NSText

Inherits From: NSView : NSResponder : NSObject

Conforms To: NSChangeSpelling
NSIgnoreMisspelledWords
NSCoding (NSResponder)
NSObject (NSObject)

Declared In: AppKit/NSText.h

Class at a Glance

Purpose

NSText declares the most general programmatic interface for objects that manage text. You usually use one of its subclasses, NSTextView or NSCStringText, but both share the methods and other definitions of this class.

Principal Attributes

- Supports rich text and graphics
- Works with the Font Panel and menu
- Works with the Services facility
- Provides delegation and notification
- Works with the pasteboard
- Works with the spell-checking service

Creation

See the class description.

Commonly Used Methods

– readRTFDFromFile:	Reads an **.rtf** or **.rtfd** file.
– writeRTFDToFile:atomically:	Writes the receiver's text to a file.
– string	Returns the receiver's text, without attributes.
– RTFFromRange:	Returns the receiver's text with attributes.
– RTFDFromRange:	Returns the receiver's text with attributes and attachments.

Class Description

The NSText class declares the most general programmatic interface to objects that manage text. NSText is an abstract class with two concrete subclasses, NSCStringText and NSTextView. You can create an

NSText object when constructing your application's interface with Interface Builder, or at run time using **initWithFrame:**. For the most generic and portable use of text, this is the recommended approach. NSCStringText is most suitable for use when backward compatibility with the NEXTSTEP Text object is needed. NSTextView is a NeXT addition to the OpenStep specification that acts as the front end to NeXT's revised text system. 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.

The text classes are unlike most other classes in the Application Kit in the richness and complexity of their interface. One of their design goals is to provide a comprehensive set of text-handling features so that you'll rarely need to create a subclass. Among other things, a text object can:

- Control whether the user can select or edit text.
- Control the font and layout characteristics of its text by working with the Font Panel and menu.
- Let the user control the format of paragraphs by manipulating a ruler.
- Control the color of its text and background.
- Wrap text on a word or character basis.
- Display graphic images within its text.
- Write text to or read text from files in the form of RTFD—Rich Text Format files that contain TIFF or EPS images, or attached files.
- Let another object, the delegate, dynamically control its properties.
- Let the user copy and paste text within and between applications.
- Let the user copy and paste font and format information between NSText objects.
- Let the user check the spelling of words in its text.

Graphical user-interface building tools (such as Interface Builder) may give you access to text objects in several different configurations, such as those found in the NSTextField, NSForm, and NSScrollView objects. These classes configure a text object for their own specific purposes. Additionally, all NSTextFields, NSForms, NSButtons within the same window—in short, all objects that access a text object through associated cells—share the same text object, reducing the memory demands of an application. Thus, it's generally best to use one of these classes whenever it meets your needs, rather than create text objects yourself. If one of these classes doesn't provide enough flexibility for your purposes, you can create text objects programmatically.

Text objects typically work closely with various other objects. Some of these—such as the delegate or an embedded graphic object—require a degree of programming on your part. Others—such as the Font Panel, spell checker, or ruler—take no effort other than deciding whether the service should be enabled or disabled. Several of the following sections discuss these interrelationships.

Plain and Rich Text Objects

Text objects are differentiated into two groups: those that allow only one set of text attributes for all of their text, and those that allow multiple fonts, sizes, indents, and other attributes for different sets of characters and paragraphs. Text objects in the former group are called *plain* text objects, while those in the latter are called *rich* text objects. You can control whether a text object is plain or rich using the **setRichText:** method. Rich text objects are also capable of allowing the user to drag images and files into them. This behavior is controlled by the **setImportsGraphics:** method.

A rich NSText object can use RTF (Rich Text Format) as an interchange format. Not all RTF control words are supported: On input, an NSText object ignores any control word it doesn't recognize; some of those it can read and interpret it doesn't write out. The table below lists the RTF control words that any text object recognizes. Subclasses may recognize more.

Control Word	Read	Write
\ansi	yes	yes
\b	yes	yes
\cb	yes	yes
\cf	yes	yes
\colortbl	yes	yes
\dnn	yes	yes
\fin	yes	yes
\fn	yes	yes
\fonttbl	yes	yes
\fsn	yes	yes
\i	yes	yes
\lin	yes	yes
\margrn	yes	yes
\paperwn	yes	yes
\mac	yes	no
\margln	yes	yes
\par	yes	yes
\pard	yes	no
\pca	yes	no
\qc	yes	yes
\ql	yes	yes
\qr	yes	yes
\sn	yes	no
\tab	yes	yes
\upn	yes	yes

Notifying a Text Object's Delegate

Many of an NSText object's actions can be controlled through an associated object, the NSText object's delegate. The delegate can be any object you choose, and one delegate can control multiple NSText objects. If it implements any of the following methods, the delegate receives the corresponding message at the appropriate time:

textShouldBeginEditing:
textDidBeginEditing:
textDidChange:
textShouldEndEditing:
textDidEndEditing:

Of special note are the two "textShould" methods. These methods are requests for permission. Any time a text object begins an operation that would change its text or attributes, it uses **textShouldBeginEditing:** to request approval for the change. The delegate can return YES to permit the change, or NO to forbid it. Similarly, **textShouldEndEditing:** enables the delegate to prevent a text object from ending editing, such as when it contains an invalid value.

Adding Graphics and Other Attachments to the Text

A rich text object may allow graphics or other file attachments to be embedded in the text. Each graphic is treated as a single (possibly large) "character": The text's line height and character placement are adjusted to accommodate the graphic. Graphics are embedded in the text in either of two ways: programmatically or directly through user actions. In the programmatic approach, graphic objects can be added using **replaceRange:WithRTFD:**, or through a more specific method defined by a subclass.

An alternate means of adding an image or other attachment to the text is for the user to drag an image or other file directly into the text object. The text object automatically creates an attachment object to manage the display of the image (the implementation of attachment differs between NSTextView and NSCStringText). This feature requires a rich text object that has been configured to receive dragged images using the **setImportsGraphics:** method.

Images that have been imported can be written as part of an RTFD document. RTFD documents use a file package, or directory, to store the components of the document (the "D" stands for "directory"). The file package has the name of the document plus an **.rtfd** extension. The file package always contains a file called **TXT.rtf** for the text of the document, and one or more TIFF or EPS files for the images, plus the files for other attachments. A text object can transfer information in an RTFD document to a file and read it from a file using the **writeRTFDToFile:atomically:** and **readRTFDFromFile:** methods.

Working with the Font Panel

Text objects are designed to work with the Application Kit's font conversion system, defined by the NSFontPanel and NSFontManager classes. By default, a text object keeps the Font Panel updated with the first font in its selection, or of its typing attributes (defined below). It also changes the font in response to

messages from the Font Panel and Font menu. Such changes apply to the selected text or typing attributes for a rich text object, or to all the text in a plain text object. You can turn this behavior off using the **setUsesFontPanel:** method. Doing so is recommended for a text object that serves as a field editor, for example.

Working with Rulers and Paragraph Styles

Text objects also provide for a ruler, by which the user can edit paragraph attributes such as indents and tabs. NSCStringText uses its own ruler object, and defines some methods for altering paragraph attributes. NSTextView works with the public NSRulerView class and uses the NSTextStorage and NSParagraphStyle classes to handle paragraph attributes. To show or hide a text object's ruler, use the **toggleRuler:** action method. Similar to the Font Panel, NSTextView can be set not to use a ruler with the **setUsesRuler:** method.

Adopted Protocols

NSChangeSpelling	– changeSpelling:
NSIgnoreMisspelledWords	– ignoreSpelling:

Method Types

Creating instances	– initWithFrame:
Getting the characters	– string
Setting graphic attributes	– setBackgroundColor: – backgroundColor – setDrawsBackground: – drawsBackground
Setting behavioral attributes	– setEditable: – isEditable – setSelectable: – isSelectable – setFieldEditor: – isFieldEditor – setRichText: – isRichText – setImportsGraphics: – importsGraphics

Using the Font Panel and menu	– setUsesFontPanel: – usesFontPanel
Using the ruler	– toggleRuler: – isRulerVisible
Changing the selection	– setSelectedRange: – selectedRange
Replacing text	– replaceCharactersInRange:withRTF: – replaceCharactersInRange:withRTFD: – replaceCharactersInRange:withString: – setString:
Action methods for editing	– selectAll: – copy: – cut: – paste: – copyFont: – pasteFont: – copyRuler: – pasteRuler: – delete:
Changing the font	– changeFont: – setFont: – setFont:range: – font
Setting text alignment	– setAlignment: – alignCenter: – alignLeft: – alignRight: – alignment
Setting text color	– setTextColor: – setTextColor:range: – textColor
Setting super- and subscripting	– superscript: – subscript: – unscript:
Underlining text	– underline:
Reading and writing RTF	– readRTFDFromFile: – writeRTFDToFile:atomically: – RTFDFromRange: – RTFFromRange:

Checking spelling	– checkSpelling: – showGuessPanel:
Constraining size	– setMaxSize: – maxSize – setMinSize: – minSize – setVerticallyResizable: – isVerticallyResizable – setHorizontallyResizable: – isHorizontallyResizable – sizeToFit
Scrolling	– scrollRangeToVisible:
Setting the delegate	– setDelegate: – delegate

Instance Methods

alignCenter:

– (void)**alignCenter:**(id)*sender*

This action method applies center alignment to selected paragraphs (or all text if the receiver is a plain text object).

See also: – **alignLeft:**, – **alignRight:**, – **alignment**, – **setAlignment:**

alignLeft:

– (void)**alignLeft:**(id)*sender*

This action method applies left alignment to selected paragraphs (or all text if the receiver is a plain text object).

See also: – **alignCenter:**, – **alignRight:**, – **alignment**, – **setAlignment:**

alignRight:

– (void)**alignRight:**(id)*sender*

This action method applies right alignment to selected paragraphs (or all text if the receiver is a plain text object).

See also: – **alignLeft:**, – **alignCenter:**, – **alignment**, – **setAlignment:**

alignment

– (NSTextAlignment)**alignment**

Returns the alignment of the first selected paragraph, or of all text for a plain text object. This value is one of:

NSLeftTextAlignment
NSRightTextAlignment
NSCenterTextAlignment
NSJustifiedTextAlignment
NSNaturalTextAlignment (realized as one of the above depending on the script)

backgroundColor

– (NSColor *)**backgroundColor**

Returns the receiver's background color.

See also: – **drawsBackground**, – **setBackgroundColor:**

changeFont:

– (void)**changeFont:**(id)*sender*

This action method changes the font of the selection for a rich text object, or of all text for a plain text object. If the receiver doesn't use the Font Panel, however, this method does nothing.

This method changes the font by sending **convertFont:** messages to *sender* (which is presumed to be an NSFontManager or similarly capable object) and applying each NSFont returned to the appropriate text. If a rich text object's selection contains multiple fonts, **convertFont:** is invoked for each contiguous range of characters that share a font. See the NSFontManager class specification for more information on font conversion.

See also: – **usesFontPanel**

checkSpelling:

– (void)**checkSpelling:**(id)*sender*

This action method searches for a misspelled word in the receiver's text. 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:** 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:**

copy:

– (void)**copy:**(id)*sender*

This action method copies the selected text onto the general pasteboard, in as many formats as the receiver supports. A plain text object uses NSStringPboardType for plain text, and a rich text object also uses NSRTFPboardType.

See also: – **copyFont:**, – **copyRuler:**, – **cut:**, – **paste:**

copyFont:

– (void)**copyFont:**(id)*sender*

This action method copies the font information for the first character of the selection (or for the insertion point) onto the font pasteboard, as NSFontPboardType.

See also: – **copy:**, – **copyRuler:**, – **cut:**, – **paste:**

copyRuler:

– (void)**copyRuler:**(id)*sender*

This action method copies the paragraph style information for first selected paragraph onto the ruler pasteboard, as NSRulerPboardType, and expands the selection to paragraph boundaries.

See also: – **copy:**, – **copyFont:**, – **cut:**, – **paste:**

cut:

– (void)**cut:**(id)*sender*

This action method deletes the selected text and places it onto the general pasteboard, in as many formats as the receiver supports. A plain text object uses NSStringPboardType for plain text, and a rich text object also uses NSRTFPboardType.

See also: – **delete:**, – **copy:**, – **copyFont:**, – **copyRuler:**, – **paste:**

delegate

– (id)**delegate**

Returns the receiver’s delegate, or **nil** if it has none.

See also: – **setDelegate:**

delete:

– (void)**delete:**(id)*sender*

This action method deletes the selected text.

See also: – **cut:**

drawsBackground

– (BOOL)**drawsBackground**

Returns YES if the receiver draws its background, NO if it doesn't.

See also: – **backgroundColor**, – **setDrawsBackground:**

font

– (NSFont *)**font**

Returns the font of the first character in the receiver's text, or of the insertion point if there's no text.

See also: – **setFont:**, – **setFont:range:**

importsGraphics

– (BOOL)**importsGraphics**

Returns YES if the receiver allows the user to import files by dragging, NO if it doesn't. A text object that accepts dragged files is also a rich text object.

See also: – **isRichText**, – **setImportsGraphics:**

initWithFrame:

– (id)**initWithFrame:**(NSRect)*frameRect*

Initializes the receiver with *frameRect* as its frame rectangle. This method actually substitutes an instance of a concrete subclass of NSText, such as NSCStringText or NSTextView, depending on the platform, and configures that instance to archive itself in a manner portable across OpenStep implementations.

isEditable

– (BOOL)**isEditable**

Returns YES if the receiver allows the user to edit text, NO if it doesn't. You can change the receiver's text programmatically regardless of this setting.

If the receiver is editable, it's also selectable.

See also: – **isSelectable**, – **setEditable:**

isFieldEditor

– (BOOL)**isFieldEditor**

Returns YES if the receiver interprets Tab, Shift-Tab, and Return (Enter) as cues to end editing, and possibly to change the first responder; no if it accepts them as text input. See the NSWindow class specification for more information on field editors. By default, NSText objects don't behave as field editors.

See also: – **setFieldEditor:**

isHorizontallyResizable

– (BOOL)**isHorizontallyResizable**

Returns YES if the receiver automatically changes its width to accommodate the width of its text, NO if it doesn't.

See also: – **isVerticallyResizable**, – **setHorizontallyResizable**

isRichText

– (BOOL)**isRichText**

Returns YES if the receiver allows the user to apply attributes to specific ranges of the text, NO if it doesn't.

See also: – **importsGraphics**, – **setRichText:**

isRulerVisible

– (BOOL)**isRulerVisible**

Returns YES if the receiver's enclosing scroll view shows its ruler, NO otherwise.

See also: – **usesRuler**, – **toggleRuler:**

isSelectable

– (BOOL)**isSelectable**

Returns YES if the receiver allows the user to select text, NO if it doesn't.

See also: – **isEditable**, – **setSelectable:**

isVerticallyResizable

– (BOOL)**isVerticallyResizable**

Returns YES if the receiver automatically changes its height to accommodate the height of its text, NO if it doesn't.

See also: – **isHorizontallyResizable**, – **setVerticallyResizable**

maxSize

– (NSSize)**maxSize**

Returns the receiver's maximum size.

See also: – **minSize**, – **setMaxSize:**

minSize

– (NSSize)**minSize**

Returns the receiver's minimum size.

See also: – **maxSize**, – **setMinSize:**

paste:

– (void)**paste:**(id)*sender*

This action method pastes text from the general pasteboard at the insertion point or over the selection.

See also: – **copy:**, – **cut:**, – **pasteFont:**, – **pasteRuler:**

pasteFont:

– (void)**pasteFont:**(id)*sender*

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.

See also: – **copyFont:**, – **pasteRuler:**

pasteRuler:

– (void)**pasteRuler:**(id)*sender*

This action method pastes paragraph style information from the ruler pasteboard onto the selected paragraphs of a rich text object. It doesn't apply to a plain text object.

See also: – **copyFont:**, – **pasteRuler:**

readRTFDFromFile:

– (BOOL)**readRTFDFromFile:**(NSString *)*path*

Attempts to read the RTFD file at *path*, returning YES if successful and NO if not. *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:atomically:**

replaceCharactersInRange:withRTF:

– (void)**replaceCharactersInRange:**(NSRange)*aRange* **withRTF:**(NSData *)*rtfData*

Replaces the characters in *aRange* with RTF text interpreted from *rtfData*. This method applies only to rich text objects.

See also: – **replaceCharactersInRange:withRTFD:**, – **replaceCharactersInRange:withString:**

replaceCharactersInRange:withRTFD:

– (void)**replaceCharactersInRange:**(NSRange)*aRange* **withRTFD:**(NSData *)*rtfdData*

Replaces the characters in *aRange* with RTFD text interpreted from *rtfdData*. This method applies only to rich text objects.

See also: – **replaceCharactersInRange:withRTF:**, – **replaceCharactersInRange:withString:**

replaceCharactersInRange:withString:

– (void)**replaceCharactersInRange:**(NSRange)*aRange* **withString:**(NSString *)*aString*

Replaces the characters in *aRange* with *aString*. 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 zero. If the range's location is zero, the formatting attributes of the first character in the receiver are used.

See also: – **replaceCharactersInRange:withRTF:**, – **replaceCharactersInRange:withRTFD:**

RTFDFromRange:

– (NSData *)**RTFDFromRange:**(NSRange)*aRange*

Returns an NSData object that contains an RTFD stream corresponding to the characters and attributes within *aRange*. Raises an NSRangeException 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 **setData:forType:** method, with a second argument of NSRTFDPboardType.

See also: – **RTFFromRange:**

RTFFromRange:

– (NSData *)**RTFFromRange:**(NSRange)*range*

Returns an NSData object that contains an RTF stream corresponding to the characters and attributes within *aRange*, omitting any attachment characters and attributes. Raises an NSRangeException 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 **setData:forType:** method, with a second argument of NSRTFPboardType.

See also: – **RTFDFromRange:documentAttributes:**

scrollRangeToVisible:

– (void)**scrollRangeToVisible:**(NSRange)*aRange*

Scrolls the receiver in its enclosing scroll view so that the first characters of *aRange* are visible.

selectAll:

– (void)**selectAll:**(id)*sender*

This action method selects all of the receiver's text.

selectedRange

– (NSRange)**selectedRange**

Returns the range of selected characters.

See also: – **setSelectedRange:**

setAlignment:

– (void)**setAlignment:**(NSTextAlignment)*mode*

Sets the alignment of all the receiver's text to *mode*, which may be one of:

NSLeftTextAlignment
NSRightTextAlignment
NSCenterTextAlignment
NSJustifiedTextAlignment
NSNaturalTextAlignment (realized as one of the above depending on the script)

See also: – **alignment**, – **alignLeft:**, – **alignCenter:**, – **alignRight:**

setBackgroundColor:

– (void)**setBackgroundColor:**(NSColor *)*aColor*

Sets the receiver's background color to *aColor*.

See also: – **setDrawsBackground:**, – **backgroundColor**

setDelegate:

– (void)**setDelegate:**(id)*anObject*

Sets the receiver's delegate to *anObject*, without retaining it.

See also: – **delegate**

setDrawsBackground:

– (void)**setDrawsBackground:**(BOOL)*flag*

Controls whether the receiver draws its background. If *flag* is YES, the receiver fills its background with the background color; if *flag* is NO, it doesn't.

See also: – **setBackgroundColor:**, – **drawsBackground**

setEditable:

– (void)**setEditable:**(BOOL)*flag*

Controls whether the receiver allows the user to edit its text. If *flag* is YES, the receiver allows the user to edit text and attributes; if *flag* is NO, 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:**, – **isEditable**

setFieldEditor:

– (void)**setFieldEditor:**(BOOL)*flag*

Controls whether the receiver interprets Tab, Shift-Tab, and Return (Enter) as cues to end editing, and possibly to change the first responder. If *flag* is YES, it does; if *flag* is NO, it doesn't, instead accepting these characters as text input. See the NSWindow class specification for more information on field editors. By default, NSText objects don't behave as field editors.

See also: – **isFieldEditor**

setFont:

– (void)**setFont:**(NSFont *)*aFont*

Sets the font of all the receiver's text to *aFont*.

See also: – **setFont:range:**, – **font**

setFont:range:

– (void)**setFont:**(NSFont *)*aFont* **range:**(NSRange)*aRange*

Sets the font of characters within *aRange* to *aFont*. This method applies only to a rich text object.

See also: – **setFont:**, – **font**

setHorizontallyResizable:

– (void)**setHorizontallyResizable:**(BOOL)*flag*

Controls whether the receiver changes its width to fit the width of its text. If *flag* is YES it does; if *flag* is NO it doesn't.

See also: – **setVerticallyResizable:**, – **isHorizontallyResizable**

setImportsGraphics:

– (void)**setImportsGraphics:**(BOOL)*flag*

Controls whether the receiver allows the user to import files by dragging. If *flag* is YES, it does; if *flag* is NO, 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:**, – **importsGraphics**

setMaxSize:

– (void)**setMaxSize:**(NSSize)*aSize*

Sets the receiver's maximum size to *aSize*.

See also: – **setMinSize:**, – **maxSize**

setMinSize:

– (void)**setMinSize:**(NSSize)*aSize*

Sets the receiver's minimum size to *aSize*.

See also: – **setMaxSize:**, – **minSize**

setRichText:

– (void)**setRichText:**(BOOL)*flag*

Controls whether the receiver allows the user to apply attributes to specific ranges of the text. If *flag* is YES it does; if *flag* is NO it doesn't. If *flag* is NO, 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**, – **setImportsGraphics:**

setSelectable:

– (void)**setSelectable:**(BOOL)*flag*

Controls whether the receiver allows the user to select its text. If *flag* is YES, the receiver allows the user to select text; if *flag* is NO, 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:**, – **isSelectable**

setSelectedRange:

– (void)**setSelectedRange:**(NSRange)*aRange*

Selects the receiver's characters within *aRange*.

See also: – **selectedRange**

setString:

– (void)**setString:**(NSString *)*aString*

Replaces the receiver's entire text with *aString*, applying the formatting attributes of the old first character to its new contents.

setTextColor:

– (void)**setTextColor:**(NSColor *)*color*

Sets the text color of all characters in the receiver to *aColor*. Removes the text color attribute if *aColor* is **nil**.

See also: – **setTextColor:range** – **textColor:**

setTextColor:range:

– (void)**setTextColor:**(NSColor *)*aColor* **range:**(NSRange)*aRange*

Sets the text color of characters within *aRange* to *aColor*. Removes the text color attribute if *aColor* is **nil**. This method applies only to rich text objects.

See also: – **setTextColor:** – **textColor:**

setUsesFontPanel:

– (void)**setUsesFontPanel:**(BOOL)*flag*

Controls whether the receiver uses the Font Panel and Font menu. If *flag* is YES, 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 NO the receiver doesn't do any of this. By default, an NSText object uses the Font Panel and menu.

See also: – **usesFontPanel**

setVerticallyResizable:

– (void)**setVerticallyResizable:**(BOOL)*flag*

Controls whether the receiver changes its height to fit the height of its text. If *flag* is YES it does; if *flag* is NO it doesn't.

See also: – **setHorizontallyResizable:**, – **isVerticallyResizable**

showGuessPanel:

– (void)**showGuessPanel:**(id)*sender*

This action method opens the Spelling panel, allowing the user to make a correction during spell checking.

See also: – **checkSpelling:**

sizeToFit

– (void)**sizeToFit**

Resizes the receiver to fit its text.

See also: – **isHorizontallyResizable**, – **isVerticallyResizable**

string

– (NSString *)**string**

Returns the characters of the receiver's text.

See also: – **setString:**

subscript:

– (void)**subscript:**(id)*sender*

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.

See also: – **subscript:**, – **unscript:**, – **lowerBaseline:** (NSTextView)

superscript:

– (void)**superscript:**(id)*sender*

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.

See also: – **subscript:**, – **unscript:**, – **raiseBaseline:** (NSTextView)

textColor

– (NSColor *)**textColor**

Returns the color of the receiver's first character, or for the insertion point if there's no text.

See also: – **setTextColor:**, – **setTextColor:range:**

toggleRuler:

– (void)**toggleRuler:**(id)*sender*

This action method shows or hides the ruler, if the receiver is enclosed in a scroll view.

underline:

– (void)**underline:**(id)*sender*

This action method underlines selected text for a rich text object, or all text for a plain text object.

unscript:

– (void)**unscript:**(id)*sender*

This action method removes any superscripting or subscripting from selected text (or all text if the receiver is a plain text object).

See also: – **subscript:**, – **superscript:**, – **raiseBaseline:** (NSTextView), – **lowerBaseline:** (NSTextView)

usesFontPanel

– (BOOL)**usesFontPanel**

Returns YES if the receiver uses the Font Panel, NO otherwise.

See also: – **setUsesFontPanel:**

writeRTFDToFile:atomically:

– (BOOL)**writeRTFDToFile:**(NSString *)*path* **atomically:**(BOOL)*flag*

Writes the receiver's text as RTF with attachments to a file or directory at *path*. Returns YES on success and NO on failure. If *atomicFlag* is YES, 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:**, – **RTFDFromRange:**, – **readRTFDFromFile:**

Methods Implemented By the Delegate

textDidBeginEditing:

– (void)**textDidBeginEditing:**(NSNotification *)*aNotification*

Informs the delegate that the text object has begun editing (that it has become first responder). The name of *aNotification* is NSTextViewDidBeingEditingNotification.

textDidChange:

– (void)**textDidChange:**(NSNotification *)*aNotification*

Informs the delegate that the text object has changed its characters or formatting attributes. The name of *aNotification* is NSTextViewDidChangeNotification.

textDidEndEditing:

– (void)**textDidEndEditing:**(NSNotification *)*aNotification*

Informs the delegate that the text object has finished editing (that it has resigned first responder status). The name of *aNotification* is NSTextViewDidEndEditingNotification.

textShouldBeginEditing:

– (BOOL)**textShouldBeginEditing:**(NSText *)*aTextObject*

Invoked from a text object's implementation of **becomeFirstResponder**, this method requests permission for *aTextObject* to begin editing. If the delegate returns YES, the text object proceeds to make changes. If the delegate returns NO, the text object abandons the editing operation. This method is invoked whenever *aTextObject* attempts to become first responder.

See also: – **makeFirstResponder:** (NSWindow), – **becomeFirstResponder** (NSResponder)

textShouldEndEditing:

– (BOOL)**textShouldEndEditing:**(NSText *)*aTextObject*

Invoked from a text object's implementation of **resignFirstResponder**, this method requests permission for *aTextObject* to end editing. If the delegate returns YES, the text object proceeds to finish editing and resign first responder status. If the delegate returns NO, the text object selects all of its text and remains the first responder.

See also: – **resignFirstResponder** (NSResponder)

Notifications

NSTextDidBeginEditingNotification

Posted when an NSText object begins any operation that changes characters or formatting attributes.

The notification contains:

Notification Object	The notifying NSText object.
Userinfo	None

NSTextDidChangeNotification

Posted after an NSText object performs any operation that changes characters or formatting attributes.

The notification contains:

Notification Object	The notifying NSText object.
Userinfo	None

NSTextDidEndEditingNotification

Posted when an NSText object resigns first responder status.

The notification contains:

Notification Object	The notifying NSText object.
Userinfo	None
Key	**Value**
NSTextMovement	The keyboard operation that ended editing, an NSNumber (**int**).

The value for NSTextMovement is one of:

NSIllegalTextMovement
NSReturnTextMovement
NSTabTextMovement
NSBacktabTextMovement
NSLeftTextMovement
NSRightTextMovement
NSUpTextMovement
NSDownTextMovement

The value for NSTextMovement is typically one of the first four. NSIllegalTextMovement indicates that the text object ended editing because of something other than keyboard input, typically a mouse click in another view. NSReturnTextMovement, NSTabTextMovement, and NSBacktabTextMovement indicate that a field editor ended editing because the user pressed the key named in the value. The remaining four values indicate the same for arrow keys.