns `true` if the application is successfully launched or already running, `false` if it can't be launched.

Before this method begins, it posts an `WorkspaceWillLaunchApplicationNotification` (page 1915) to the NSWorkspace's notification center. When the operation is complete, it posts an `WorkspaceDidLaunchApplicationNotification` (page 1913).

launchedApplications

Returns an array of dictionaries, one entry for each running application.

```
public NSArray launchedApplications()
```

Discussion

The dictionary contains as many of the keys described in the constants section as are available.

Availability

Available in Mac OS X v10.2 and later.

See Also

`activeApplication` (page 1903)

mountedLocalVolumePaths

Returns an array containing the mount points of all local volumes, not just the removable ones returned by `mountedRemovableMedia` (page 1906).

```
public NSArray mountedLocalVolumePaths()
```

mountedRemovableMedia

Returns an NSArray of Strings containing the full pathnames of all currently mounted removable disks.

```
public NSArray mountedRemovableMedia()
```

Discussion
If the computer provides an interrupt or other notification when the user inserts a disk into a drive, the Finder will mount the disk immediately. However, if no notification is given, the Finder won't be aware that a disk needs to be mounted. On such systems, an application should invoke either `mountNewRemovableMedia` (page 1907) or `checkForRemovableMedia` (page 1903) before invoking `mountedRemovableMedia` (page 1906). Either of these methods cause the Finder to poll the drives to see if a disk is present. If a disk has been inserted but not yet mounted, these methods will cause the Finder to mount it.

The Disk button in an Open or Save panel invokes `mountedRemovableMedia` (page 1906) and `mountNewRemovableMedia` (page 1907) as part of its operation, so most applications won't need to invoke these methods directly.

See Also
`checkForRemovableMedia` (page 1903)
`mountNewRemovableMedia` (page 1907)

mountNewRemovableMedia

Polls 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 an NSArray of Strings containing full pathnames to all newly mounted disks.

```
public NSArray mountNewRemovableMedia()
```

Discussion
This method posts a `WorkspaceDidMountNotification` (page 1913) to the NSWorkspace's notification center when it is finished. Currently provides the same functionality as `mountedRemovableMedia` (page 1906).

See Also
`checkForRemovableMedia` (page 1903)
`mountedRemovableMedia` (page 1906)

noteFileSystemChanged

Informs NSWorkspace that the file system has changed.

```
public void noteFileSystemChanged()
```

Discussion
NSWorkspace then gets the status of all the files and directories it is interested in and updates itself appropriately. This method is used by many objects that write or delete files.

NSDocument and NSSavePanel use this method when saving a file. If you create a file directly, you should call `noteFileSystemChanged` (page 1907) so that the Finder can update the folder if it is open.

See Also
`fileSystemChanged` (page 1904)

noteFileSystemChangedAtPath

Informs NSWorkspace that the file system specified by *`path`* has changed.

```
public void noteFileSystemChangedAtPath(String path)
```

Discussion
NSWorkspace then gets the status of all the files and directories it is interested in and updates itself appropriately. This method is used by many objects that write or delete files.

See Also
`fileSystemChanged` (page 1904)

noteUserDefaultsChanged

Informs NSWorkspace that the defaults database has changed.

```
public void noteUserDefaultsChanged()
```

Discussion
NSWorkspace then reads all the defaults it is interested in and reconfigures itself appropriately. For example, this method is used by the Preferences application to notify the Finder whether the user prefers to see hidden files. Currently has no effect.

See Also
`userDefaultsChanged` (page 1911)

notificationCenter

Returns the notification center for workspace notifications.

```
public NSNotificationCenter notificationCenter()
```

openFile

Opens the file specified by *fullPath* using the *appName* application.

```
public boolean openFile(String fullPath, String appName, boolean flag)
```

Discussion
appName need not be specified with a full path and, in the case of an application wrapper, may be specified with or without the `.app` extension, as described in "Use of .app Extension". If *appName* is `null`, the default application for the file's type is used. If *flag* is `true`, the sending application is deactivated before the request is sent, allowing the opening application to become the active application. Returns `true` if the file is successfully opened, `false` otherwise.

Opens the file specified by *fullPath* using the *appName* application.

```
public boolean openFile(String fullPath, String appName)
```

Discussion
appName need not be specified with a full path and, in the case of an application wrapper, may be specified with or without the `.app` extension, as described in "Use of .app Extension". The sending application is deactivated before the request is sent. Returns `true` if the file is successfully opened, `false` otherwise.

Opens the file specified by *fullPath* using the default application for its type.

```
public boolean openFile(String fullPath, NSImage anImage, NSPoint point, NSView
    aView)
```

Discussion

The Finder provides animation before opening the file to give the user feedback that the file is to be opened. To provide this animation, *anImage* should contain an icon for the file, and its image should be displayed at *point*, specified in the coordinates of *aView*.

The sending application is deactivated before the request is sent. Returns `true` if the file is successfully opened, `false` otherwise.

Opens the file specified by *fullPath* using the default application for its type; returns `true` if the file was successfully opened, `false` otherwise.

```
public boolean openFile(String fullPath)
```

Discussion

The sending application is deactivated before the request is sent.

openTempFile

Opens the temporary file specified by *fullPath* using the default application for its type.

```
public boolean openTempFile(String fullPath)
```

Discussion

The sending application is deactivated before the request is sent. Using this method instead of `openFile` (page 1908) lets the receiving application know that it should delete the file when it no longer needs it. Returns `true` if the file is successfully opened, `false` otherwise. Currently provides the same functionality as `openFile` (page 1908).

See Also

`openFile` (page 1908)

openURL

Opens the location specified by *url*; returns `true` if the location was successfully opened, `false` otherwise.

```
public boolean openURL(java.net.URL url)
```

openURLs

Opens one or more files from an array of URLs.

```
public boolean openURLs(NSArray urls, String bundleIdentifier, int options,
    NSAppleEventDescriptor descriptor, NSMutableArray launchIdentifiers)
```

Discussion

The *urls* parameter contains an array of NSURL objects, each identifying one URL to open. The *bundleIdentifier* parameter contains the bundle identifier of the application to use to open the URLs, or NULL to use the default system bindings. Possible values for the *options* parameter are described in the constants section. To specify additional options using an AppleEvent-style descriptor, specify a value for the *additionalEventParamDescriptor* parameter.

If you specify a parameter for *launchIdentifier*, the method returns an array of unique identifiers (one for each URL) for this launch attempt. You can use these values to distinguish individual launch requests.

Availability

Available in Mac OS X v10.3 and later.

performFileOperation

Performs a file operation on a set of files in a particular directory.

```
public int performFileOperation(String operation, String source, String destination,
     NSArray files)
```

Discussion

operation is some file operation, such as compressing or moving files. *files* contains Strings specifying the names of the files to be manipulated. The filenames are given relative to the source directory. The list can contain both files and directories; all of them must be located directly within *source* (not in one of its subdirectories).

Some operations—such as moving, copying, and linking files—require a destination directory to be specified. If not, *destination* should be the empty string (" ").

The possible values for operation are described in the "Constants" section.

This method returns a negative integer if the operation fails, 0 if the operation is performed synchronously and succeeds, and a positive integer if the operation is performed asynchronously. The positive integer identifies the requested file operation. Before this method returns, it posts a WorkspaceDidPerformFileOperationNotification (page 1913) to NSWorkspace's notification center.

selectFile

Selects the file specified by *fullPath*.

```
public boolean selectFile(String fullPath, String rootFullPath)
```

Discussion

If a path is specified by *rootFullPath*, a new file viewer is opened. If *rootFullPath* is an empty string (" "), the file is selected in the main viewer. Returns true if the file is successfully selected, false otherwise.

unmountAndEjectDeviceAtPath

Unmounts and ejects the device at *path*.

```
public boolean unmountAndEjectDeviceAtPath(String path)
```

Discussion

Returns `true` if the unmount operation succeeded, `false` otherwise. When this method begins, it posts a `WorkspaceWillUnmountNotification` (page 1915) to NSWorkspace's notification center. When it is finished, it posts a `WorkspaceDidUnmountNotification` (page 1914).

userDefaultsChanged

Returns whether a change to the defaults database has been registered with a `noteUserDefaultsChanged` (page 1908) message since the last `userDefaultsChanged` (page 1911) message.

```
public boolean userDefaultsChanged()
```

Discussion

Currently always returns `false`.

Constants

These constants specify different types of files:

Constant	Description
`PlainFileType`	Plain (untyped) file
`DirectoryFileType`	Directory
`ApplicationFileType`	Cocoa application
`FilesystemFileType`	File-system mount point
`ShellCommandFileType`	Executable shell command

The constants specify different types of file operations. They're used by `performFileOperation` (page 1910).

Constant	Description
`MoveOperation`	Move file to destination.
`CopyOperation`	Copy file to destination.
`LinkOperation`	Create hard link to file in destination.
`CompressOperation`	Compress file. Currently unavailable.
`DecompressOperation`	Decompress file. Currently unavailable.
`EncryptOperation`	Encrypt file. Currently unavailable.
`DecryptOperation`	Decrypt file. Currently unavailable.
`DestroyOperation`	Destroy file.

Constant	Description
RecycleOperation	Move file to recycler.
DuplicateOperation	Duplicate file in source directory.

The following describes keys for an NSDictionary containing information about an application. This dictionary is returned by activeApplication (page 1903) and launchedApplications (page 1906), and is also provided in the *userInfo* of NSWorkspace notifications for application launch and termination.

Key	Value
"NSApplicationPath"	The full path to the application, as a string.
"NSApplicationName"	The application's name, as a string.
"NSApplicationBundleIdentifier"	The application’s bundle identifier., as a string.
"NSApplicationProcessIdentifier"	The application's process id, as an integer.
"NSApplicationProcessSerialNumberHigh"	The high long of the process serial number (PSN), as an integer.
"NSApplicationProcessSerialNumberLow"	The low long of the process serial number (PSN), as an integer.

The following table describes launch options you can pass to openURLs (page 1909).

Constant	Description
LaunchAndPrint	Print items instead of opening them.
LaunchInhibitingBackgroundOnly	Causes launch to fail if the target is background-only.
LaunchWithoutAddingToRecents	Do not add the application or documents to the Recents menu.
LaunchWithoutActivation	Launch the application but do not bring it into the foreground.
LaunchAsync	Launch the application and return the results asynchronously.
LaunchAllowingClassicStartup	Start up the Classic compatibility environment, if it is required by the application.
LaunchPreferringClassic	Force the application to launch in the Classic compatibility environment.
LaunchNewInstance	Create a new instance of the application, even if one is already running.
LaunchAndHide	Tell the application to hide itself as soon as it has finished launching.
LaunchAndHideOthers	Hide all applications except the newly launched one.

Constant	Description
LaunchDefault	Launch the application asynchronously and launch it in the Classic environment, if required.

The following table describes the NSWorkspaceIconCreationOptions values. These values are combined using the C bitwise OR operator.

Constant	Description
NSExcludeQuickDraw-ElementsIconCreationOption	Supress generation of the QuickDraw format icon representations that are used Mac OS X v10.0 through v10.4. Available in Mac OS X v10.4 and later.
NSExclude10_-4ElementsIconCreationOption	Supress generation of the new higher resolution icon representations that are supported in Mac OS X v10.4. Available in Mac OS X v10.4 and later.

Notifications

All NSWorkspace notifications are posted to NSWorkspace's own notification center, not the application's default notification center. Access this center using NSWorkspace's notificationCenter (page 1908) method.

WorkspaceDidLaunchApplicationNotification

Posted when a new application has started up.

The notification object is the shared NSWorkspace instance. The *userInfo* dictionary contains the keys and values described in the constants section.

WorkspaceDidMountNotification

Posted when a new device has been mounted.

The notification object is the shared NSWorkspace instance. The *userInfo* dictionary contains the following information:

Key	Value
"NSDevicePath"	The path where the device was mounted, as a string.

WorkspaceDidPerformFileOperationNotification

Posted when a file operation has been performed in the receiving application.

The notification object is the shared NSWorkspace instance. The *userInfo* dictionary contains the following information:

Key	Value
`"NSOperationNumber"`	An integer indicating the type of file operation completed.

WorkspaceDidTerminateApplicationNotification

Posted when an application finishes executing.

The notification object is the shared NSWorkspace instance. The `userInfo` dictionary contains the keys and values described in the constants section.

WorkspaceDidWakeNotification

Posted when the machine wakes from sleep.

The notification object is the shared NSWorkspace instance. The notification does not contain a `userInfo` dictionary.

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

WorkspaceDidUnmountNotification

Posted when the Finder has unmounted a device.

The notification object is the shared NSWorkspace instance. The `userInfo` dictionary contains the following information:

Key	Value
`"NSDevicePath"`	The path where the device was previously mounted, as a string.

WorkspaceSessionDidBecomeActiveNotification

Posted after a user session is switched in. This allows an application to reenable some processing when a switched out session gets switched back in, for example.

The notification object is the shared NSWorkspace instance. The notification does not contain a `userInfo` dictionary.

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

WorkspaceSessionDidResignActiveNotification

Posted before a user session is switched out. This allows an application to disable some processing when its user session is switched out, and reenable when that session gets switched back in, for example.

The notification object is the shared NSWorkspace instance. The notification does not contain a `userInfo` dictionary.

If an application is launched in an inactive session, WorkspaceSessionDidResignActiveNotification is sent after ApplicationWillFinishLaunchingNotification (page 141) and before sending ApplicationDidFinishLaunchingNotification (page 140).

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

WorkspaceWillLaunchApplicationNotification

Posted when the Finder is about to launch an application.

The notification object is the shared NSWorkspace instance. The *userInfo* dictionary contains the keys and values described in the constants section.

WorkspaceWillPowerOffNotification

Posted when the user has requested a logout or that the machine be powered off.

The notification object is the shared NSWorkspace instance. This notification does not contain a *userInfo* dictionary.

WorkspaceWillSleepNotification

Posted before the machine goes to sleep. An observer of this message can delay sleep for up to 30 seconds while handling this notification.

The notification object is the shared NSWorkspace instance. The notification does not contain a *userInfo* dictionary.

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

WorkspaceWillUnmountNotification

Posted when the Finder is about to unmount a device. This notification will not be delivered if a volume was forcibly and immediately made unavailable, such as when a FireWire drive is simply unplugged, because there is no chance to deliver it before the volume becomes unavailable.

The notification object is the shared NSWorkspace instance. The *userInfo* dictionary contains the following information:

Key	Value
"NSDevicePath"	The path where the device is mounted, as a string.

Interfaces

_NSObsoleteMenuItemProtocol

Implements	NSValidatedUserInterfaceItem NSCoding
Package:	com.apple.cocoa.application
Companion guide	Application Menu and Pop-up List Programming Topics for Cocoa

Overview

Warning: The NSMenuItem interface is being removed from the Application Kit; you must use the NSMenuItem class instead. This change does not affect binary compatibility between different versions of projects, but might cause failures in project builds. To adapt your projects to this change, alter all references to the interface to references to the class.

Refer to the NSMenuItem (page 1919) class description, which replaces this interface.

Tasks

Enabling a Menu Item

`setEnabled` (page 1927)
: Sets whether the receiver is enabled based on *`flag`*.

`isEnabled` (page 1924)
: Returns `true` if the receiver is enabled, `false` if not.

Setting the Target and Action

`setTarget` (page 1932)
: Sets the receiver’s target to *`anObject`*.

`target` (page 1933)
: Returns the receiver’s target.

`setAction` (page 1926)
: Sets the receiver’s action method to *`aSelector`*.

Setting the Title

`setTitle` (page 1932)
Sets the receiver's title to *aString*.

`title` (page 1933)
Returns the receiver's title.

`setAttributedTitle` (page 1927)
Specifies a custom string for a menu item.

`attributedTitle` (page 1923)
Returns the custom title string for a menu item.

Setting the Tag

`setTag` (page 1931)
Sets the receiver's tag to *anInt*.

Setting the State

`setState` (page 1931)
Sets the state of the receiver to *itemState*, which should be one of `NSCell.OffState`, `NSCell.OnState`, or `NSCell.MixedState`.

`state` (page 1933)
Returns the state of the receiver, which is `NSCell.OffState` (the default), `NSCell.OnState`, or `NSCell.MixedState`.

Setting the Image

`setImage` (page 1928)
Sets the receiver's image to *menuImage*.

`image` (page 1923)
Returns the image displayed by the receiver, or `null` if it displays no image.

`setOnStateImage` (page 1930)
Sets the image of the receiver that indicates an "on" state.

`onStateImage` (page 1926)
Returns the image used to depict the receiver's "on" state, or `null` if the image has not been set.

`setOffStateImage` (page 1930)
Sets the image of the receiver that indicates an "off" state.

`offStateImage` (page 1925)
Returns the image used to depict the receiver's "off" state, or `null` if the image has not been set.

`setMixedStateImage` (page 1929)
Sets the image of the receiver that indicates a "mixed" state, that is, a state neither "on" nor "off."

`mixedStateImage` (page 1925)
Returns the image used to depict a "mixed state."

Managing Submenus

`setSubmenu` (page 1931)
    Sets the submenu of the receiver to *aSubmenu*.

`submenu` (page 1933)
    Returns the submenu associated with the receiving menu item, or `null` if no submenu is associated with it.

`hasSubmenu` (page 1923)
    Returns `true` if the receiver has a submenu, `false` if it doesn't.

Getting a Separator Item

`isSeparatorItem` (page 1924)
    Returns whether the receiver is a separator item (that is, a menu item used to visually segregate related menu items).

Setting the Owning Menu

`setMenu` (page 1929)
    Sets the receiver's menu to *aMenu*.

`menu` (page 1925)
    Returns the menu to which the receiver belongs, or `null` if no menu has been set.

Managing Key Equivalents

`setKeyEquivalent` (page 1928)
    Sets the receiver's unmodified key equivalent to *aKeyEquivalent*.

`keyEquivalent` (page 1924)
    Returns the receiver's unmodified keyboard equivalent, or the empty string if one hasn't been defined.

`setKeyEquivalentModifierMask` (page 1928)
    Sets the receiver's keyboard equivalent modifiers (indicating modifiers such as the Shift or Option key) to those in *mask*.

`keyEquivalentModifierMask` (page 1924)
    Returns the receiver's keyboard equivalent modifier mask.

Managing Mnemonics

`setMnemonicLocation` (page 1930)
    Sets the character of the menu item title at *location* that is to be underlined

`mnemonicLocation` (page 1925)
    Returns the position of the underlined character in the menu item title used as a mnemonic.

`setTitleWithMnemonic` (page 1932)
    Sets the title of a menu item with a character underlined to denote an access key.

`mnemonic` (page 1925)
Returns the character in the menu item title that appears underlined for use as a mnemonic.

Managing User Key Equivalents

`userKeyEquivalent` (page 1934)
Returns the user-assigned key equivalent for the receiver.

`userKeyEquivalentModifierMask` (page 1934)
Returns the modifier mask for the receiver's user-assigned key equivalent.

Managing Alternates

`setAlternate` (page 1926)
Marks the receiver as an alternate to the previous menu item.

`isAlternate` (page 1924)
Returns whether the receiver is an alternate to the previous menu item.

Managing Indentation Levels

`setIndentationLevel` (page 1928)
Sets the menu item indentation level for the receiver.

`indentationLevel` (page 1923)
Returns the menu item indentation level for the receiver.

Managing Tool Tips

`setToolTip` (page 1932)
Sets a help tag for a menu item.

`toolTip` (page 1933)
Returns the help tag for a menu item.

Representing an Object

`setRepresentedObject` (page 1930)
Sets the object represented by the receiver to *anObject*.

`representedObject` (page 1926)
Returns the object that the receiving menu item represents.

Instance Methods

attributedTitle

Returns the custom title string for a menu item.

```
public abstract NSAttributedString attributedTitle()
```

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

See Also
`setAttributedTitle` (page 1927)
`title` (page 1933)

hasSubmenu

Returns `true` if the receiver has a submenu, `false` if it doesn't.

```
public abstract boolean hasSubmenu()
```

See Also
`setSubmenuForItem` (page 922) (NSMenu)

image

Returns the image displayed by the receiver, or `null` if it displays no image.

```
public abstract NSImage image()
```

See Also
`setImage` (page 1928)

indentationLevel

Returns the menu item indentation level for the receiver.

```
public abstract int indentationLevel()
```

Discussion
The return value will be from 0 to 15. The default indentation level is 0.

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

See Also
`setIndentationLevel` (page 1928)

isAlternate

Returns whether the receiver is an alternate to the previous menu item.

```
public abstract boolean isAlternate()
```

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

See Also
`setAlternate` (page 1926)

isEnabled

Returns `true` if the receiver is enabled, `false` if not.

```
public abstract boolean isEnabled()
```

See Also
`setEnabled` (page 1927)

isSeparatorItem

Returns whether the receiver is a separator item (that is, a menu item used to visually segregate related menu items).

```
public abstract boolean isSeparatorItem()
```

keyEquivalent

Returns the receiver's unmodified keyboard equivalent, or the empty string if one hasn't been defined.

```
public abstract String keyEquivalent()
```

Discussion
Use `keyEquivalentModifierMask` (page 1924) to determine the modifier mask for the key equivalent.

See Also
`userKeyEquivalent` (page 1934)
`mnemonic` (page 1925)
`setKeyEquivalent` (page 1928)

keyEquivalentModifierMask

Returns the receiver's keyboard equivalent modifier mask.

```
public abstract int keyEquivalentModifierMask()
```

See Also
`setKeyEquivalentModifierMask` (page 1928)

menu

Returns the menu to which the receiver belongs, or `null` if no menu has been set.

```
public abstract NSMenu menu()
```

See Also
`setMenu` (page 1929)

mixedStateImage

Returns the image used to depict a "mixed state."

```
public abstract NSImage mixedStateImage()
```

Discussion
A mixed state is useful for indicating "off" and "on" attribute values in a group of selected objects, such as a selection of text containing bold and plain (nonbolded) words.

See Also
`setMixedStateImage` (page 1929)

mnemonic

Returns the character in the menu item title that appears underlined for use as a mnemonic.

```
public abstract String mnemonic()
```

Discussion
If there is no mnemonic character, returns an empty string. Mnemonics are not supported in Mac OS X.

See Also
`setTitleWithMnemonic` (page 1932)

mnemonicLocation

Returns the position of the underlined character in the menu item title used as a mnemonic.

```
public abstract int mnemonicLocation()
```

Discussion
The position is the zero-based index of that character in the title string. If the receiver has no mnemonic character, returns `NSArray.NotFound`. Mnemonics are not supported in Mac OS X.

See Also
`setMnemonicLocation` (page 1930)

offStateImage

Returns the image used to depict the receiver's "off" state, or `null` if the image has not been set.

```
public abstract NSImage offStateImage()
```

Discussion
By default, there is no off state image.

See Also
`setOffStateImage` (page 1930)

onStateImage

Returns the image used to depict the receiver's "on" state, or `null` if the image has not been set.

```
public abstract NSImage onStateImage()
```

Discussion
By default, the on state image is a checkmark.

See Also
`setOnStateImage` (page 1930)

representedObject

Returns the object that the receiving menu item represents.

```
public abstract Object representedObject()
```

Discussion
For example, you might have a menu list the names of views that are swapped into the same panel. The represented objects would be the appropriate NSView objects. The user would then be able to switch back and forth between the different views that are displayed by selecting the various menu items.

See Also
`setRepresentedObject` (page 1930)

setAction

Sets the receiver's action method to *aSelector*.

```
public abstract void setAction(NSSelector aSelector)
```

Discussion
See *Action Messages* for additional information on action messages.

See Also
`setTarget` (page 1932)

setAlternate

Marks the receiver as an alternate to the previous menu item.

```
public abstract void setAlternate(boolean isAlternate)
```

Discussion

If the receiver has the same key equivalent as the previous item, but has different key equivalent modifiers, the items are folded into a single visible item and the appropriate item shows while tracking the menu. The menu items may also have no key equivalent as long as the key equivalent modifiers are different.

If there are two or more items with no key equivalent but different modifiers, then the only way to get access to the alternate items is with the mouse. If you mark items as alternates but their key equivalents don't match, they might be displayed as separate items. Marking the first item as an alternate has no effect.

The *isAlternate* value is archived.

Availability

Available in Mac OS X v10.3 and later.

See Also

isAlternate (page 1924)

setAttributedTitle

Specifies a custom string for a menu item.

```
public abstract void setAttributedTitle(NSAttributedString string)
```

Discussion

You can use this method to add styled text and embedded images to menu item strings. If you do not set a text color for the attributed string, it is black when not selected, white when selected, and gray when disabled. Colored text remains unchanged when selected.

When you call this method to set the menu title to an attributed string, the setTitle (page 1932) method is also called to set the menu title with a plain string. If you clear the attributed title, the plain title remains unchanged.

The attributed string is not archived in the old nib format.

Availability

Available in Mac OS X v10.3 and later.

See Also

attributedTitle (page 1923)

setTitle (page 1932)

setEnabled

Sets whether the receiver is enabled based on *flag*.

```
public abstract void setEnabled(boolean flag)
```

Discussion

If a menu item is disabled, its keyboard equivalent is also disabled. See the NSMenu.MenuValidation (page 2011) interface specification for cautions regarding this method.

See Also

isEnabled (page 1924)

setImage

Sets the receiver's image to *menuImage*.

```
public abstract void setImage(NSImage menuImage)
```

Discussion

If *menuImage* is `null`, the current image (if any) is removed. This image is not affected by changes in menu-item state.

See Also

`image` (page 1923)

setIndentationLevel

Sets the menu item indentation level for the receiver.

```
public abstract void setIndentationLevel(int indentationLevel)
```

Discussion

The value for *indentationLevel* may be from 0 to 15. If *indentationLevel* is greater than 15, the value is pinned to the maximum. If *indentationLevel* is less than 0 an exception is thrown. The default indentation level is 0.

indentationLevel is archived.

Availability

Available in Mac OS X v10.3 and later.

See Also

`indentationLevel` (page 1923)

setKeyEquivalent

Sets the receiver's unmodified key equivalent to *aKeyEquivalent*.

```
public abstract void setKeyEquivalent(String aKeyEquivalent)
```

Discussion

If you want to remove the key equivalent from a menu item, pass an empty string ("") for *aKeyEquivalent* (never pass `null`). Use `setKeyEquivalentModifierMask` (page 1928) to set the appropriate mask for the modifier keys for the key equivalent.

See Also

`setMnemonicLocation` (page 1930)

`keyEquivalent` (page 1924)

setKeyEquivalentModifierMask

Sets the receiver's keyboard equivalent modifiers (indicating modifiers such as the Shift or Option key) to those in *mask*.

```
public abstract void setKeyEquivalentModifierMask(int mask)
```

Discussion
mask is an integer bit field containing any of these modifier key masks, combined using the C bitwise OR operator:

```
NSEvent.ShiftKeyMask
NSEvent.AlternateKeyMask
NSEvent.CommandKeyMask
```

You should always set `NSEvent.CommandKeyMask` in *mask*.

`NSEvent.ShiftKeyMask` is relevant only for function keys—that is, for key events whose modifier flags include `NSEvent.FunctionKeyMask`. For all other key events `NSEvent.ShiftKeyMask` is ignored and characters typed while the Shift key is pressed are interpreted as the shifted versions of those characters; for example, Command-Shift-c is interpreted as Command-C.

See the NSEvent (page 603) class specification for more information about modifier mask values.

See Also
`keyEquivalentModifierMask` (page 1924)

setMenu

Sets the receiver's menu to *aMenu*.

```
public abstract void setMenu(NSMenu aMenu)
```

Discussion
This method is invoked by the owning NSMenu when the receiver is added or removed. You shouldn't have to invoke this method in your own code, although it can be overridden to provide specialized behavior.

See Also
`menu` (page 1925)

setMixedStateImage

Sets the image of the receiver that indicates a "mixed" state, that is, a state neither "on" nor "off."

```
public abstract void setMixedStateImage(NSImage itemImage)
```

Discussion
If *itemImage* is `null`, any current mixed-state image is removed.

See Also
`mixedStateImage` (page 1925)
`setOffStateImage` (page 1930)
`setOnStateImage` (page 1930)
`setState` (page 1931)

setMnemonicLocation

Sets the character of the menu item title at *location* that is to be underlined

```
public abstract void setMnemonicLocation(int location)
```

Discussion

. This character identifies the access key by which users can access the menu item. Mnemonics are not supported in Mac OS X.

See Also

`mnemonicLocation` (page 1925)

setOffStateImage

Sets the image of the receiver that indicates an "off" state.

```
public abstract void setOffStateImage(NSImage itemImage)
```

Discussion

If *itemImage* is `null`, any current off-state image is removed.

See Also

`offStateImage` (page 1925)

`setMixedStateImage` (page 1929)

`setOffStateImage` (page 1930)

`setState` (page 1931)

setOnStateImage

Sets the image of the receiver that indicates an "on" state.

```
public abstract void setOnStateImage(NSImage itemImage)
```

Discussion

If *itemImage* is `null`, any current on-state image is removed.

See Also

`onStateImage` (page 1926)

`setMixedStateImage` (page 1929)

`setOffStateImage` (page 1930)

`setState` (page 1931)

setRepresentedObject

Sets the object represented by the receiver to *anObject*.

```
public abstract void setRepresentedObject(Object anObject)
```

Discussion

By setting a represented object for a menu item, you make an association between the menu item and that object. The represented object functions as a more specific form of tag that allows you to associate any object, not just an int, with the items in a menu.

For example, an NSView object might be associated with a menu item—when the user chooses the menu item, the represented object is fetched and displayed in a panel. Several menu items might control the display of multiple views in the same panel.

See Also

`setTag` (page 1931)

`representedObject` (page 1926)

setState

Sets the state of the receiver to *`itemState`*, which should be one of `NSCell.OffState`, `NSCell.OnState`, or `NSCell.MixedState`.

```
public abstract void setState(int itemState)
```

Discussion

The image associated with the new state is displayed to the left of the menu item.

See Also

`state` (page 1933)

`setMixedStateImage` (page 1929)

`setOffStateImage` (page 1930)

`setOnStateImage` (page 1930)

setSubmenu

Sets the submenu of the receiver to *`aSubmenu`*.

```
public abstract void setSubmenu(NSMenu aSubmenu)
```

Discussion

The default implementation throws an exception if *`aSubmenu`* already has a supermenu.

See Also

`submenu` (page 1933)

`hasSubmenu` (page 1923)

setTag

Sets the receiver's tag to *`anInt`*.

```
public abstract void setTag(int anInt)
```

See Also

`setRepresentedObject` (page 1930)

setTarget

Sets the receiver's target to *anObject*.

```
public abstract void setTarget(Object anObject)
```

See Also
setAction (page 1926)
target (page 1933)

setTitle

Sets the receiver's title to *aString*.

```
public abstract void setTitle(String aString)
```

See Also
title (page 1933)

setTitleWithMnemonic

Sets the title of a menu item with a character underlined to denote an access key.

```
public abstract void setTitleWithMnemonic(String aString)
```

Discussion
Mnemonics are not supported in Mac OS X.

See Also
mnemonic (page 1925)
setMnemonicLocation (page 1930)

setToolTip

Sets a help tag for a menu item.

```
public abstract void setToolTip(String toolTip)
```

Discussion
You can call this method for any menu item, including items in the main menu bar.

This string is not archived in the old nib format.

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

See Also
toolTip (page 1933)

state

Returns the state of the receiver, which is NSCell.OffState (the default), NSCell.OnState, or NSCell.MixedState.

```
public abstract int state()
```

See Also

setState (page 1931)

submenu

Returns the submenu associated with the receiving menu item, or null if no submenu is associated with it.

```
public abstract NSMenu submenu()
```

Discussion

If the receiver responds true to hasSubmenu (page 1923), the submenu is returned.

See Also

hasSubmenu (page 1923)
setSubmenu (page 1931)

target

Returns the receiver's target.

```
public abstract Object target()
```

See Also

setTarget (page 1932)

title

Returns the receiver's title.

```
public abstract String title()
```

See Also

setTitle (page 1932)

toolTip

Returns the help tag for a menu item.

```
public abstract String toolTip()
```

Availability

Available in Mac OS X v10.3 and later.

See Also
setToolTip (page 1932)

userKeyEquivalent

Returns the user-assigned key equivalent for the receiver.

```
public abstract String userKeyEquivalent()
```

See Also
keyEquivalent (page 1924)

userKeyEquivalentModifierMask

Returns the modifier mask for the receiver's user-assigned key equivalent.

```
public abstract int userKeyEquivalentModifierMask()
```

NSCellForTextAttachment

Package: com.apple.cocoa.application

Companion guides Text System Overview
Text Attachment Programming Topics for Cocoa

Overview

The NSCellForTextAttachment interface declares the interface for objects that draw text attachment icons and handle mouse events on their icons. With the exceptions of `cellBaselineOffset` (page 1936), `setAttachment` (page 1938), and `attachment` (page 1936), all of these methods are implemented by the NSCell class and described in that class specification.

See the NSAttributedString and NSTextView (page 1609) class specifications for general information on text attachments.

Tasks

Drawing

`drawWithFrameInView` (page 1937)
Draws the receiver's image within *cellFrame* in *aView*, which should be the view currently focused.

`highlightWithFrameInView` (page 1937)
Draws the receiver's image—with highlighting if *flag* is `true`—within *cellFrame* in *aView*, which should be the focus view.

Cell Size and Position

`cellSize` (page 1937)
Returns the size of the attachment's icon.

`cellBaselineOffset` (page 1936)
Returns the position where the attachment cell's image should be drawn in text, relative to the current point established in the glyph layout.

`cellFrame` (page 1936)
Returns the frame of the cell as it would be drawn as the character at the given glyph *position*, and character index, *charIndex*, in *textContainer*.

Event Handling

wantsToTrackMouse (page 1939)
Returns true if the receiver will handle a mouse event occurring over its image (to support dragging, for example), false otherwise.

wantsToTrackMouseForEvent (page 1939)
Allows an attachment to specify what events it would want to track the mouse for.

trackMouse (page 1938)
Handles a mouse-down event on the receiver's image.

Setting the Attachment

setAttachment (page 1938)
Sets the text attachment object that owns the receiver to *anAttachment*.

attachment (page 1936)
Returns the text attachment object that owns the receiver.

Instance Methods

attachment

Returns the text attachment object that owns the receiver.

```
public abstract NSTextAttachment attachment()
```

See Also
setAttachment (page 1938)

cellBaselineOffset

Returns the position where the attachment cell's image should be drawn in text, relative to the current point established in the glyph layout.

```
public abstract NSPoint cellBaselineOffset()
```

Discussion
The image should be drawn so its lower-left corner lies on this point.

See Also
icon (page 636) (NSFileWrapper)

cellFrame

Returns the frame of the cell as it would be drawn as the character at the given glyph *position*, and character index, *charIndex*, in *textContainer*.

```
public abstract NSRect cellFrame(NSTextContainer textContainer, NSRect lineFrag,
    NSPoint position, int charIndex)
```

Discussion

The proposed line fragment is specified by *lineFrag*.

cellSize

Returns the size of the attachment's icon.

```
public abstract NSSize cellSize()
```

See Also

icon (page 636) (NSFileWrapper)

fileWrapper (page 1537) (NSTextAttachment)

drawWithFrameInView

Draws the receiver's image within *cellFrame* in *aView*, which should be the view currently focused.

```
public abstract void drawWithFrameInView(NSRect cellFrame, NSView aView)
```

See Also

drawWithFrameInView (page 310) (NSCell)

lockFocus (page 1759) (NSView)

Draws the receiver's image within *cellFrame* in *aView*, which is the view currently focused.

```
public abstract void drawWithFrameInView(NSRect cellFrame, NSView aView, int
    charIndex)
```

Discussion

charIndex is the index of the attachment character within the text.

Draws the receiver's image within *cellFrame* in *aView*, which is the view currently focused.

```
public abstract void drawWithFrameInView(NSRect cellFrame, NSView aView, int
    charIndex, NSLayoutManager layoutManager)
```

Discussion

charIndex is the index of the attachment character within the text. *layoutManager* is the layout manager for the text.

highlightWithFrameInView

Draws the receiver's image—with highlighting if *flag* is true—within *cellFrame* in *aView*, which should be the focus view.

```
public abstract void highlightWithFrameInView(boolean flag, NSRect cellFrame, NSView
     aView)
```

See Also

highlightWithFrameInView (page 312) (NSCell)

lockFocus (page 1759) (NSView)

setAttachment

Sets the text attachment object that owns the receiver to *anAttachment*.

```
public abstract void setAttachment(NSTextAttachment anAttachment)
```

See Also

attachment (page 1936)

setAttachmentCell (page 1537) (NSTextAttachment)

trackMouse

Handles a mouse-down event on the receiver's image.

```
public abstract boolean trackMouse(NSEvent theEvent, NSRect cellFrame, NSView
    aTextView, int charIndex, boolean flag)
```

Discussion

theEvent is the mouse-down event. *cellFrame* is the region of *aTextView* in which further mouse events should be tracked. *charIndex* is the position in the text at which this attachment appears. *aTextView* is the view that received the event. It's assumed to be an NSTextView and should be the focus view. If *flag* is `true`, the receiver tracks the mouse until a mouse-up event occurs; if *flag* is `false`, it stops tracking when a mouse-dragged event occurs outside of *cellFrame*. Returns `true` if the receiver successfully finished tracking the mouse (typically through a mouse-up event), `false` otherwise (such as when the cursor is dragged outside *cellFrame*).

NSTextAttachmentCell's implementation of this method calls upon the delegate of *aTextView* to handle the event. If *theEvent* is a mouse-up event for a double click, the text attachment cell sends the delegate a textViewDoubleClickedCell (page 1669) message and returns `true`. Otherwise, depending on whether the user clicks or drags the cell, it sends the delegate a textViewClickedCell (page 1667) or a textViewDraggedCell (page 1669) message and returns `true`. NSTextAttachmentCell's implementation returns `false` only if *flag* is `false` and the cursor is dragged outside of *cellFrame*. The delegate methods are invoked only if the delegate responds.

See Also

wantsToTrackMouse (page 1939)

trackMouse (page 336) (NSCell)

lockFocus (page 1759) (NSView)

Handles a mouse-down event on the receiver's image.

```
public abstract boolean trackMouse(NSEvent theEvent, NSRect cellFrame, NSView
    aTextView, boolean flag)
```

Discussion

theEvent is the mouse-down event. *cellFrame* is the region of *aTextView* in which further mouse events should be tracked. *aTextView* is the view that received the event. It's assumed to be an NSTextView and should be the focus view. If *flag* is `true`, the receiver tracks the mouse until a mouse-up event occurs; if *flag* is `false`, it stops tracking when a mouse-dragged event occurs outside of *cellFrame*. Returns `true` if the receiver successfully finished tracking the mouse (typically through a mouse-up event), `false` otherwise (such as when the cursor is dragged outside *cellFrame*).

NSTextAttachmentCell's implementation of this method calls upon the delegate of *aTextView* to handle the event. If *theEvent* is a mouse-up event for a double click, the text attachment cell sends the delegate a `textViewDoubleClickedCell` (page 1669) message and returns `true`. Otherwise, depending on whether the user clicks or drags the cell, it sends the delegate a `textViewClickedCell` (page 1667) or a `textViewDraggedCell` (page 1669) message and returns `true`. NSTextAttachmentCell's implementation returns `false` only if *flag* is `false` and the cursor is dragged outside of *cellFrame*. The delegate methods are invoked only if the delegate responds.

See Also

`wantsToTrackMouse` (page 1939)
`trackMouse` (page 336) (NSCell)
`lockFocus` (page 1759) (NSView)

wantsToTrackMouse

Returns `true` if the receiver will handle a mouse event occurring over its image (to support dragging, for example), `false` otherwise.

```
public abstract boolean wantsToTrackMouse()
```

Discussion

NSTextAttachmentCell's implementation of this method returns `true`. The NSView containing the cell should invoke this method before sending a `trackMouse` (page 1938) message.

For an attachment in an attributed string, if the attachment cell returns `false` its attachment character should be selected rather than the cell being asked to track the mouse. This results in the attachment icon behaving as any regular glyph in text.

wantsToTrackMouseForEvent

Allows an attachment to specify what events it would want to track the mouse for.

```
public abstract boolean wantsToTrackMouseForEvent(NSEvent theEvent, NSRect cellFrame,
      NSView controlView, int charIndex)
```

Discussion

theEvent is the event in question that occurred in *cellFrame* inside *controlView*. *charIndex* is the index of the attachment character within the text. If `wantsToTrackMouse` (page 1939) returns `true`, this method allows the attachment to decide whether it wishes to do so for particular events.

NSChangeSpelling

Package:	com.apple.cocoa.application
Companion guide	Spell Checking

Overview

This interface is implemented by objects in the responder chain that can correct a misspelled word. See the NSSpellChecker (page 1379) class description for more information.

Tasks

Changing Spellings

`changeSpelling` (page 1941)
Replaces the selected word in the receiver with a corrected version from the Spelling panel.

Instance Methods

changeSpelling

Replaces the selected word in the receiver with a corrected version from the Spelling panel.

```
public abstract void changeSpelling(Object sender)
```

Discussion
This message is sent by the NSSpellChecker to the object whose text is being checked. To get the corrected spelling, ask *sender* for the string value of its selected cell (visible to the user as the text field in the Spelling panel). This method should replace the selected portion of the text with the string that it gets from the NSSpellChecker.

NSColorPickingCustom

Package:	com.apple.cocoa.application
Companion guide	Color Programming Topics for Cocoa

Overview

Together with the NSColorPickingDefault interface, NSColorPickingCustom provides a way to add color pickers—custom user interfaces for color selection—to an application's NSColorPanel. The NSColorPickingDefault interface provides basic behavior for a color picker. The NSColorPicker class adopts the NSColorPickingDefault interface.

Tasks

Setting the Current Color

`setColor` (page 1944)
Adjusts the receiver to make *`color`* the currently selected color.

Getting the Mode

`currentMode` (page 1944)
Returns the receiver's current mode (or submode, if applicable).

`supportsMode` (page 1944)
Returns whether or not the receiver supports the specified picking *`mode`*.

Getting the View

`provideNewView` (page 1944)
Returns the view containing the receiver's user interface.

Instance Methods

currentMode

Returns the receiver's current mode (or submode, if applicable).

```
public abstract int currentMode()
```

Discussion

The returned value should be unique to your color picker. See this interface description's list of the unique values for the standard color pickers used by the Application Kit.

See Also

`supportsMode` (page 1944)

provideNewView

Returns the view containing the receiver's user interface.

```
public abstract NSView provideNewView(boolean initialRequest)
```

Discussion

This message is sent to the color picker whenever the color panel attempts to display it. This may be when the panel is first presented, when the user switches pickers, or when the picker is switched through an API. The argument *initialRequest* is `true` only when this method is first invoked for your color picker. If *initialRequest* is `true`, the method should perform any initialization required (such as lazily loading a nib file, initializing the view, or performing any other custom initialization required for your picker). The NSView returned by this method should be set to automatically resize both its width and height.

setColor

Adjusts the receiver to make *color* the currently selected color.

```
public abstract void setColor(NSColor color)
```

Discussion

This method is invoked on the current color picker each time NSColorPanel's `setColor` (page 391) method is invoked. If *color* is actually different from the color picker's color (as it would be if, for example, the user dragged a color into NSColorPanel's color well), this method could be used to update the color picker's color to reflect the change.

supportsMode

Returns whether or not the receiver supports the specified picking *mode*.

```
public abstract boolean supportsMode(int mode)
```

Discussion

This method is invoked when the NSColorPanel is first initialized: It is used to attempt to restore the user's previously selected mode. It is also invoked by NSColorPanel's `setMode` (page 392) method to find the color picker that supports a particular mode. See this interface description's list of the unique mode values for the standard color pickers used by the Application Kit.

See Also

`currentMode` (page 1944)

NSColorPickingDefault

Package:	com.apple.cocoa.application
Companion guide	Color Programming Topics for Cocoa

Overview

The NSColorPickingDefault interface, together with the NSColorPickingCustom interface, provides an interface for adding color pickers—custom user interfaces for color selection—to an application's NSColorPanel. The NSColorPickingDefault interface provides basic behavior for a color picker. The NSColorPickingCustom interface provides implementation-specific behavior.

Tasks

Setting the Mode

`setMode` (page 1949)
Sets the color picker's mode.

Using Color Lists

`attachColorList` (page 1948)
Tells the color picker to attach the given *colorList*, if it isn't already displaying the list.

`detachColorList` (page 1948)
Tells the color picker to detach the given *colorList*, unless the receiver isn't displaying the list.

Adding Button Images

`insertNewButtonImage` (page 1949)
Sets *newButtonImage* as the image of *buttonCell*.

`provideNewButtonImage` (page 1949)
Returns the image for the mode button the user uses to select this picker in the color panel, that is, the color picker's representation in the NSColorPanel's picker NSMatrix.

Showing Opacity Controls

`alphaControlAddedOrRemoved` (page 1948)

Responding to a Resized View

`viewSizeChanged` (page 1950)
Tells the color picker when the NSColorPanel's view size changes in a way that might affect the color picker.

Instance Methods

alphaControlAddedOrRemoved

```
public abstract void alphaControlAddedOrRemoved(Object sender)
```

Discussion
Sent by the *sender* color panel when the opacity controls have been hidden or displayed. Invoked automatically when the NSColorPanel's opacity slider is added or removed; you never invoke this method directly.

If the color picker has its own opacity controls, it should hide or display them, depending on whether the sender's `showsAlpha` (page 392) method returns `false` or `true`.

attachColorList

Tells the color picker to attach the given *colorList*, if it isn't already displaying the list.

```
public abstract void attachColorList(NSColorList colorList)
```

Discussion
You never invoke this method; it's invoked automatically by the NSColorPanel when its `attachColorList` (page 389) method is invoked. Because NSColorPanel's list mode manages NSColorLists, this method need only be implemented by a custom color picker that manages NSColorLists itself.

See Also
`detachColorList` (page 1948)

detachColorList

Tells the color picker to detach the given *colorList*, unless the receiver isn't displaying the list.

```
public abstract void detachColorList(NSColorList colorList)
```

Discussion

You never invoke this method; it's invoked automatically by the NSColorPanel when its `detachColorList` (page 390) method is invoked. Because NSColorPanel's list mode manages NSColorLists, this method need only be implemented by a custom color picker that manages NSColorLists itself.

See Also

`attachColorList` (page 1948)

insertNewButtonImage

Sets *newButtonImage* as the image of *buttonCell*.

```
public abstract void insertNewButtonImage(NSImage newButtonImage, NSButtonCell
    buttonCell)
```

Discussion

buttonCell is the NSButtonCell object that lets the user choose the picker from the color panel—the color picker's representation in the NSColorPanel's picker NSMatrix. This method should perform application-specific manipulation of the image before it's inserted and displayed by the button cell.

See Also

`provideNewButtonImage` (page 1949)

provideNewButtonImage

Returns the image for the mode button the user uses to select this picker in the color panel, that is, the color picker's representation in the NSColorPanel's picker NSMatrix.

```
public abstract NSImage provideNewButtonImage()
```

Discussion

(This image is the same one the color panel uses as an argument when sending the `insertNewButtonImage` (page 1949) message.)

setMode

Sets the color picker's mode.

```
public abstract void setMode(int mode)
```

Discussion

This method is invoked by NSColorPanel's `setMode` (page 392) method to ensure the color picker reflects the current mode. For example, invoke this method during color picker initialization to ensure that all color pickers are restored to the mode the user left them in the last time an NSColorPanel was used.

Most color pickers have only one mode and thus don't need to do any work in this method. An example of a color picker that uses this method is the slider picker, which can choose from one of several submodes depending on the value of *mode*. The available modes are described in "Choosing the Color Pickers in a Color Panel".

viewSizeChanged

Tells the color picker when the NSColorPanel's view size changes in a way that might affect the color picker.

```
public abstract void viewSizeChanged(Object sender)
```

Discussion

sender is the NSColorPanel that contains the color picker. Use this method to perform special preparation when resizing the color picker's view. Because this method is invoked only as appropriate, it's better to implement this method than to override the method `superviewSizeChanged` for the NSView in which the color picker's user interface is contained.

See Also

`provideNewView` (page 1944) (NSColorPickingCustom interface)

NSComboBox.DataSource

(informal protocol)

Package: com.apple.cocoa.application

Overview

The NSComboBox.DataSource interface declares the methods that an NSComboBox (page 411) uses to access the contents of its data source object. For more information, see “Providing Data for a Combo Box”.

Tasks

Returning Information About Combo Box Items

`comboBoxValueForItemAtIndex` (page 1952)

`numberOfItemsInComboBox` (page 1952)

Working with Entered Strings

`comboBoxCompletedString` (page 1951)

`comboBoxIndexOfItem` (page 1952)

Instance Methods

comboBoxCompletedString

```
public abstract String comboBoxCompletedString(NSComboBox aComboBox, String
    uncompletedString)
```

Discussion

An NSComboBox, *aComboBox*, uses this method to perform incremental—or “smart”—searches when the user types into the text field. Your implementation should return the first complete string that starts with *uncompletedString*.

As the user types in the text field, the receiver uses this method to search for items from the pop-up list that start with what the user has typed. The receiver adds the new text to the end of the field and selects the new text, so when the user types another character, it replaces the new text.

This method is optional. If you don't implement it, the receiver does not perform incremental searches.

comboBoxIndexOfItem

```
public abstract int comboBoxIndexOfItem(NSComboBox aComboBox, String aString)
```

Discussion

An NSComboBox, `aComboBox`, uses this method to synchronize the pop-up list's selected item with the text field's contents. Your implementation of this method should return the index for the item that matches `aString`, or `NSArray.NotFound` if no item matches. If `comboBoxCompletedString` (page 1951) is implemented, `aString` is the string returned by that method. Otherwise, `aString` is the text that the user has typed.

This method is optional. If you don't implement it, the receiver does not synchronize the pop-up list's selected item with the text field's contents.

comboBoxValueForItemAtIndex

```
public abstract Object comboBoxObjectValueForItemAtIndex(NSComboBox aComboBox, int
    index)
```

Discussion

Implement this method to return the object that corresponds to the item at `index` in `aComboBox`. Your data source must implement this method.

numberOfItemsInComboBox

```
public abstract int numberOfItemsInComboBox(NSComboBox aComboBox)
```

Discussion

Implement this method to return the number of items managed for `aComboBox` by your data source object. An NSComboBox uses this method to determine how many items it should display in its pop-up list. Your data source must implement this method.

NSComboBoxCell.DataSource

(informal protocol)

Package: com.apple.cocoa.application

Overview

The NSComboBoxCell.DataSource interface declares the methods that an NSComboBoxCell (page 427) uses to access the contents of its data source object. For more information, see “Providing Data for a Combo Box”.

Tasks

Returning Information About Combo Box Items

`comboBoxCellObjectValueForItemAtIndex` (page 1954)

`numberOfItemsInComboBoxCell` (page 1954)

Working with Entered Strings

`comboBoxCellCompletedString` (page 1953)

`comboBoxCellIndexOfItem` (page 1954)

Instance Methods

comboBoxCellCompletedString

```
public abstract String comboBoxCellCompletedString(NSComboBoxCell aComboBoxCell,
    String uncompletedString)
```

Discussion

An NSComboBoxCell uses this method to perform incremental—or “smart”—searches when the user types into the text field. Your implementation should return the first complete string that starts with *uncompletedString*.

As the user types in the text field, the receiver uses this method to search for items from the pop-up list that start with what the user has typed. The receiver adds the new text to the end of the field and selects the new text, so when the user types another character, it replaces the new text.

This method is optional. If you don't implement it, the receiver does not perform incremental searches.

comboBoxCellIndexOfItem

```
public abstract int comboBoxCellIndexOfItem(NSComboBoxCell aComboBoxCell, String
    aString)
```

Discussion

An NSComboBoxCell, *aComboBoxCell*, uses this method to synchronize the pop-up list's selected item with the text field's contents. Your implementation of this method should return the index for the item that matches *aString*, or `NSArray.NotFound` if no item matches. If `comboBoxCellCompletedString` (page 1953) is implemented, *aString* is the string returned by that method. Otherwise, *aString* is the text that the user has typed.

This method is optional. If you don't implement it, the receiver does not synchronize the pop-up list's selected item with the text field's contents.

comboBoxCellObjectValueForItemAtIndex

```
public abstract Object comboBoxCellObjectValueForItemAtIndex(NSComboBoxCell
    aComboBoxCell, int index)
```

Discussion

Implement this method to return the object that corresponds to the item at *index* in *aComboBoxCell*. Your data source must implement this method.

numberOfItemsInComboBoxCell

```
public abstract int numberOfItemsInComboBoxCell(NSComboBoxCell aComboBoxCell)
```

Discussion

Implement this method to return the number of items managed for *aComboBoxCell* by your data source object. An NSComboBoxCell uses this method to determine how many items it should display in its pop-up list. Your data source must implement this method.

NSDraggingDestination

(informal protocol)

Package: com.apple.cocoa.application

Companion guide Drag and Drop Programming Topics for Cocoa

Overview

The NSDraggingDestination interface declares methods that the destination object (or recipient) of a dragged image must implement. The destination automatically receives NSDraggingDestination messages for pasteboard data types it has registered for as an image enters, moves around inside, and then exits or is released within the destination's boundaries.

Tasks

Before the Image Is Released

`draggingEntered` (page 1956)

`draggingUpdated` (page 1957)

`draggingEnded` (page 1956)
Implement this method to be notified when a drag operation ends in some other destination.

`draggingExited` (page 1957)
Invoked when the dragged image exits the destination's bounds rectangle (in the case of a view object) or its frame rectangle (in the case of a window object).

`wantsPeriodicDraggingUpdates` (page 1958)
Requests the destination object whether it wants to receive periodic `draggingUpdated` (page 1957) messages.

After the Image Is Released

`prepareForDragOperation` (page 1958)

`performDragOperation` (page 1957)

`concludeDragOperation` (page 1956)

Instance Methods

concludeDragOperation

```
public abstract void concludeDragOperaton(NSDraggingInfo sender)
```

Discussion
Invoked when the dragging operation is complete and the previous `performDragOperation` (page 1957) returned `true`. The destination implements this method to perform any tidying up that it needs to do, such as updating its visual representation now that it has incorporated the dragged data. This message is the last message sent from *sender* to the destination during a dragging session.

draggingEnded

Implement this method to be notified when a drag operation ends in some other destination.

```
public abstract void draggingEnded(NSDraggingInfo sender)
```

Discussion
This method might be used by a destination doing autoexpansion in order to collapse any autoexpands. *sender* contains details about the dragging operation. This method has not yet been implemented.

draggingEntered

```
public abstract int draggingEntered(NSDraggingInfo sender)
```

Discussion
Invoked when a dragged image enters the destination but only if the destination has registered for the pasteboard data type involved in the drag operation. Specifically, this method is invoked when the mouse pointer enters the destination's bounds rectangle (if it is a view object) or its frame rectangle (if it is a window object).

This method must return a value that indicates which dragging operation the destination will perform when the image is released. In deciding which dragging operation to return, the method should evaluate the overlap between both the dragging operations allowed by the source (obtained from *sender* with the `draggingSourceOperationMask` (page 1961) method) and the dragging operations and pasteboard data types the destination itself supports. The returned value should be exactly one of the dragging operation constants described in NSDraggingInfo's "Constants" section.

If none of the operations is appropriate, this method should return `NSDraggingInfo.DragOperationNone` (this is the default response if the method is not implemented by the destination). A destination will still receive `draggingUpdated` (page 1957) and `draggingExited` (page 1957) even if `NSDraggingInfo.DragOperationNone` is returned by this method.

See Also
`draggingUpdated` (page 1957)

`draggingExited` (page 1957)
`prepareForDragOperation` (page 1958)

draggingExited

Invoked when the dragged image exits the destination's bounds rectangle (in the case of a view object) or its frame rectangle (in the case of a window object).

```
public abstract void draggingExited(NSDraggingInfo sender)
```

Discussion
sender contains details about the dragging operation.

draggingUpdated

```
public abstract int draggingUpdated(NSDraggingInfo sender)
```

Discussion
Invoked periodically as the image is held within the destination if the destination has registered for the pasteboard data type involved in the drag operation. The messages continue until the image is either released or dragged out of the window or view. The returned value should be exactly one of the dragging operation constants described in NSDraggingInfo's constants section. The default return value (if this method is not implemented by the destination) is the value returned by the previous `draggingEntered` (page 1956) message.

This method provides the destination with an opportunity to modify the dragging operation depending on the position of the mouse pointer inside of the destination view or window object. For example, you may have several graphics or areas of text contained within the same view and wish to tailor the dragging operation, or to ignore the drag event completely, depending upon which object is underneath the mouse pointer at the time when the user releases the dragged image and the `performDragOperation` (page 1957) method is invoked. *sender* contains details about the dragging operation.

You typically examine the contents of the pasteboard in the `draggingEntered` (page 1956) method, where this examination is performed only once, rather than in the `draggingUpdated` (page 1957) method, which is invoked multiple times.

Only one destination at a time receives a sequence of `draggingUpdated` (page 1957) messages. If the mouse pointer is within the bounds of two overlapping views that are both valid destinations, the uppermost view receives these messages until the image is either released or dragged out.

See Also
`draggingExited` (page 1957)
`prepareForDragOperation` (page 1958)

performDragOperation

```
public abstract boolean performDragOperation(NSDraggingInfo sender)
```

Discussion

Invoked after the released image has been removed from the screen and the previous `prepareForDragOperation` (page 1958) message has returned `true`. The destination should implement this method to do the real work of importing the pasteboard data represented by the image. If the destination accepts the data, it returns `true`; otherwise it returns `false`. The default is to return `false`. Use *`sender`* to obtain details about the dragging operation.

See Also

`concludeDragOperation` (page 1956)

prepareForDragOperation

```
public abstract boolean prepareForDragOperation(NSDraggingInfo sender)
```

Discussion

Invoked when the image is released, if the most recent `draggingEntered` (page 1956) or `draggingUpdated` (page 1957) message returned an acceptable drag-operation value. Returns `true` if the receiver agrees to perform the drag operation and `false` if not. Use *`sender`* to obtain details about the dragging operation.

See Also

`performDragOperation` (page 1957)

wantsPeriodicDraggingUpdates

Requests the destination object whether it wants to receive periodic `draggingUpdated` (page 1957) messages.

```
public abstract boolean wantsPeriodicDraggingUpdates()
```

Discussion

If the destination returns `NO`, these messages are sent only when the mouse moves or a modifier flag changes. Otherwise the destination gets the default behavior, where it receives periodic dragging-updated messages even if nothing changes.

Availability

Available in Mac OS X v10.4 and later.

NSDraggingInfo

Package:	com.apple.cocoa.application
Companion guide	Drag and Drop Programming Topics for Cocoa

Overview

The NSDraggingInfo interface declares methods that supply information about a dragging session. NSDraggingInfo methods are designed to be invoked from within a class's implementation of NSDraggingDestination interface methods. The Application Kit automatically passes an object that conforms to the NSDraggingInfo interface as the argument to each of the methods defined by NSDraggingDestination. NSDraggingInfo messages should be sent to this object; you never need to create a class that implements the NSDraggingInfo interface.

Tasks

Dragging-session Information

`draggingSource` (page 1961)
  Returns the source, or owner, of the dragged data or `null` if the source is not in the same application as the destination.

`draggingSourceOperationMask` (page 1961)
  Returns the dragging operation mask declared by the dragging source (through its `draggingSourceOperationMaskForLocal` (page 1966) method).

`draggingDestinationWindow` (page 1960)
  Returns the destination window for the dragging operation.

`draggingPasteboard` (page 1961)
  Returns the pasteboard object that holds the data being dragged.

`draggingSequenceNumber` (page 1961)
  Returns a number that uniquely identifies the dragging session.

`draggingLocation` (page 1961)
  Returns the current location of the mouse pointer in the base coordinate system of the destination object's window.

`namesOfPromisedFilesDroppedAtDestination` (page 1962)
  Sets the drop location for promised files to *`dropDestination`* and returns the names (not full paths) of the files that the receiver promises to create there.

Image Information

`draggedImage` (page 1960)
    Returns the image being dragged.

`draggedImageLocation` (page 1960)
    Returns the current location of the dragged image's origin in the base coordinate system of the destination object's window.

Sliding the Image

`slideDraggedImageTo` (page 1962)
    Slides the image to *`aPoint`*, a specified location in the screen coordinate system.

Instance Methods

draggedImage

Returns the image being dragged.

```
public abstract NSImage draggedImage()
```

Discussion
This image object visually represents the data put on the pasteboard during the drag operation; however, it is the pasteboard data and not this image that is ultimately utilized in the dragging operation.

See Also
`draggedImageLocation` (page 1960)

draggedImageLocation

Returns the current location of the dragged image's origin in the base coordinate system of the destination object's window.

```
public abstract NSPoint draggedImageLocation()
```

Discussion
The image moves along with the mouse pointer (the position of which is given by `draggingLocation` (page 1961)) but may be positioned at some offset.

See Also
`draggedImage` (page 1960)

draggingDestinationWindow

Returns the destination window for the dragging operation.

```
public abstract NSWindow draggingDestinationWindow()
```

Discussion

Either this window is the destination itself, or it contains the view object that is the destination.

draggingLocation

Returns the current location of the mouse pointer in the base coordinate system of the destination object's window.

```
public abstract NSPoint draggingLocation()
```

See Also

`draggedImageLocation` (page 1960)

draggingPasteboard

Returns the pasteboard object that holds the data being dragged.

```
public abstract NSPasteboard draggingPasteboard()
```

Discussion

The dragging operation that is ultimately performed utilizes this pasteboard data and not the image returned by the `draggedImage` (page 1960) method.

draggingSequenceNumber

Returns a number that uniquely identifies the dragging session.

```
public abstract int draggingSequenceNumber()
```

draggingSource

Returns the source, or owner, of the dragged data or `null` if the source is not in the same application as the destination.

```
public abstract Object draggingSource()
```

Discussion

The dragging source implements methods from the NSDraggingSource (page 1965) interface.

draggingSourceOperationMask

Returns the dragging operation mask declared by the dragging source (through its `draggingSourceOperationMaskForLocal` (page 1966) method).

```
public abstract int draggingSourceOperationMask()
```

Discussion

If the source permits dragging operations, the elements in the mask are one or more of the constants described in the "Constants" section, combined using the C bitwise OR operator.

If the source does not permit any dragging operations, this method should return `DragOperationNone`.

If the user is holding down a modifier key during the dragging session and the source does not prohibit modifier keys from affecting the drag operation (through its `ignoreModifierKeysWhileDragging` (page 1966) method), then the operating system combines the dragging operation value that corresponds to the modifier key (see the descriptions below) with the source's mask using the C bitwise AND operator.

The modifier keys are associated with the dragging operation options shown below:

Modifier Key	Dragging Operation
Control	`DragOperationLink`
Option	`DragOperationCopy`
Command	`DragOperationGeneric`

namesOfPromisedFilesDroppedAtDestination

Sets the drop location for promised files to *`dropDestination`* and returns the names (not full paths) of the files that the receiver promises to create there.

```
public abstract NSArray namesOfPromisedFilesDroppedAtDestination(java.net.URL
    dropDestination)
```

Discussion
Drag destinations should invoke this method within their `performDragOperation` (page 1957) method. The source may or may not have created the files by the time this method returns.

Availability
Available in Mac OS X v10.2 and later.

slideDraggedImageTo

Slides the image to *`aPoint`*, a specified location in the screen coordinate system.

```
public abstract void slideDraggedImageTo(NSPoint aPoint)
```

Discussion
This method can be used to snap the image down to a particular location. It should only be invoked from within the destination's implementation of `prepareForDragOperation` (page 1958)—in other words, after the user has released the image but before it is removed from the screen.

Constants

The following constants are defined by NSDraggingInfo and are used by `draggingSourceOperationMask` (page 1961):

Constant	Description
`DragOperationCopy`	The data represented by the image can be copied.
`DragOperationLink`	The data can be shared.
`DragOperationGeneric`	The operation can be defined by the destination.
`DragOperationPrivate`	The operation is negotiated privately between the source and the destination.
`DragOperationMove`	The data can be moved.
`DragOperationDelete`	The data can be deleted.
`DragOperationEvery`	All of the above.
`DragOperationAll`	Deprecated. Use `DragOperationEvery` instead.
`DragOperationNone`	No drag operations are allowed.

NSDraggingSource

(informal protocol)

Package: com.apple.cocoa.application

Companion guide Drag and Drop Programming Topics for Cocoa

Overview

The NSDraggingSource interface declares methods that are implemented by the source object in a dragging session. The dragging source is specified as an argument to the `dragImage` (page 1751) message, sent to a window or view object to initiate the dragging session.

Of the methods declared below, only `draggingSourceOperationMaskForLocal` (page 1966) must be implemented. The other methods are invoked only if the dragging source implements them. All methods are invoked automatically during a dragging session—you never send an NSDraggingSource message directly to an object.

Tasks

Specifying Dragging Options

`draggingSourceOperationMaskForLocal` (page 1966)

`ignoreModifierKeysWhileDragging` (page 1966)
Sets whether the use of modifier keys should have an effect on the type of operation performed.

Responding to Dragging Sessions

`startedDraggingImage` (page 1967)

`finishedDraggingImage` (page 1966)
Similar to the previous version of this method, but includes an indication of the operation performed by the destination in *`operation`*, rather than a boolean indicating acceptance.

`movedDraggingImage` (page 1967)
Informs the dragging source about *`draggedImage`* moving to a new screen coordinate, *`screenPoint`*, similar to the dragging destination being sent `draggingUpdated` (page 1957) messages.

`namesOfPromisedFilesDroppedAtDestination` (page 1967)
Returns the names (not full paths) of the files that the receiver promises to create at *`dropDestination`*.

Instance Methods

draggingSourceOperationMaskForLocal

```
public abstract int draggingSourceOperationMaskForLocal(boolean isLocal)
```

Discussion

This method is the only NSDraggingSource method that must be implemented by the source object. It should return a mask, built by combining the allowed dragging operations listed in NSDraggingInfo's constants section, using the C bitwise OR operator. You should use this mask to indicate which types of dragging operations the source object will allow to be performed on the dragged image's data. A `true` value for *`isLocal`* indicates that the candidate destination object (the window or view over which the dragged image is currently poised) is in the same application as the source, while a `false` value indicates that the destination object is in a different application.

If the source does not permit any dragging operations, it should return `NSDraggingInfo.DragOperationNone`.

finishedDraggingImage

```
public abstract void finishedDraggingImage(NSImage anImage, NSPoint aPoint, boolean
     flag)
```

Discussion

Invoked after *`anImage`* has been released and the dragging destination has been given a chance to operate on the data it represents. *`aPoint`* is the location of the image's origin in the screen coordinate system when it was released. A `true` value for *`flag`* indicates that the destination accepted the dragged data, while a `false` value indicates that it was rejected.

This method provides the source object with an opportunity to respond to either a successful or a failed dragging session. For example, if you are moving data from one location to another, you could use this method to make the source data disappear from its previous location, if the dragging session is successful, or reset itself to its previous state, in the event of a failure.

Similar to the previous version of this method, but includes an indication of the operation performed by the destination in *`operation`*, rather than a boolean indicating acceptance.

```
public abstract void finishedDraggingImage(NSImage anImage, NSPoint aPoint, int
    operation)
```

See Also

`convertScreenToBase` (page 1824) (NSWindow)
`convertBaseToScreen` (page 1824) (NSWindow)
`convertPointFromView` (page 1744) (NSView)
`convertPointToView` (page 1745) (NSView)

ignoreModifierKeysWhileDragging

Sets whether the use of modifier keys should have an effect on the type of operation performed.

```
public abstract boolean ignoreModifierKeysWhileDragging()
```

Discussion
If this method is not implemented or returns `false`, the user can tailor the drag operation by holding down a modifier key during the drag. The dragging option that corresponds to the modifier key is combined with the source's mask (as set with the `draggingSourceOperationMaskForLocal` (page 1966) method) using the C bitwise AND operator. See the description for the `draggingSourceOperationMask` (page 1961) method in the NSDraggingInfo interface specification for more information about dragging masks and modifier keys.

movedDraggingImage

Informs the dragging source about *draggedImage* moving to a new screen coordinate, *screenPoint*, similar to the dragging destination being sent `draggingUpdated` (page 1957) messages.

```
public abstract void movedDraggingImage(NSImage draggedImage, NSPoint screenPoint)
```

namesOfPromisedFilesDroppedAtDestination

Returns the names (not full paths) of the files that the receiver promises to create at *dropDestination*.

```
public abstract NSArray namesOfPromisedFilesDroppedAtDestination(java.net.URL
    dropDestination)
```

Discussion
This method is invoked when the drop has been accepted by the destination and the destination, in the case of another Cocoa application, invokes the NSDraggingInfo method `namesOfPromisedFilesDroppedAtDestination` (page 1962). For long operations, you can cache *dropDestination* and defer the creation of the files until the `finishedDraggingImage` (page 1966) method to avoid blocking the destination application.

Availability
Available in Mac OS X v10.2 and later.

startedDraggingImage

```
public abstract void startedDraggingImage(NSImage anImage, NSPoint aPoint)
```

Discussion
Invoked when *anImage* is displayed but before it starts following the mouse. *aPoint* is the origin of the image in screen coordinates. This method provides the source object with an opportunity to respond to the initiation of a dragging session. For example, you might choose to have the source give a visual indication to the user that data is being dragged from the source.

See Also
`convertScreenToBase` (page 1824) (NSWindow)
`convertBaseToScreen` (page 1824) (NSWindow)
`convertPointFromView` (page 1744) (NSView)
`convertPointToView` (page 1745) (NSView)

NSEditor

(informal protocol)

Package:	com.apple.cocoa.application
Availability	Available in Mac OS X v10.3 and later.
Companion guide	Cocoa Bindings Programming Topics

Overview

The NSEditor interface provides a means for requesting that the receiver commit or discard any pending edits.

These methods are typically invoked on user interface elements by a controller. They can also be sent to a controller in response to a user's attempt to save a document or quit an application.

Tasks

Managing Editing

`commitEditing` (page 1969)
Returns whether the receiver was able to commit any pending edits.

`discardEditing` (page 1970)
Causes the receiver to discard any changes, restoring the previous values.

Instance Methods

commitEditing

Returns whether the receiver was able to commit any pending edits.

```
public abstract boolean commitEditing()
```

Discussion
Returns `true` if the changes were successfully applied to the model, `false` otherwise. A commit is denied if the receiver fails to apply the changes to the model object, perhaps due to a validation error.

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

See Also
`discardEditing` (page 1970)

discardEditing

Causes the receiver to discard any changes, restoring the previous values.

```
public abstract void discardEditing()
```

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

See Also
`commitEditing` (page 1969)

NSEditorRegistration

(informal protocol)

Package:	com.apple.cocoa.application
Availability	Available in Mac OS X v10.3 and later.
Companion guide	Cocoa Bindings Programming Topics

Overview

The NSEditorRegistration interface is implemented by controllers to provide an interface for a view, the editor, to inform the controller when it has uncommitted changes.

An implementor is responsible for tracking which editors have uncommitted changes, and sending those editors `commitEditing` (page 1969) and `discardEditing` (page 1970) messages, as appropriate, to force the editor to submit, or discard, their values.

Tasks

Managing Editing

`objectDidBeginEditing` (page 1971)
This message should be sent to the receiver when *editor* has uncommitted changes that can affect the receiver.

`objectDidEndEditing` (page 1972)
This message should be sent to the receiver when *editor* has finished editing a property belonging to the receiver.

Instance Methods

objectDidBeginEditing

This message should be sent to the receiver when *editor* has uncommitted changes that can affect the receiver.

```
public abstract void objectDidBeginEditing(Object editor)
```

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

See Also
`objectDidEndEditing` (page 1972)

objectDidEndEditing

This message should be sent to the receiver when *editor* has finished editing a property belonging to the receiver.

```
public abstract void objectDidEndEditing(Object editor)
```

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

See Also
`objectDidBeginEditing` (page 1971)

NSIgnoreMisspelledWords

Package:	com.apple.cocoa.application
Companion guide	Spell Checking

Overview

Implement this interface 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 interface must be implemented.

This interface 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, so usually several spelling checks are run in succession until no misspelled words are found. This interface allows the list of ignored words to be maintained per document, even though the spelling checks are not run per document.

The NSIgnoreMisspelledWords interface specifies a single method, `ignoreSpelling` (page 1974).

The second argument to the NSSpellChecker method `ignoreWord` (page 1383) is a tag that the NSSpellChecker can use to distinguish the documents being checked. 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 (page 1379) class description for more information.

Tasks

Ignoring Spellings

`ignoreSpelling` (page 1974)

Instance Methods

ignoreSpelling

```
public abstract void ignoreSpelling(Object sender)
```

Discussion

Implement this action method 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.

NSInputServerMouseTracker

Package:	com.apple.cocoa.application
Companion guide	Text Input Management

Overview

An "NSInputServiceProvider" (page 1977) object (an "NSInputServer" (page 809) subclass object or a delegate of an NSInputServer object) may need to implement this interface. See the "NSInputServiceProvider" (page 1977) interface description.

The methods in this interface differ from typical mouse events in that they have an additional argument which is the index of the character within the text view's text storage. When an text view object forwards a mouse event to the input manager , the input manager calls the text view's `characterIndexForPoint` (page 2026) method to get the index, which it then passes on to the appropriate method in this interface.

Tasks

Handling Mouse Events

`mouseDownOnCharacterIndex` (page 1975)
: A mouse down event happened at given *`index`* within the *`sender`* text view's text storage, at the given *`point`*, with modifier keys identified in *`flags`*.

`mouseDraggedOnCharacterIndex` (page 1976)
: A mouse dragged event happened at given *`index`* within the *`sender`* text view's text storage, at the given *`point`*, with modifier keys identified in *`flags`*.

`mouseUpOnCharacterIndex` (page 1976)
: A mouse up event happened at given *`index`* within the *`sender`* text view's text storage, at the given *`point`*, with modifier keys identified in *`flags`*.

Instance Methods

mouseDownOnCharacterIndex

A mouse down event happened at given *`index`* within the *`sender`* text view's text storage, at the given *`point`*, with modifier keys identified in *`flags`*.

```
public abstract boolean mouseDownOnCharacterIndex(int index, NSPoint point, int
    flags, Object sender)
```

Discussion

Returns `true` if it consumes the event; in that case, a mouse dragged or a mouse up message will follow. If `false` is returned, then neither of the other two events will follow.

mouseDraggedOnCharacterIndex

A mouse dragged event happened at given *index* within the *sender* text view's text storage, at the given *point*, with modifier keys identified in *flags*.

```
public abstract boolean mouseDraggedOnCharacterIndex(int index, NSPoint point, int
     flags, Object sender)
```

Discussion

Returns `true` if it consumes the event; in that case, either another mouse dragged or a mouse up message will follow. If `false` is returned, then neither message will follow.

mouseUpOnCharacterIndex

A mouse up event happened at given *index* within the *sender* text view's text storage, at the given *point*, with modifier keys identified in *flags*.

```
public abstract boolean mouseUpOnCharacterIndex(int index, NSPoint point, int flags,
     Object sender)
```

Discussion

This event is always consumed.

NSInputServiceProvider

Package: com.apple.cocoa.application

Companion guide Text Input Management

Overview

The NSInputServiceProvider interface embodies most of the functionality of NSInputServer (page 809).

There are two ways you might use this interface:

- You can subclass NSInputServer and create an instance of your subclass. Your subclass must override most or all of the NSInputServiceProvider interface methods.
- You can create an NSInputServer object and designate a delegate. The delegate must implement the NSInputServiceProvider interface.

All messages in this interface are sent by the client text view except `insertText` (page 1980) and `doCommandBySelector` (page 1979), which are sent by NSInputManager (page 801).

Tasks

Command Instance Methods Sent by Client

`activeConversationChanged` (page 1978)
Keyboard focus just switched from another text view to this one.

`activeConversationWillChange` (page 1978)
Keyboard focus is about to move away from this text view.

`canBeDisabled` (page 1979)
Returns `true` if the receiver can be disabled when the sender is not a text view, `false`

`inputClientBecomeActive` (page 1979)
The client, *`sender`*, has become active.

`inputClientDisabled` (page 1980)
A text view in the client, *`sender`*, has ceased to be the key-receiving first responder.

`inputClientEnabled` (page 1980)
A text view in the client, *`sender`*, has become the key-receiving first responder.

`inputClientResignActive` (page 1980)
The client, *`sender`*, is about to become inactive.

`markedTextAbandoned` (page 1981)
  Abandon any marked text state that may be in process.

`markedTextSelectionChanged` (page 1981)

`terminate` (page 1981)
  The client application is quitting.

Query Instance Methods Sent by Client

`wantsToDelayTextChangeNotifications` (page 1981)
  A `true` return value tells the client that only a call to its `insertText` (page 1980) method constitutes a modification to its text storage.

`wantsToHandleMouseEvents` (page 1982)
  Returns `true` if the client should forward all mouse events within the text view to the input server.

`wantsToInterpretAllKeystrokes` (page 1982)
  Returns `true` if the server wants all keystrokes to be sent to it as characters.

Instance Methods Sent by NSInputManager

`doCommandBySelector` (page 1979)
  Handle the command identified by *aSelector*.

`insertText` (page 1980)
  Interpret the characters in *aString*, which is actually always a `String`.

Instance Methods

activeConversationChanged

Keyboard focus just switched from another text view to this one.

```
public abstract void activeConversationChanged(Object sender, int newConversation)
```

Discussion
This is called only when switching within the same application. *sender* can be cast to NSTextInput.

See Also
`activeConversationWillChange` (page 1978)
`conversationIdentifier` (page 2027) (NSTextInput)

activeConversationWillChange

Keyboard focus is about to move away from this text view.

```
public abstract void activeConversationWillChange(Object sender, int oldConversation)
```

Discussion

This is called only when switching within the same application. *sender* can be cast to NSTextInput.

See Also

activeConversationChanged (page 1978)

conversationIdentifier (page 2027) (NSTextInput)

canBeDisabled

Returns true if the receiver can be disabled when the sender is not a text view, false

```
public abstract boolean canBeDisabled()
```

Discussion

otherwise.

doCommandBySelector

Handle the command identified by *aSelector*.

```
public abstract void doCommandBySelector(NSSelector aSelector, Object sender)
```

Discussion

The command can be from the set of NSResponder action methods or from the set of selector values in the DefaultKeyBindings dictionary referenced in the input server's "Info" file. *sender* can be cast to NSTextInput.

If you are subclassing NSInputServer (page 809), there is no need to override this method in the subclass. All you have to do is implement in the subclass the command methods you want to handle. If you do need to override this method, then you must call super for commands not handled.

If your NSInputServer (page 809) uses a delegate, the delegate's implementation of this method must call doCommandBySelector(sender, aSelector) for commands it does not handle.

See Also

doCommandBySelector (page 2027) (NSTextInput)

inputClientBecomeActive

The client, *sender*, has become active.

```
public abstract void inputClientBecomeActive(Object sender)
```

Discussion

This is called when the client application starts up and whenever it becomes active after being inactive. *sender* can be cast to NSTextInput.

See Also

inputClientEnabled (page 1980)

inputClientResignActive (page 1980)

inputClientDisabled

A text view in the client, *sender*, has ceased to be the key-receiving first responder.

```
public abstract void inputClientDisabled(Object sender)
```

Discussion

inputClientResignActive (page 1980) may also be called just after this is called. *sender* can be cast to NSTextInput.

See Also

inputClientEnabled (page 1980)

inputClientResignActive (page 1980)

inputClientEnabled

A text view in the client, *sender*, has become the key-receiving first responder.

```
public abstract void inputClientEnabled(Object sender)
```

Discussion

This is called the first time any text view becomes enabled after client application activation and again whenever focus switches to a text view. inputClientBecomeActive (page 1979) may have been called just before this is called. *sender* can be cast to NSTextInput.

See Also

inputClientBecomeActive (page 1979)

inputClientDisabled (page 1980)

inputClientResignActive

The client, *sender*, is about to become inactive.

```
public abstract void inputClientResignActive(Object sender)
```

Discussion

This is called when the client application quits and whenever it is deactivated. *sender* can be cast to NSTextInput.

See Also

inputClientBecomeActive (page 1979)

inputClientDisabled (page 1980)

terminate (page 1981)

insertText

Interpret the characters in *aString*, which is actually always a `String`.

```
public abstract void insertText(Object aString, Object sender)
```

Discussion
Here is where you do the interpreting of keyboard input. If your server's interpretation is disabled or the characters in `aString` are not of interest to the server, you can simply pass `aString` along to the sender's `insertText` (page 2028) method. `sender` can be cast to NSTextInput.

See Also
`insertText` (page 2028) (NSTextInput)

markedTextAbandoned

Abandon any marked text state that may be in process.

```
public abstract void markedTextAbandoned(Object sender)
```

Discussion
This can happen if the user clicks the mouse outside of the marked text area or if the window containing the text view closes. The client can do what it wants with the marked text. `NSTextView` leaves it as inserted text. `sender` can be cast to NSTextInput.

See Also
`markedTextSelectionChanged` (page 1981)

markedTextSelectionChanged

```
public abstract void markedTextSelectionChanged(NSRange newSelection, Object sender)
```

Discussion
The user selected a portion of the marked text or clicked at the beginning or end of marked text or somewhere in between. `sender` can be cast to NSTextInput.

See Also
`markedTextAbandoned` (page 1981)

terminate

The client application is quitting.

```
public abstract void terminate(Object sender)
```

Discussion
This is called after `inputClientResignActive` (page 1980). `sender` can be cast to NSTextInput.

See Also
`inputClientResignActive` (page 1980)

wantsToDelayTextChangeNotifications

A `true` return value tells the client that only a call to its `insertText` (page 1980) method constitutes a modification to its text storage.

```
public abstract boolean wantsToDelayTextChangeNotifications()
```

Discussion

A `false` return value tells the client that all text given to it, whether marked text or not, should constitute a modification to its text storage. A `true` return value tells the client that only unmarked text given to it should constitute a modification to its text storage. The client may for example want to filter all text that is part of a modification but leave marked text unfiltered.

See Also

`wantsToDelayTextChangeNotifications` (page 807) (NSInputManager)

wantsToHandleMouseEvents

Returns `true` if the client should forward all mouse events within the text view to the input server.

```
public abstract boolean wantsToHandleMouseEvents()
```

Discussion

If the server needs to implement the NSInputServerMouseTracker (page 1975) interface, return `true`.

See Also

`wantsToHandleMouseEvents` (page 808) (NSInputManager)

wantsToInterpretAllKeystrokes

Returns `true` if the server wants all keystrokes to be sent to it as characters.

```
public abstract boolean wantsToInterpretAllKeystrokes()
```

Discussion

If this method returns `false`, control key combinations and function keys (the arrow keys, PageDown, F5, and so on) are delivered to the input server via the key binding mechanism and `doCommandBySelector` (page 1979).

The Unicode values for the characters representing keyboard function keys (the arrow keys, PageDown, F5, and so on) names like `NSEvent.UpArrowFunctionKey`, and are documented in NSEvent (page 603). Control-key combinations are the usual ASCII control character codes.

For more information on key bindings, see “About Key Bindings”.

See Also

`wantsToInterpretAllKeystrokes` (page 808) (NSInputManager)

NSKeyBindingResponder

Package: com.apple.cocoa.application

Companion guide Cocoa Event-Handling Guide

Overview

This interface is implemented by objects in the responder chain that manipulate selections and edit text.

Tasks

Action Methods

`cancelOperation` (page 1987)
: Implemented by subclasses to cancel the current operation.

`capitalizeWord` (page 1987)
: This action method capitalizes the word or words surrounding the insertion point or selection, expanding the selection if necessary.

`centerSelectionInVisibleArea` (page 1988)
: This action method scrolls the selection, whatever it is, inside its visible area.

`changeCaseOfLetter` (page 1988)
: This action method changes the case of a letter or letters in the selection, perhaps by opening a panel with capitalization options or by cycling through possible case combinations.

`complete` (page 1988)
: This action method completes an operation in progress or a partially constructed element.

`deleteBackward` (page 1988)
: This action method deletes the selection, if there is one, or a single element backward from the insertion point (a letter or character in text, for example).

`deleteBackwardByDecomposingPreviousCharacter` (page 1989)

`deleteForward` (page 1989)
: This action method deletes the selection, if there is one, or a single element forward from the insertion point (a letter or character in text, for example).

`deleteToBeginningOfLine` (page 1989)
: This action method deletes the selection, if there is one, or all text from the insertion point to the beginning of a line (typically of text).

`deleteToBeginningOfParagraph` (page 1989)
: This action method deletes the selection, if there is one, or all text from the insertion point to the beginning of a paragraph of text.

`deleteToEndOfLine` (page 1989)
: This action method deletes the selection, if there is one, or all text from the insertion point to the end of a line (typically of text).

`deleteToEndOfParagraph` (page 1990)
: This action method deletes the selection, if there is one, or all text from the insertion point to the end of a paragraph of text.

`deleteToMark` (page 1990)
: This action method deletes the selection, if there is one, or all items from the insertion point to a previously placed mark, including the selection itself if not empty.

`deleteWordBackward` (page 1990)
: This action method deletes the selection, if there is one, or a single word backward from the insertion point.

`deleteWordForward` (page 1990)
: This action method deletes the selection, if there is one, or a single word forward from the insertion point.

`indent` (page 1991)
: This action method indents the selection or the insertion point if there is no selection.

`insertBacktab` (page 1991)
: This action method handles a “backward tab.”

`insertNewline` (page 1991)
: This action method inserts a line-break character at the insertion point or selection, deleting the selection if there is one, or to end editing if the receiver is a text field or other field editor.

`insertNewlineIgnoringFieldEditor` (page 1991)
: This action method inserts a line-break character at the insertion point or selection, deleting the selection if there is one.

`insertParagraphSeparator` (page 1992)
: This action method inserts a paragraph separator at the insertion point or selection, deleting the selection if there is one.

`insertTab` (page 1992)
: This action method inserts a tab character at the insertion point or selection, deleting the selection if there is one, or to end editing if the receiver is a text field or other field editor.

`insertTabIgnoringFieldEditor` (page 1992)
: This action method inserts a tab character at the insertion point or selection, deleting the selection if there is one.

`insertText` (page 1992)
: Inserts *`anObject`* at the insertion point or selection, deleting the selection if there is one.

`lowercaseWord` (page 1992)
: This action method lowercases every letter in the word or words surrounding the insertion point or selection, expanding the selection if necessary.

`moveBackward` (page 1992)
: This action method moves the selection or insertion point one element or character backward.

moveBackwardAndModifySelection (page 1993)
This action method expands or reduce either end of the selection backward by one element or character.

moveDown (page 1993)
This action method moves the selection or insertion point one element or character down.

moveDownAndModifySelection (page 1993)
This action method expands or reduces the top or bottom end of the selection downward by one element, character, or line (whichever is appropriate for text direction).

moveForward (page 1993)
This action method moves the selection or insertion point one element or character forward.

moveForwardAndModifySelection (page 1994)
This action method expands or reduces either end of the selection forward by one element or character.

moveLeft (page 1994)
This action method moves the selection or insertion point one element or character to the left.

moveLeftAndModifySelection (page 1994)
Implemented by subclasses to expand or reduce either end of the selection to the left (display order) by one element or character.

moveRight (page 1995)
This action method moves the selection or insertion point one element or character to the right.

moveRightAndModifySelection (page 1995)
Implemented by subclasses to expand or reduce either end of the selection to the right (display order) by one element or character.

moveToBeginningOfDocument (page 1995)
This action method moves the selection to the first element of the document, or the insertion point to the beginning.

moveToBeginningOfLine (page 1995)
This action method moves the selection to the first element of the selected line, or the insertion point to the beginning of the line.

moveToBeginningOfParagraph (page 1995)
This action method moves the insertion point to the beginning of the selected paragraph.

moveToEndOfDocument (page 1996)
This action method moves the selection to the last element of the document, or the insertion point to the end.

moveToEndOfLine (page 1996)
This action method moves the selection to the last element of the selected line, or the insertion point to the end of the line.

moveToEndOfParagraph (page 1996)
This action method moves the insertion point to the end of the selected paragraph.

moveUp (page 1996)
This action method moves the selection or insertion point one element or character up.

moveUpAndModifySelection (page 1996)
This action method expands or reduces the top or bottom end of the selection upward by one element, character, or line (whichever is appropriate for text direction).

moveWordBackward (page 1996)
This action method moves the selection or insertion point one word backward.

moveWordBackwardAndModifySelection (page 1997)
: This action method expands or reduces either end of the selection backward by one whole word.

moveWordForward (page 1997)
: This action method moves the selection or insertion point one word forward.

moveWordForwardAndModifySelection (page 1997)
: This action method expands or reduces either end of the selection forward by one whole word.

moveWordLeft (page 1998)
: Implemented by subclasses to expand or reduce either end of the selection left by one whole word in display order.

moveWordLeftAndModifySelection (page 1998)
: Implemented by subclasses to expand or reduce either end of the selection left by one whole word in display order.

moveWordRight (page 1998)
: Implemented by subclasses to move the selection or insertion point one word right.

moveWordRightAndModifySelection (page 1999)
: Implemented by subclasses to expand or reduce either end of the selection to the right by one whole word.

pageDown (page 1999)
: This action method scrolls the receiver down (or back) one page in its scroll view, also moving the insertion point to the top of the newly displayed page.

pageUp (page 1999)
: This action method scrolls the receiver up (or forward) one page in its scroll view, also moving the insertion point to the top of the newly-displayed page.

scrollLineDown (page 2000)
: This action method scrolls the receiver one line down in its scroll view, without changing the selection.

scrollLineUp (page 2000)
: This action method scrolls the receiver one line up in its scroll view, without changing the selection.

scrollPageDown (page 2000)
: This action method scrolls the receiver one page down in its scroll view, without changing the selection.

scrollPageUp (page 2000)
: This action method scrolls the receiver one page up in its scroll view, without changing the selection.

selectAll (page 2000)
: This action method selects all selectable elements.

selectLine (page 2001)
: This action method selects all elements in the line or lines containing the selection or insertion point.

selectParagraph (page 2001)
: This action method selects all paragraphs containing the selection or insertion point.

selectToMark (page 2001)
: This action method selects all items from the insertion point or selection to a previously placed mark, including the selection itself if not empty.

selectWord (page 2001)
: This action method extends the selection to the nearest word boundaries outside it (up to, but not including, word delimiters).

`setMark` (page 2001)
: This action method sets a mark at the insertion point or selection, which is used by `deleteToMark` (page 1990) and `selectToMark` (page 2001).

`swapWithMark` (page 2001)
: This action method swaps the mark and the selection or insertion point, so that what was marked is now the selection or insertion point, and what was the insertion point or selection is now the mark.

`transpose` (page 2002)
: This action method transposes the characters to either side of the insertion point and advances the insertion point past both of them. Does nothing to a selected range of text.

`transposeWords` (page 2002)
: This action method transposes the two words prior to the insertion point and advances the insertion point past both of them. Not currently implemented by NSTextView.

`uppercaseWord` (page 2002)
: This action method makes uppercase every letter in the word or words surrounding the insertion point or selection, expanding the selection if necessary.

`yank` (page 2002)
: This action method replaces the insertion point or selection with text from the kill buffer.

Dispatch Methods

`doCommandBySelector` (page 1990)
: Attempts to perform the method indicated by *aSelector*.

Instance Methods

cancelOperation

Implemented by subclasses to cancel the current operation.

```
public abstract void cancelOperation(Object sender)
```

Discussion

This method is bound to the Escape and Command-. (period) keys. The key window first propagates the key equivalent down its view hierarchy. If none of these views handles the key equivalent, the window sends a default action message of `cancel` to the first responder and from there the message travels up the responder chain. The *sender* argument is typically the object that invoked this method.

Availability

Available in Mac OS X v10.3 and later.

capitalizeWord

This action method capitalizes the word or words surrounding the insertion point or selection, expanding the selection if necessary.

```
public abstract void capitalizeWord(Object sender)
```

Discussion

If either end of the selection partially covers a word, that entire word is made lowercase.

See Also

`lowercaseWord` (page 1992)

`uppercaseWord` (page 2002)

`changeCaseOfLetter` (page 1988)

centerSelectionInVisibleArea

This action method scrolls the selection, whatever it is, inside its visible area.

```
public abstract void centerSelectionInVisibleArea(Object sender)
```

See Also

`scrollLineDown` (page 2000)

`scrollLineUp` (page 2000)

`scrollPageDown` (page 2000)

`scrollPageUp` (page 2000)

changeCaseOfLetter

This action method changes the case of a letter or letters in the selection, perhaps by opening a panel with capitalization options or by cycling through possible case combinations.

```
public abstract void changeCaseOfLetter(Object sender)
```

See Also

`lowercaseWord` (page 1992)

`uppercaseWord` (page 2002)

`capitalizeWord` (page 1987)

complete

This action method completes an operation in progress or a partially constructed element.

```
public abstract void complete(Object sender)
```

Discussion

This method can be interpreted, for example, as a request to attempt expansion of a partial word, such as for expanding a glossary shortcut, or to close a graphics item being drawn.

deleteBackward

This action method deletes the selection, if there is one, or a single element backward from the insertion point (a letter or character in text, for example).

```
public abstract void deleteBackward(Object sender)
```

deleteBackwardByDecomposingPreviousCharacter

```
public abstract void deleteBackwardByDecomposingPreviousCharacter(Object sender)
```

Discussion
Implemented by subclasses to delete the selection, if there is one, or a single character backward from the insertion point. If the previous character is canonically decomposable, this method should try to delete only the last character in the grapheme cluster (for example, deleting "a"+ "´" results in "a"). The *sender* argument is typically the object that invoked this method.

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

deleteForward

This action method deletes the selection, if there is one, or a single element forward from the insertion point (a letter or character in text, for example).

```
public abstract void deleteForward(Object sender)
```

deleteToBeginningOfLine

This action method deletes the selection, if there is one, or all text from the insertion point to the beginning of a line (typically of text).

```
public abstract void deleteToBeginningOfLine(Object sender)
```

Discussion
Also places the deleted text into the kill buffer.

See Also
yank (page 2002)

deleteToBeginningOfParagraph

This action method deletes the selection, if there is one, or all text from the insertion point to the beginning of a paragraph of text.

```
public abstract void deleteToBeginningOfParagraph(Object sender)
```

Discussion
Also places the deleted text into the kill buffer.

See Also
yank (page 2002)

deleteToEndOfLine

This action method deletes the selection, if there is one, or all text from the insertion point to the end of a line (typically of text).

```
public abstract void deleteToEndOfLine(Object sender)
```

Discussion
Also places the deleted text into the kill buffer.

deleteToEndOfParagraph

This action method deletes the selection, if there is one, or all text from the insertion point to the end of a paragraph of text.

```
public abstract void deleteToEndOfParagraph(Object sender)
```

Discussion
Also places the deleted text into the kill buffer.

See Also
`yank` (page 2002)

deleteToMark

This action method deletes the selection, if there is one, or all items from the insertion point to a previously placed mark, including the selection itself if not empty.

```
public abstract void deleteToMark(Object sender)
```

Discussion
Also places the deleted text into the kill buffer.

See Also
`setMark` (page 2001)
`selectToMark` (page 2001)
`yank` (page 2002)

deleteWordBackward

This action method deletes the selection, if there is one, or a single word backward from the insertion point.

```
public abstract void deleteWordBackward(Object sender)
```

deleteWordForward

This action method deletes the selection, if there is one, or a single word forward from the insertion point.

```
public abstract void deleteWordForward(Object sender)
```

doCommandBySelector

Attempts to perform the method indicated by *aSelector*.

```
public abstract void doCommandBySelector(NSSelector aSelector)
```

Discussion
The method should take a single argument of type `Object` and return `void`. If the receiver responds to *aSelector*, it invokes the method with `null` as the argument. If the receiver doesn't respond, it sends this message to its next responder with the same selector. NSWindow and NSApplication also send the message to their delegates. If the receiver has no next responder or delegate, it beeps.

See Also
`tryToPerform` (page 1199)
`sendActionToTargetFromSender` (page 121) (NSApplication)

indent

This action method indents the selection or the insertion point if there is no selection.

```
public abstract void indent(Object sender)
```

insertBacktab

This action method handles a "backward tab."

```
public abstract void insertBacktab(Object sender)
```

Discussion
A field editor might respond to this method by selecting the field before it, while a regular text object either doesn't respond to or ignores such a message.

insertNewline

This action method inserts a line-break character at the insertion point or selection, deleting the selection if there is one, or to end editing if the receiver is a text field or other field editor.

```
public abstract void insertNewline(Object sender)
```

insertNewlineIgnoringFieldEditor

This action method inserts a line-break character at the insertion point or selection, deleting the selection if there is one.

```
public abstract void insertNewlineIgnoringFieldEditor(Object sender)
```

Discussion
Unlike `insertNewline` (page 1991), this method always inserts a line-break character and doesn't cause the receiver to end editing.

insertParagraphSeparator

This action method inserts a paragraph separator at the insertion point or selection, deleting the selection if there is one.

```
public abstract void insertParagraphSeparator(Object sender)
```

insertTab

This action method inserts a tab character at the insertion point or selection, deleting the selection if there is one, or to end editing if the receiver is a text field or other field editor.

```
public abstract void insertTab(Object sender)
```

insertTabIgnoringFieldEditor

This action method inserts a tab character at the insertion point or selection, deleting the selection if there is one.

```
public abstract void insertTabIgnoringFieldEditor(Object sender)
```

Discussion

Unlike `insertTab` (page 1992), this method always inserts a tab character and doesn't cause the receiver to end editing.

insertText

Inserts *anObject* at the insertion point or selection, deleting the selection if there is one.

```
public abstract void insertText(Object anObject)
```

lowercaseWord

This action method lowercases every letter in the word or words surrounding the insertion point or selection, expanding the selection if necessary.

```
public abstract void lowercaseWord(Object sender)
```

Discussion

If either end of the selection partially covers a word, that entire word is made lowercase.

See Also

`uppercaseWord` (page 2002)

`capitalizeWord` (page 1987)

`changeCaseOfLetter` (page 1988)

moveBackward

This action method moves the selection or insertion point one element or character backward.

```
public abstract void moveBackward(Object sender)
```

Discussion
In text, if there is a selection it should be deselected, and the insertion point should be placed at the beginning of the former selection.

moveBackwardAndModifySelection

This action method expands or reduce either end of the selection backward by one element or character.

```
public abstract void moveBackwardAndModifySelection(Object sender)
```

Discussion
If the end being modified is the backward end, this method expands the selection; if the end being modified is the forward end, it reduces the selection. The first `moveBackwardAndModifySelection` or `moveForwardAndModifySelection` (page 1994) method in a series determines the end being modified by always expanding. Hence, this method results in the backward end becoming the mobile one if invoked first.

See Also
`moveLeftAndModifySelection` (page 1994)

moveDown

This action method moves the selection or insertion point one element or character down.

```
public abstract void moveDown(Object sender)
```

Discussion
In text, if there is a selection it should be deselected, and the insertion point should be placed below the beginning of the former selection.

moveDownAndModifySelection

This action method expands or reduces the top or bottom end of the selection downward by one element, character, or line (whichever is appropriate for text direction).

```
public abstract void moveDownAndModifySelection(Object sender)
```

Discussion
If the end being modified is the bottom, this method expands the selection; if the end being modified is the top, it reduces the selection. The first `moveDownAndModifySelection` or `moveUpAndModifySelection` (page 1996) method in a series determines the end being modified by always expanding. Hence, this method results in the bottom end becoming the mobile one if invoked first.

moveForward

This action method moves the selection or insertion point one element or character forward.

```
public abstract void moveForward(Object sender)
```

Discussion
In text, if there is a selection it should be deselected, and the insertion point should be placed at the end of the former selection.

moveForwardAndModifySelection

This action method expands or reduces either end of the selection forward by one element or character.

```
public abstract void moveForwardAndModifySelection(Object sender)
```

Discussion
If the end being modified is the backward end, this method reduces the selection; if the end being modified is the forward end, it expands the selection. The first `moveBackwardAndModifySelection` (page 1993) or `moveForwardAndModifySelection` method in a series determines the end being modified by always expanding. Hence, this method results in the forward end becoming the mobile one if invoked first.

See Also
`moveRightAndModifySelection` (page 1995)

moveLeft

This action method moves the selection or insertion point one element or character to the left.

```
public abstract void moveLeft(Object sender)
```

Discussion
In text, if there is a selection it should be deselected, and the insertion point should be placed at the left end of the former selection.

moveLeftAndModifySelection

Implemented by subclasses to expand or reduce either end of the selection to the left (display order) by one element or character.

```
public abstract void moveLeftAndModifySelection(Object sender)
```

Discussion
If the end being modified is the left end, this method expands the selection; if the end being modified is the right end, it reduces the selection. The first `moveLeftAndModifySelection` (page 1994) or `moveRightAndModifySelection` (page 1995) method in a series determines the end being modified by always expanding. Hence, this method results in the left end becoming the mobile one if invoked first.

The *sender* argument is typically the object that invoked this method.

The essential difference between this method and the corresponding `moveBackwardAndModifySelection` (page 1993) is that the latter method moves in logical order, which can differ in bidirectional text, whereas this method moves in display order.

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

moveRight

This action method moves the selection or insertion point one element or character to the right.

```
public abstract void moveRight(Object sender)
```

Discussion

In text, if there is a selection it should be deselected, and the insertion point should be placed at the right end of the former selection.

moveRightAndModifySelection

Implemented by subclasses to expand or reduce either end of the selection to the right (display order) by one element or character.

```
public abstract void moveRightAndModifySelection(Object sender)
```

Discussion

If the end being modified is the left end, this method reduces the selection; if the end being modified is the right end, it expands the selection. The first `moveLeftAndModifySelection` (page 1994) or `moveRightAndModifySelection` method in a series determines the end being modified by always expanding. Hence, this method results in the right end becoming the mobile one if invoked first.

The *sender* argument is typically the object that invoked this method.

The essential difference between this method and the corresponding `moveForwardAndModifySelection` (page 1994) is that the latter method moves in logical order, which can differ in bidirectional text, whereas this method moves in display order.

Availability

Available in Mac OS X v10.3 and later.

moveToBeginningOfDocument

This action method moves the selection to the first element of the document, or the insertion point to the beginning.

```
public abstract void moveToBeginningOfDocument(Object sender)
```

moveToBeginningOfLine

This action method moves the selection to the first element of the selected line, or the insertion point to the beginning of the line.

```
public abstract void moveToBeginningOfLine(Object sender)
```

moveToBeginningOfParagraph

This action method moves the insertion point to the beginning of the selected paragraph.

```
public abstract void moveToBeginningOfParagraph(Object sender)
```

moveToEndOfDocument

This action method moves the selection to the last element of the document, or the insertion point to the end.

```
public abstract void moveToEndOfDocument(Object sender)
```

moveToEndOfLine

This action method moves the selection to the last element of the selected line, or the insertion point to the end of the line.

```
public abstract void moveToEndOfLine(Object sender)
```

moveToEndOfParagraph

This action method moves the insertion point to the end of the selected paragraph.

```
public abstract void moveToEndOfParagraph(Object sender)
```

moveUp

This action method moves the selection or insertion point one element or character up.

```
public abstract void moveUp(Object sender)
```

Discussion

In text, if there is a selection it should be deselected, and the insertion point should be placed above the beginning of the former selection.

moveUpAndModifySelection

This action method expands or reduces the top or bottom end of the selection upward by one element, character, or line (whichever is appropriate for text direction).

```
public abstract void moveUpAndModifySelection(Object sender)
```

Discussion

If the end being modified is the bottom, this method reduces the selection; if the end being modified is the top, it expands the selection. The first `moveDownAndModifySelection` (page 1993) or `moveUpAndModifySelection` method in a series determines the end being modified by always expanding. Hence, this method results in the top end becoming the mobile one if invoked first.

moveWordBackward

This action method moves the selection or insertion point one word backward.

```
public abstract void moveWordBackward(Object sender)
```

Discussion

If there is a selection it should be deselected, and the insertion point should be placed at the end of the first word preceding the former selection.

See Also

`moveWordLeft` (page 1998)

moveWordBackwardAndModifySelection

This action method expands or reduces either end of the selection backward by one whole word.

```
public abstract void moveWordBackwardAndModifySelection(Object sender)
```

Discussion

If the end being modified is the backward end, this method expands the selection; if the end being modified is the forward end, it reduces the selection. The first `moveWordBackwardAndModifySelection` or `moveWordForwardAndModifySelection` (page 1997) method in a series determines the end being modified by always expanding. Hence, this method results in the backward end becoming the mobile one if invoked first.

See Also

`moveWordLeftAndModifySelection` (page 1998)

moveWordForward

This action method moves the selection or insertion point one word forward.

```
public abstract void moveWordForward(Object sender)
```

Discussion

If there is a selection it should be deselected, and the insertion point should be placed at the beginning of the first word following the former selection.

See Also

`moveWordRight` (page 1998)

moveWordForwardAndModifySelection

This action method expands or reduces either end of the selection forward by one whole word.

```
public abstract void moveWordForwardAndModifySelection(Object sender)
```

Discussion

Implemented by subclasses to move the selection or insertion point one word to the left, in display order.

If the end being modified is the backward end, this method reduces the selection; if the end being modified is the forward end, it expands the selection. The first `moveWordBackwardAndModifySelection` (page 1997) or `moveWordForwardAndModifySelection` method in a series determines the end being modified by always expanding. Hence, this method results in the forward end becoming the mobile one if invoked first.

See Also

`moveWordRightAndModifySelection` (page 1999)

moveWordLeft

Implemented by subclasses to expand or reduce either end of the selection left by one whole word in display order.

```
public abstract void moveWordLeft(Object sender)
```

Discussion

If there is a selection it should be deselected, and the insertion point should be placed at the end of the first word to the left of the former selection. The *sender* argument is typically the object that invoked this method.

The main difference between this method and the corresponding `moveWordBackward` (page 1996) method is that the latter moves in logical order, which is important in bidirectional text, whereas this method moves in display order.

Availability

Available in Mac OS X v10.3 and later.

moveWordLeftAndModifySelection

Implemented by subclasses to expand or reduce either end of the selection left by one whole word in display order.

```
public abstract void moveWordLeftAndModifySelection(Object sender)
```

Discussion

If the end being modified is the left end, this method expands the selection; if the end being modified is the right end, it reduces the selection. The first `moveWordLeftAndModifySelection` or `moveWordRightAndModifySelection` (page 1999) method in a series determines the end being modified by always expanding. Hence, this method results in the left end becoming the mobile one if invoked first.

The main difference between this method and the corresponding `moveWordBackwardAndModifySelection` (page 1997) method is that the latter moves in logical order, which is important in bidirectional text, whereas this method moves in display order.

The *sender* argument is typically the object that invoked this method.

Availability

Available in Mac OS X v10.3 and later.

moveWordRight

Implemented by subclasses to move the selection or insertion point one word right.

```
public abstract void moveWordRight(Object sender)
```

Discussion

If there is a selection it should be deselected, and the insertion point should be placed at the beginning of the first word to the right of the former selection. The *sender* argument is typically the object that invoked this method.

The main difference between this method and the corresponding `moveWordForward` (page 1997) method is that the latter moves in logical order, which is important in bidirectional text, whereas this method moves in display order.

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

moveWordRightAndModifySelection

Implemented by subclasses to expand or reduce either end of the selection to the right by one whole word.

```
public abstract void moveWordRightAndModifySelection(Object sender)
```

Discussion
If the end being modified is the backward end, this method reduces the selection; if the end being modified is the forward end, it expands the selection. The first `moveWordBackwardAndModifySelection` (page 1997) or `moveWordForwardAndModifySelection` method in a series determines the end being modified by always expanding. Hence, this method results in the forward end becoming the mobile one if invoked first. The *sender* argument is typically the object that invoked this method.

The main difference between this method and the corresponding `moveWordForwardAndModifySelection` (page 1997) method is that the latter moves in logical order, which is important in bidirectional text, whereas this method moves in display order.

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

pageDown

This action method scrolls the receiver down (or back) one page in its scroll view, also moving the insertion point to the top of the newly displayed page.

```
public abstract void pageDown(Object sender)
```

See Also
`scrollPageDown` (page 2000)
`scrollPageUp` (page 2000)

pageUp

This action method scrolls the receiver up (or forward) one page in its scroll view, also moving the insertion point to the top of the newly-displayed page.

```
public abstract void pageUp(Object sender)
```

See Also
`scrollPageDown` (page 2000)
`scrollPageUp` (page 2000)

scrollLineDown

This action method scrolls the receiver one line down in its scroll view, without changing the selection.

```
public abstract void scrollLineDown(Object sender)
```

See Also

`scrollLineUp` (page 2000)

`lineScroll` (page 1275) (NSScrollView)

scrollLineUp

This action method scrolls the receiver one line up in its scroll view, without changing the selection.

```
public abstract void scrollLineUp(Object sender)
```

See Also

`scrollLineDown` (page 2000)

`lineScroll` (page 1275) (NSScrollView)

scrollPageDown

This action method scrolls the receiver one page down in its scroll view, without changing the selection.

```
public abstract void scrollPageDown(Object sender)
```

See Also

`pageDown` (page 1999)

`pageUp` (page 1999)

`pageScroll` (page 1276) (NSScrollView)

scrollPageUp

This action method scrolls the receiver one page up in its scroll view, without changing the selection.

```
public abstract void scrollPageUp(Object sender)
```

See Also

`pageDown` (page 1999)

`pageUp` (page 1999)

`pageScroll` (page 1276) (NSScrollView)

selectAll

This action method selects all selectable elements.

```
public abstract void selectAll(Object sender)
```

selectLine

This action method selects all elements in the line or lines containing the selection or insertion point.

```
public abstract void selectLine(Object sender)
```

selectParagraph

This action method selects all paragraphs containing the selection or insertion point.

```
public abstract void selectParagraph(Object sender)
```

selectToMark

This action method selects all items from the insertion point or selection to a previously placed mark, including the selection itself if not empty.

```
public abstract void selectToMark(Object sender)
```

See Also

`setMark` (page 2001)

`deleteToMark` (page 1990)

selectWord

This action method extends the selection to the nearest word boundaries outside it (up to, but not including, word delimiters).

```
public abstract void selectWord(Object sender)
```

setMark

This action method sets a mark at the insertion point or selection, which is used by `deleteToMark` (page 1990) and `selectToMark` (page 2001).

```
public abstract void setMark(Object sender)
```

See Also

`swapWithMark` (page 2001)

swapWithMark

This action method swaps the mark and the selection or insertion point, so that what was marked is now the selection or insertion point, and what was the insertion point or selection is now the mark.

```
public abstract void swapWithMark(Object sender)
```

See Also

`setMark` (page 2001)

transpose

This action method transposes the characters to either side of the insertion point and advances the insertion point past both of them. Does nothing to a selected range of text.

```
public abstract void transpose(Object sender)
```

transposeWords

This action method transposes the two words prior to the insertion point and advances the insertion point past both of them. Not currently implemented by NSTextView.

```
public abstract void transposeWords(Object sender)
```

uppercaseWord

This action method makes uppercase every letter in the word or words surrounding the insertion point or selection, expanding the selection if necessary.

```
public abstract void uppercaseWord(Object sender)
```

Discussion

If either end of the selection partially covers a word, that entire word is made uppercase.

See Also

`lowercaseWord` (page 1992)

`capitalizeWord` (page 1987)

`changeCaseOfLetter` (page 1988)

yank

This action method replaces the insertion point or selection with text from the kill buffer.

```
public abstract void yank(Object sender)
```

Discussion

If invoked sequentially, cycles through the kill buffer in reverse order.

See Also

`deleteToBeginningOfLine` (page 1989)

`deleteToEndOfLine` (page 1989)

`deleteToBeginningOfParagraph` (page 1989)

`deleteToEndOfParagraph` (page 1990)

`deleteToMark` (page 1990)

NSKeyValueBindingCreation

(informal protocol)

Package:	com.apple.cocoa.application
Availability	Available in Mac OS X v10.3 and later.
Companion guide	Cocoa Bindings Programming Topics

Overview

The NSKeyValueBindingCreation interface provides methods to create and remove bindings between view objects and controllers or controllers and model objects. In addition, it provides a means for a view subclass to advertise the bindings that it exposes.

When a new binding is created it relates the receiver's binding (for example, a property of the view object) to a property of the observable object specified by a key path. When the value of the specified property of the observable object changes, the receiver is notified using the key-value observing mechanism. A binding also specifes binding options that can further customize how the observing and the observed objects interact.

Bindings between objects are typically established in Interface Builder using the Bindings inspector. However, there are times it must be done programmatically, such as when establishing a binding between objects in different nib files.

Tasks

Managing Bindings

`valueClassForBinding` (page 2005)
: Returns the class of the value that will be returned for the specified *`binding`*.

`bind` (page 2004)
: Establishes a binding between the receiver's *`binding`* property and the property of *`observableController`* specified by *`keyPath`*.

`infoForBinding` (page 2004)
: Returns a dictionary describing the receiver's *`binding`*.

`unbind` (page 2004)
: Removes the *`binding`* between the receiver and a controller.

`exposedBindings` (page 2004)
: Returns an array containing the bindings exposed by the receiver.

Instance Methods

bind

Establishes a binding between the receiver's *binding* property and the property of *observableController* specified by *keyPath*.

```
public abstract void bind(String binding, Object observableController, String
    keyPath, NSDictionary options)
```

Discussion
The *binding* is the key path for a property of the receiver previously exposed. The *options* dictionary is optional. If present, it contains placeholder objects or an NSValueTransformer identifier as described in "Constants" (page 2005).

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

See Also
`unbind` (page 2004)

exposedBindings

Returns an array containing the bindings exposed by the receiver.

```
public abstract NSArray exposedBindings()
```

Discussion
Override this method to remove bindings that are exposed by a superclass that are not appropriate for the subclass.

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

infoForBinding

Returns a dictionary describing the receiver's *binding*.

```
public abstract NSDictionary infoForBinding(NSString binding)
```

Discussion
The returned dictionary contains an `NSObservedObjectKey`, `NSObservedKeyPathKey` and `NSOptionsKey`.

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

unbind

Removes the *binding* between the receiver and a controller.

```
public abstract void unbind(String binding)
```

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

See Also
`bind` (page 2004)

valueClassForBinding

Returns the class of the value that will be returned for the specified *binding*.

```
public abstract Class valueClassForBinding(String binding)
```

Discussion
This method is used by Interface Builder to determine the appropriate transformers for a binding.

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

Constants

The following values are used as keys in the options dictionary passed to the `bind` (page 2004) method. These keys are also used in the dictionary returned as the `NSOptionsKey` value of `infoForBinding` (page 2004). See the *Cocoa Bindings Reference* for more information

Key	Description
NSAllowsEditingMultipleValuesSelectionBindingOption	An NSNumber containing a Boolean value that determines if the binding allows editing when the value represents a multiple selection. Available in Mac OS X v10.4 and later.
NSAllowsNullArgumentBindingOption	An NSNumber containing a Boolean value that determines if the argument bindings allows passing argument values of `null`. Available in Mac OS X v10.4 and later.
NSConditionallySetsEditableBindingOption	An NSNumber containing a Boolean value that determines if the editable state of the user interface item is automatically configured based on the controller's selection. Available in Mac OS X v10.4 and later.
NSConditionallySetsEnabledBindingOption	An NSNumber containing a Boolean value that determines if the enabled state of the user interface item is automatically configured based on the controller's selection. Available in Mac OS X v10.4 and later.

Key	Description
NSConditionallySetsHiddenBindingOption	An NSNumber containing a Boolean value that determines if the hidden state of the user interface item is automatically configured based on the controller's selection. Available in Mac OS X v10.4 and later.
NSContinuouslyUpdatesValueBindingOption	An NSNumber containing a Boolean value that determines whether the value of the binding is updated as edits are made to the user interface item or is updated only when the user interface item resigns as the responder. Available in Mac OS X v10.4 and later.
NSCreatesSortDescriptorBindingOption	An NSNumber containing a Boolean value that determines if a sort descriptor is created for a table column. If this value is false, then the table column does not allow sorting. Available in Mac OS X v10.4 and later.
NSDeletesObjectsOnRemoveBindingsOption	An NSNumber containing a Boolean value that determines if an object is deleted from the managed context immediately upon being removed from a relationship. Available in Mac OS X v10.4 and later.
NSDisplayNameBindingOption	An NSString containing a human readable string to be displayed for a predicate. Available in Mac OS X v10.4 and later.
NSDisplayPatternBindingOption	An NSString that specifies a format string used to construct the final value of a string. Available in Mac OS X v10.4 and later.
NSHandlesContentAsCompoundValueBindingOption	An NSNumber containing a Boolean value that determines if the content is treated as a compound value. Available in Mac OS X v10.4 and later.
NSInsertsNullPlaceholderBindingOption	An NSNumber containing a Boolean value that determines if an additional item which represents null is inserted into a matrix or pop-up menu before the items in the content array. Available in Mac OS X v10.4 and later.
NSInvokesSeparatelyWithArrayObjectsBindingOption	An NSNumber containing a Boolean value that determines whether the specified selector is invoked with the array as the argument or is invoked repeatedly with each array item as an argument. Available in Mac OS X v10.4 and later.

Key	Description
NSMultipleValuesPlaceholderBindingOption	An object that is used as a placeholder when the key path of the bound controller returns `multipleValuesMarker` (page 475) for a binding. Available in Mac OS X v10.4 and later.
NSNoSelectionPlaceholderBindingOption	An object that is used as a placeholder when the key path of the bound controller returns `noSelectionMarker` (page 475) for a binding. Available in Mac OS X v10.4 and later.
NSNotApplicablePlaceholderBindingOption	An object that is used as a placeholder when the key path of the bound controller returns for a binding. Available in Mac OS X v10.4 and later.
NSNullPlaceholderBindingOption	An object that is used as a placeholder when the key path of the bound controller returns `null` for a binding. Available in Mac OS X v10.4 and later.
NSRaisesForNotApplicableKeysBindingOption	An NSNumber containing a Boolean value that specifies if an exception is raised when the binding is bound to a key that is not applicable—for example when an object is not key-value coding compliant for a key. Available in Mac OS X v10.4 and later.
NSPredicateFormatBindingOption	An NSString containing the predicate pattern string for the predicate bindings. Use $value to refer to the value in the search field. Available in Mac OS X v10.4 and later.
NSSelectorNameBindingOption	An NSString that specifies the method selector invoked by the target binding when the user interface item is clicked. Available in Mac OS X v10.4 and later.
NSSelectsAllWhenSettingContentBindingOption	An NSNumber containing a Boolean value that specifies if all the items in the array controller are selected when the content is set. Available in Mac OS X v10.4 and later.
NSValidatesImmediatelyBindingOption	An NSNumber containing a Boolean value that determines if the contents of the binding are validated immediately. Available in Mac OS X v10.4 and later.
NSValueTransformerNameBindingOption	The value for this key is an identifier of a registered NSValueTransformer instance that is applied to the bound value. Available in Mac OS X v10.4 and later.

Key	Description
NSValueTransformerBindingOption	An NSValueTransformer instance that is applied to the bound value. Available in Mac OS X v10.4 and later.

The following values are used as keys in the dictionary returned by `infoForBinding` (page 2004)

Key	Description
NSObservedObjectKey	The object that is the observable controller of the binding . Available in Mac OS X v10.4 and later.
NSObservedKeyPathKey	An NSString containing the key path of the binding. Available in Mac OS X v10.4 and later.
`NSOptionsKey`	An NSDictionary containing key value pairs as specified in the options dictionary when the binding was created. Available in Mac OS X v10.4 and later.

The following values are used to specify a binding to `bind` (page 2004), `infoForBinding` (page 2004), `unbind` (page 2004) and `valueClassForBinding` (page 2005). See the *Cocoa Bindings Reference* for more information.

Key	Description
NSAlignmentBinding	Available in Mac OS X v10.4 and later.
NSAlternateImageBinding	Available in Mac OS X v10.4 and later.
NSAlternateTitleBinding	Available in Mac OS X v10.4 and later.
NSAnimateBinding	Available in Mac OS X v10.4 and later.
NSAnimationDelayBinding	Available in Mac OS X v10.4 and later.
NSArgumentBinding	Available in Mac OS X v10.4 and later.
NSAttributedStringBinding	Available in Mac OS X v10.4 and later.
NSContentArrayBinding	Available in Mac OS X v10.4 and later.
NSContentArrayForMultipleSelectionBinding	Available in Mac OS X v10.4 and later.
NSContentBinding	Available in Mac OS X v10.4 and later.
NSContentHeightBinding	Available in Mac OS X v10.4 and later.
NSContentObjectBinding	Available in Mac OS X v10.4 and later.
NSContentObjectsBinding	Available in Mac OS X v10.4 and later.
NSContentSetBinding	Available in Mac OS X v10.4 and later.

Key	Description
NSContentValuesBinding	Available in Mac OS X v10.4 and later.
NSContentWidthBinding	Available in Mac OS X v10.4 and later.
NSCriticalValueBinding	Available in Mac OS X v10.4 and later.
NSDataBinding	Available in Mac OS X v10.4 and later.
NSObservedObjectKey	Available in Mac OS X v10.4 and later.
NSDisplayPatternTitleBinding	Available in Mac OS X v10.4 and later.
NSDisplayPatternValueBinding	Available in Mac OS X v10.4 and later.
NSDocumentEditedBinding	Available in Mac OS X v10.4 and later.
NSEditableBinding	Available in Mac OS X v10.4 and later.
NSEnabledBinding	Available in Mac OS X v10.4 and later.
NSFontBinding	Available in Mac OS X v10.4 and later.
NSFontBoldBinding	Available in Mac OS X v10.4 and later.
NSFontFamilyNameBinding	Available in Mac OS X v10.4 and later.
NSFontItalicBinding	Available in Mac OS X v10.4 and later.
NSFontNameBinding	Available in Mac OS X v10.4 and later.
NSFontSizeBinding	Available in Mac OS X v10.4 and later.
NSHeaderTitleBinding	Available in Mac OS X v10.4 and later.
NSHiddenBinding	Available in Mac OS X v10.4 and later.
NSImageBinding	Available in Mac OS X v10.4 and later.
NSIsIndeterminateBinding	Available in Mac OS X v10.4 and later.
NSLabelBinding	Available in Mac OS X v10.4 and later.
NSManagedObjectContextBinding	Available in Mac OS X v10.4 and later.
NSMaxValueBinding	Available in Mac OS X v10.4 and later.
NSMaxWidthBinding	Available in Mac OS X v10.4 and later.
NSMinValueBinding	Available in Mac OS X v10.4 and later.
NSMinWidthBinding	Available in Mac OS X v10.4 and later.
NSMixedStateImageBinding	Available in Mac OS X v10.4 and later.

Key	Description
NSOffStateImageBinding	Available in Mac OS X v10.4 and later.
NSOnStateImageBinding	Available in Mac OS X v10.4 and later.
NSPredicateBinding	Available in Mac OS X v10.4 and later.
NSRecentSearchesBinding	Available in Mac OS X v10.4 and later.
NSRepresentedFilenameBinding	Available in Mac OS X v10.4 and later.
NSRowHeightBinding	Available in Mac OS X v10.4 and later.
NSSelectedIdentifierBinding	Available in Mac OS X v10.4 and later.
NSSelectedIndexBinding	Available in Mac OS X v10.4 and later.
NSSelectedLabelBinding	Available in Mac OS X v10.4 and later.
NSSelectedObjectBinding	Available in Mac OS X v10.4 and later.
NSSelectedObjectsBinding	Available in Mac OS X v10.4 and later.
NSSelectedTagBinding	Available in Mac OS X v10.4 and later.
NSSelectedValueBinding	Available in Mac OS X v10.4 and later.
NSSelectedValuesBinding	Available in Mac OS X v10.4 and later.
NSSelectionIndexesBinding	Available in Mac OS X v10.4 and later.
NSSelectionIndexPathsBinding	Available in Mac OS X v10.4 and later.
NSSortDescriptorsBinding	Available in Mac OS X v10.4 and later.
NSTargetBinding	Available in Mac OS X v10.4 and later.
NSTextColorBinding	Available in Mac OS X v10.4 and later.
NSTitleBinding	Available in Mac OS X v10.4 and later.
NSToolTipBinding	Available in Mac OS X v10.4 and later.
NSValueBinding	Available in Mac OS X v10.4 and later.
NSValuePathBinding	Available in Mac OS X v10.4 and later.
NSValueURLBinding	Available in Mac OS X v10.4 and later.
NSVisibleBinding	Available in Mac OS X v10.4 and later.
NSWarningValueBinding	Available in Mac OS X v10.4 and later.
NSWidthBinding	Available in Mac OS X v10.4 and later.

NSMenu.MenuValidation

(informal protocol)

Package:	com.apple.cocoa.application
Companion guide	Application Menu and Pop-up List Programming Topics for Cocoa

Overview

This interface allows your application to update the enabled or disabled status of an NSMenuItem. It declares only one method, `validateMenuItem` (page 2011).

Tasks

Validating Menu Items

`validateMenuItem` (page 2011)

Instance Methods

validateMenuItem

```
public abstract boolean validateMenuItem(_NSObsoleteMenuItemProtocol menuItem)
```

Discussion
Implemented to override the default action of enabling or disabling *`menuItem`*. The object implementing this method must be the target of *`menuItem`*. It returns `true` to enable *`menuItem`*, `false` to disable it. You can determine which menu item *`menuItem`* is by querying it for its tag or action.

NSOutlineView.DataSource

(informal protocol)

Package: com.apple.cocoa.application

Companion guide Outline View Programming Topics for Cocoa

Overview

NSOutlineView (page 1027) objects support a data source delegate in addition to the regular delegate object. The data source delegate provides data and information about that data to the outline view. The regular delegate object handles all other delegate responsibilities for the outline view. Specifying `null` as the *`item`* will refer to the "root" item. NSOutlineView requires that each item in the outline view be unique.

> **Note:** Some of the methods in this `interface`, such as `outlineViewChildOfItem` (page 2015) and `outlineViewNumberOfChildrenOfItem` (page 2016) along with other methods that return data, are called very frequently, so they must be efficient.

Tasks

Working with Items in a View

`outlineViewChildOfItem` (page 2015)
Invoked by *`outlineView`*, and returns the child item at the specified *`index`*.

`outlineViewIsItemExpandable` (page 2015)
Invoked by *`outlineView`*. This method should return `true` if *`item`* can be expanded to display its children.

`outlineViewNumberOfChildrenOfItem` (page 2016)
Invoked by *`outlineView`* to return the number of child items encompassed by *`item`*.

`outlineViewObjectValueForItem` (page 2016)
Invoked by *`outlineView`* to return the data object associated with the specified *`item`*.

`outlineViewSetObjectValueForItem` (page 2017)
Invoked by *`outlineView`* to set the data object for the specified *`item`* to

Dragging and Dropping

`outlineViewAcceptDrop` (page 2014)
Invoked by *outlineView* when the mouse button is released while the cursor is over *item* at the child location specified by *index*. *outlineView* must have previously decided to allow a drop.

`outlineViewValidateDrop` (page 2017)
Used by *outlineView* to determine a valid drop target.

`outlineViewNamesOfPromisedFilesDroppedAtDestination` (page 2016)
Returns an array of filenames (not full paths) for the created files that the receiver promises to create.

Object Persistence

`outlineViewItemForPersistentObject` (page 2015)
Invoked by *outlineView* to return the item for the archived *object*.

`outlineViewPersistentObjectForItem` (page 2016)
Invoked by *outlineView* to return an archived object for *item*.

Working with a Pasteboard

`outlineViewWriteItemsToPasteboard` (page 2018)
Invoked by *outlineView* after it has been determined that a drag should begin, but before the drag has been started.

Sorting

`outlineViewSortDescriptorsDidChange` (page 2017)
Invoked by *outlineView* to notify the data source that the descriptors changed and the data may need to be resorted.

Instance Methods

outlineViewAcceptDrop

Invoked by *outlineView* when the mouse button is released while the cursor is over *item* at the child location specified by *index*. *outlineView* must have previously decided to allow a drop.

```
public abstract boolean outlineViewAcceptDrop(NSOutlineView outlineView,
    NSDraggingInfo info, Object item, int index)
```

Discussion
info contains more details on this dragging operation. The data source should incorporate the data from the dragging pasteboard at this time.

The return value indicates success or failure of the drag operation to the system. Return `true` if the data was used, and `false` if the data could not be deposited for some reason.

Implementation of this method is optional.

See Also

`shouldCollapseAutoExpandedItemsForDeposited` (page 1037)

outlineViewChildOfItem

Invoked by *outlineView*, and returns the child item at the specified *index*.

```
public abstract object outlineViewChildOfItem(NSOutlineView outlineView, int index,
     Object item)
```

Discussion

Children of a given parent *item* are accessed sequentially. If *item* is `null`, this method should return the appropriate child item of the root object.

Implementation of this method is required.

Note: `outlineViewChildOfItem` is called very frequently, so it must be efficient.

See Also

`outlineViewNumberOfChildrenOfItem` (page 2016)

outlineViewIsItemExpandable

Invoked by *outlineView*. This method should return `true` if *item* can be expanded to display its children.

```
public abstract boolean outlineViewIsItemExpandable(NSOutlineView outlineView,
    Object item)
```

Discussion

Implementation of this method is required.

outlineViewItemForPersistentObject

Invoked by *outlineView* to return the item for the archived *object*.

```
public abstract Object outlineViewItemForPersistentObject(NSOutlineView outlineView,
     Object object)
```

Discussion

If the item is an archived object, this method may return the object. You must implement this method if you are automatically saving expanded items (that is, `autosaveExpandedItems` (page 1032) returns `true`). When the outline view is restoring the saved expanded items, this method is called for each expanded item, to translate the archived object to an outline view item.

Implementation of this method is optional.

outlineViewNamesOfPromisedFilesDroppedAtDestination

Returns an array of filenames (not full paths) for the created files that the receiver promises to create.

```
public abstract NSArray
    outlineViewNamesOfPromisedFilesDroppedAtDestination(NSOutlineView outlineView,
     URL dropDestination, NSArray items)
```

Discussion

The URL `dropDestination` represents the drop location where the files are created. For more information on file promise dragging, see documentation on the NSDraggingSource protocol and `namesOfPromisedFilesDroppedAtDestination` (page 1967).

Availability

Available in Mac OS X v10.4 and later.

outlineViewNumberOfChildrenOfItem

Invoked by *outlineView* to return the number of child items encompassed by *item*.

```
public abstract int outlineViewNumberOfChildrenOfItem(NSOutlineView outlineView,
    Object object)
```

Discussion

If *item* is `null`, this method should return the number of children for the top-level item.

Implementation of this method is required.

> **Note:** `outlineViewNumberOfChildrenOfItem` is called very frequently, so it must be efficient.

outlineViewObjectValueForItem

Invoked by *outlineView* to return the data object associated with the specified *item*.

```
public abstract Object outlineViewObjectValueForItem(NSOutlineView outlineView,
    NSTableColumn tableColumn, Object item)
```

Discussion

The item is located in the specified *tableColumn* of the view.

> **Note:** NSOutlineView requires that each item in the outline view be unique.

Implementation of this method is required.

outlineViewPersistentObjectForItem

Invoked by *outlineView* to return an archived object for *item*.

```
public abstract Object outlineViewPersistentObjectForItem(NSOutlineView outlineView,
     Object item)
```

Discussion
If the item is an archived object, this method may return the item. You must implement this method if you are automatically saving expanded items (that is, `autosaveExpandedItems` (page 1032) returns `true`). When the outline view is saving the expanded items, this method is called for each expanded item, to translate the outline view item to an archived object.

Implementation of this method is optional.

outlineViewSetObjectValueForItem

Invoked by *outlineView* to set the data object for the specified *item* to

```
public abstract void outlineViewSetObjectValueForItem(NSOutlineView outlineView,
    Object object, NSTableColumn tableColumn, Object item)
```

Discussion
object. The *item* is located in the specified *tableColumn* of the view.

Implementation of this method is optional.

outlineViewSortDescriptorsDidChange

Invoked by *outlineView* to notify the data source that the descriptors changed and the data may need to be resorted.

```
public abstract void outlineViewSortDescriptorsDidChange(NSOutlineView outlineView,
     NSArray oldDescriptors)
```

Discussion
The data source typically sorts and reloads the data, and adjusts the selections accordingly. The *oldDescriptors* array contains the previous descriptors.

Implementation of this method is optional.

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

outlineViewValidateDrop

Used by *outlineView* to determine a valid drop target.

```
public abstract int outlineViewValidateDrop(NSOutlineView outlineView, NSDraggingInfo
     info, Object item, int index)
```

Discussion
Based on the mouse position, the outline view will suggest a proposed drop location. The proposed parent is *item* and the proposed child location is *index*. *info* contains more details on this dragging operation. This method must return a value that indicates which dragging operation the data source will perform. The data source may "retarget" a drop if desired by calling `setDropItemAndDropChildIndex` (page 1036) and returning something other than `NSDraggingInfo.DragOperationNone`. You may choose to retarget for various reasons (for example, for better visual feedback when inserting into a sorted position).

Implementation of this method is optional.

outlineViewWriteItemsToPasteboard

Invoked by *outlineView* after it has been determined that a drag should begin, but before the drag has been started.

```
public abstract boolean outlineViewWriteItemsToPasteboard(NSOutlineView outlineView,
     NSArray items, NSPasteboard pboard)
```

Discussion

To refuse the drag, return `false`. To start a drag, return `true` and place the drag data onto the *pboard* (data, owner, and so on). The drag image and other drag-related information will be set up and provided by the outline view once this call returns with `true`. *items* is the list of items that will be participating in the drag.

Implementation of this method is optional.

NSTableView.DataSource

(informal protocol)

Package:	com.apple.cocoa.application
Companion guide	Table View Programming Guide

Overview

The NSTableView.DataSource interface declares the methods that an NSTableView (page 1437) uses to access the contents of its data source object.

> **Note:** Some of the methods in this interface, such as `tableViewObjectValueForLocation` (page 2021) and `numberOfRowsInTableView` (page 2020) along with other methods that return data, are called very frequently, so they must be efficient.

Tasks

Getting Values

`numberOfRowsInTableView` (page 2020)
Returns the number of records managed for *aTableView* by the data source object.

`tableViewObjectValueForLocation` (page 2021)
Returns an attribute value for the record in *aTableView* at *rowIndex*.

Setting Values

`tableViewSetObjectValueForLocation` (page 2021)
Sets an attribute value for the record in *aTableView* at *rowIndex*.

Dragging

`tableViewAcceptDrop` (page 2020)
Invoked by *tableView* when the mouse button is released over a table view that previously decided to allow a drop.

`tableViewNamesOfPromisedFilesDroppedAtDestination` (page 2021)
Returns an array of filenames that represent the *indexSet* rows for a drag to *dropDestination*.

`tableViewValidateDrop` (page 2022)
Used by *tableView* to determine a valid drop target.

Sorting

`tableViewSortDescriptorsDidChange` (page 2021)
Invoked by *tableView* to indicate that sorting may need to be done.

Deprecated Methods

`tableViewWriteRowsToPasteboard` (page 2022)
This method has been deprecated. You should implement the variant including an NSIndexSet parameter instead.

Instance Methods

numberOfRowsInTableView

Returns the number of records managed for *aTableView* by the data source object.

```
public abstract int numberOfRowsInTableView(NSTableView aTableView)
```

Discussion
An NSTableView uses this method to determine how many rows it should create and display.

Note: `numberOfRowsInTableView` is called very frequently, so it must be efficient.

tableViewAcceptDrop

Invoked by *tableView* when the mouse button is released over a table view that previously decided to allow a drop.

```
public abstract boolean tableViewAcceptDrop(NSTableView tableView, NSDraggingInfo
     info, int row, int operation)
```

Discussion
info contains details on this dragging operation. The proposed location is *row* and action is *operation*. The data source should incorporate the data from the dragging pasteboard at this time.

To accept a drop on the second row, *row* would be 2 and *operation* would be `NSTableView.DropOn`. To accept a drop below the last row, *row* would be `[tableView.numberOfRows()]` and *operation* would be `NSTableView.DropAbove`.

Implementation of this method is optional.

tableViewNamesOfPromisedFilesDroppedAtDestination

Returns an array of filenames that represent the *indexSet* rows for a drag to *dropDestination*.

```
public abstract NSArray tableViewNamesOfPromisedFilesDroppedAtDestination(NSTableView
     tv, URL dropDestination, NSIndexSet indexSet)
```

Discussion

This method is called when a destination has accepted a promise drag. You should return an array containing the filenames, not the full paths, for the files that you will provide.

Availability

Available in Mac OS X v10.4 and later.

tableViewObjectValueForLocation

Returns an attribute value for the record in *aTableView* at *rowIndex*.

```
public abstract Object tableViewObjectValueForLocation(NSTableView aTableView,
    NSTableColumn aTableColumn, int rowIndex)
```

Discussion

aTableColumn contains the identifier for the attribute, which you get by using NSTableColumn's `identifier` (page 1424) method. For example, if *aTableColumn* stands for the city 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".

> **Note:** `tableViewObjectValueForLocation` is called each time the table cell needs to be redisplayed, so it must be efficient.

tableViewSetObjectValueForLocation

Sets an attribute value for the record in *aTableView* at *rowIndex*.

```
public abstract void tableViewSetObjectValueForLocation(NSTableView aTableView,
    Object anObject, NSTableColumn aTableColumn, int rowIndex)
```

Discussion

anObject is the new value, and *aTableColumn* contains the identifier for the attribute, which you get by using NSTableColumn's `identifier` (page 1424) method.

tableViewSortDescriptorsDidChange

Invoked by *tableView* to indicate that sorting may need to be done.

```
public abstract void tableViewSortDes945 Tm(tv)Tj/F5 9 Tf1 0 0 1 14T
```

Discussion

The *tv* parameter represents the table view. To refuse the drag, return `false`. To start a drag, return `true` and place the drag data onto *pboard* (data, owner, and so on). The drag image and other drag-related information will be set up and provided by the table view once this call returns with `true`. *rowIndexes* is an index set of row numbers that will be participating in the drag.

Implementation of this method is optional.

Availability

Available in Mac OS X v10.4 and later.

NSTextInput

Package: com.apple.cocoa.application

Companion guides Text System Overview
Text Input Management

Overview

The NSTextInput interface defines the methods that Cocoa text views must implement in order to interact properly with the text input management system. NSTextView and its abstract superclass NSText are the only classes included in Cocoa that implement NSTextInput. To create another text view class, you can either subclass NSTextView (and not NSText, for historical reasons), or subclass NSView and implement the NSTextInput interface.

Tasks

Marked Text

`hasMarkedText` (page 2027)
Returns `true` if the receiver has marked text, `false` if it doesn't.

`markedRange` (page 2028)
Returns the range of the marked text.

`selectedRange` (page 2028)
Returns the range of selected text.

`setMarkedTextAndSelectedRange` (page 2028)
Replaces text in *`selRange`* within receiver's text storage with the contents of *`aString`*, which the receiver must display distinctively to indicate that it is marked text.

`unmarkText` (page 2029)
Removes any marking from pending input text, and disposes of the marked text as it wishes. The text view should accept the marked text as if it had been inserted normally.

`validAttributesForMarkedText` (page 2029)
Returns an array of String names for the attributes supported by the receiver.

Text Storage

`attributedSubstringWithRange` (page 2026)
Returns attributed string at *theRange*.

`insertText` (page 2028)
Inserts *aString* into the receiver's text storage.

Character Coordinates

`characterIndexForPoint` (page 2026)
Returns the index of the character whose frame rectangle includes *thePoint*.

`firstRectForCharacterRange` (page 2027)

Key Bindings

`doCommandBySelector` (page 2027)
Invokes *aSelector* if possible.

Other

`conversationIdentifier` (page 2027)
Returns a number used to identify the receiver's context to the input server.

Instance Methods

attributedSubstringWithRange

Returns attributed string at *theRange*.

```
public abstract NSAttributedString attributedSubstringWithRange(NSRange theRange)
```

Discussion
This method allows input mangers to query any range in text storage.

An implementation of this method should be prepared *theRange* to be out-of-bounds. The InkWell text input service can ask for the contents of the text input client that extends beyond the document's range. In this case, you should return the intersection of the document's range and *theRange*. If the location of *theRange* is completely outside of the document's range, return `null`.

characterIndexForPoint

Returns the index of the character whose frame rectangle includes *thePoint*.

```
public abstract int characterIndexForPoint(NSPoint thePoint)
```

Discussion
The returned index measures from the start of the receiver's text storage. *thePoint* is in the screen coordinate system. Returns `NSArray.NotFound` if the cursor is not within a character.

conversationIdentifier

Returns a number used to identify the receiver's context to the input server.

```
public abstract int conversationIdentifier()
```

Discussion
Each text view within an application should return a unique identifier (typically its address). However, multiple text views sharing the same text storage must all return the same identifier.

doCommandBySelector

Invokes *aSelector* if possible.

```
public abstract void doCommandBySelector(NSSelector aSelector)
```

Discussion
If *aSelector* cannot be invoked, then `doCommandBySelector` should not pass this message up the responder chain. `NSResponder` also implements this method, and it does forward uninvokable commands up the responder chain, but a text view should not. A text view implementing the NSTextInput interface will inherit from `NSView`, which inherits from `NSResponder`, so your implementation of this method will override the one in `NSResponder`. It should not call `super`.

See Also
`interpretKeyEvents` (page 1191) (NSResponder)
`doCommandBySelector` (page 1990) (NSKeyBindingResponder)

firstRectForCharacterRange

```
public abstract NSRect firstRectForCharacterRange(NSRange theRange)
```

Discussion
Returns the first frame rectangle for characters in *theRange*, in screen coordinates. If *theRange* spans multiple lines of text in the text view, the rectangle returned is the one for the characters in the first line. If the length of *theRange* is 0 (as it would be if there is nothing selected at the insertion point), the rectangle will coincide with the insertion point, and its width will be 0.

hasMarkedText

Returns `true` if the receiver has marked text, `false` if it doesn't.

```
public abstract boolean hasMarkedText()
```

Discussion
Unlike other methods in this protocol, this one is not called by an input server. The text view itself may call this method to determine whether there currently is marked text. `NSTextView`, for example, disables the Edit>Copy menu item when this method returns `true`.

See Also
`markedRange` (page 2028)

insertText

Inserts *`aString`* into the receiver's text storage.

```
public abstract void insertText(Object aString)
```

Discussion
`aString` can be either a String or an NSAttributedString.

markedRange

Returns the range of the marked text.

```
public abstract NSRange markedRange()
```

Discussion
The returned range measures from the start of the receiver's text storage. The return value's `location` is `NSArray.NotFound`, and its `length` is 0 if and only if `hasMarkedText` (page 2027) returns `false`.

See Also
`setMarkedTextAndSelectedRange` (page 2028)
`unmarkText` (page 2029)
`hasMarkedText` (page 2027)

selectedRange

Returns the range of selected text.

```
public abstract NSRange selectedRange()
```

Discussion
The returned range measures from the start of the receiver's text storage. If there is no selection, the return value's `location` is `NSArray.NotFound`, and its `length` is 0.

See Also
`setMarkedTextAndSelectedRange` (page 2028)

setMarkedTextAndSelectedRange

Replaces text in *`selRange`* within receiver's text storage with the contents of *`aString`*, which the receiver must display distinctively to indicate that it is marked text.

```
public abstract void setMarkedTextAndSelectedRange(Object aString, NSRange selRange)
```

Discussion

aString must be either a String or an NSAttributedString and not `null`.

See Also

`selectedRange` (page 2028)

`unmarkText` (page 2029)

unmarkText

Removes any marking from pending input text, and disposes of the marked text as it wishes. The text view should accept the marked text as if it had been inserted normally.

```
public abstract void unmarkText()
```

See Also

`selectedRange` (page 2028)

`setMarkedTextAndSelectedRange` (page 2028)

validAttributesForMarkedText

Returns an array of String names for the attributes supported by the receiver.

```
public abstract NSArray validAttributesForMarkedText()
```

Discussion

The input server may choose to use some of these attributes in the text it inserts or in marked text. Returns an empty array if no attributes are supported. See NSAttributedString for the set of string constants that you could return in the array.

NSToolbarItem.ItemValidation

(informal protocol)

Package:	com.apple.cocoa.application
Companion guide	Toolbar Programming Topics for Cocoa

Overview

A toolbar item with a valid target and action is enabled by default. To allow a toolbar item to be disabled in certain situations, a toolbar item's target can implement the `validateToolbarItem` (page 2031) method.

> **Note:** NSToolbarItem's `validate` (page 1717) method calls this method only if the item's target has a valid action defined on its target and if the item is not a custom view item. If you want to validate a custom view item, then you have to subclass NSToolbarItem and override `validate` (page 1717).

Tasks

Validating Toolbar Items

`validateToolbarItem` (page 2031)
: If this method is implemented and returns `false`, NSToolbar will disable *`theItem`*; returning `true` causes *`theItem`* to be enabled.

Instance Methods

validateToolbarItem

If this method is implemented and returns `false`, NSToolbar will disable *`theItem`*; returning `true` causes *`theItem`* to be enabled.

```
public abstract boolean validateToolbarItem(NSToolbarItem theItem)
```

Discussion

NSToolbar only calls this method for image items.

Note: validateToolbarItem is called very frequently, so it must be efficient.

If the receiver is the target for the actions of multiple toolbar items, it's necessary to determine which toolbar item *theItem* refers to by testing the itemIdentifier.

```
public boolean validateToolbarItem (NSToolbarItem toolbarItem) {
    boolean enable = false;
    if (toolbarItem.itemIdentifier().equals(SaveDocToolbarItemIdentifier)) {
        // We will return true (save item is enabled)
        // only when the document is dirty and needs saving.
        enable = this.isDocumentEdited();
    } else if
(toolbarItem.itemIdentifier().equals(NSToolbarItem.PrintItemIdentifier)) {
        enable = true;
    }
    return enable;
}
```

See Also

validateVisibleItems (page 1702) (NSToolbar)

validate (page 1717) (NSToolbarItem)

target (page 1716) (NSToolbarItem)

action (page 1709) (NSToolbarItem)

name (NSSelector)

NSValidatedUserInterfaceItem

Package:	com.apple.cocoa.application
Companion guide	User Interface Validation

Overview

NSValidatedUserInterfaceItem works with certain user interface items to enable or disable a control automatically, depending on whether any responder in the responder chain can handle the control's action method. NSMenuItem and NSToolbarItem implement this protocol.

By conforming to this interface, your control can participate in this validation mechanism.

Tasks

Getting Information About a User Interface Item

`action` (page 2033)
Returns the selector of the receiver's action method.

`tag` (page 2033)
Returns the receiver's tag integer.

Instance Methods

action

Returns the selector of the receiver's action method.

```
public abstract NSSelector action()
```

tag

Returns the receiver's tag integer.

```
public abstract int tag()
```

Index