NSLayoutManager Class Reference

Cocoa > Text & Fonts

2008-12-20

Contents

NSLayoutManager Class Reference

Inherits from	NSObject
Conforms to	NSGlyphStorage NSCoding NSObject (NSObject)
Framework	/System/Library/Frameworks/AppKit.framework
Availability	Available in Mac OS X v10.0 and later.
Declared in	NSLayoutManager.h
Companion guides	Text System Overview Text Layout Programming Guide for Cocoa
Related sample code	Quartz Composer WWDC 2005 TextEdit Sketch-112 TextEditPlus TextLayoutDemo Worm

Overview

An `NSLayoutManager` object 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 `NSTextContainer` objects, and displays them in a series of `NSTextView` objects. In addition to its core function of laying out text, an `NSLayoutManager` object coordinates its `NSTextView` objects, provides services to those text views to support `NSRulerView` instances 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.

Text Antialiasing

`NSLayoutManager` 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 Appearance pane of System Preferences.

Thread Safety of NSLayoutManager

Generally speaking, a given layout manager (and associated objects) should not be used on more than one thread at a time. Most layout managers are used on the main thread, since it is the main thread on which their text views are displayed, and since background layout occurs on the main thread. If it is intended that a layout manager should be used on a background thread, first make sure that text views associated with that layout manager (if any) are not displayed while the layout manager is being used on the background thread, and, second, turn off background layout for that layout manager while it is being used on the background thread.

Noncontiguous Layout

Noncontiguous layout is an optional layout manager behavior new in Mac OS X v10.5. Previously, both glyph generation and layout were always performed, in order, from the beginning to the end of the document. When noncontiguous layout is turned on, however, the layout manager gains the option of performing glyph generation or layout for one portion of the document without having done so for previous sections. This can provide significant performance improvements for large documents.

Noncontiguous layout is not turned on automatically because direct clients of `NSLayoutManager` typically have relied on the previous behavior—for example, by forcing layout for a given glyph range, and then assuming that previous glyphs would therefore be laid out. Clients who use `NSLayoutManager` only indirectly—for example, those who use `NSTextView` without directly calling the underlying layout manager—can usually turn on noncontiguous layout without difficulty. Clients using `NSLayoutManager` directly need to examine their usage before turning on noncontiguous layout.

To turn on noncontiguous layout, use `setAllowsNonContiguousLayout:` (page 68). In addition, see the other methods in "Managing Noncontiguous Layout" (page 19), many of which enable you to ensure that glyph generation and layout are performed for specified portions of the text. The behavior of a number of other layout manager methods is affected by the state of noncontiguous layout, as noted in the discussion sections of those method descriptions.

Adopted Protocols

NSCoding
- `encodeWithCoder:`
- `initWithCoder:`

NSGlyphStorage
- `attributedString`
- `insertGlyphs:length:forStartingGlyphAtIndex:characterIndex:`
- `layoutOptions`
- `setIntAttribute:value:forGlyphAtIndex:`

Tasks

Initializing

- `init` (page 49)
  Initializes the receiver, a newly created `NSLayoutManager` object.

Setting the Text Storage

- `setTextStorage:` (page 80)
  Sets the receiver's `NSTextStorage` object.
- `textStorage` (page 91)
  Returns the receiver's text storage object.
- `attributedString` (page 23)
  Returns the text storage object from which the `NSGlyphGenerator` object procures characters for glyph generation.
- `replaceTextStorage:` (page 66)
  Replaces the `NSTextStorage` object for the group of text-system objects containing the receiver with the given text storage object.

Setting Text Containers

- `textContainers` (page 91)
  Returns the receiver's text containers.
- `addTextContainer:` (page 21)
  Appends the given text container to the series of text containers where the receiver arranges text.
- `insertTextContainer:atIndex:` (page 50)
  Inserts the given text container into the series of text containers at the given index.
- `removeTextContainerAtIndex:` (page 65)
  Removes the text container at the given index and invalidates the layout as needed.

Setting the Glyph Generator

- `setGlyphGenerator:` (page 73)
  Sets the glyph generator used by this layout manager.
- `glyphGenerator` (page 43)
  Returns the glyph generator used by this layout manager.

Invalidating Glyphs and Layout

- `invalidateGlyphsForCharacterRange:changeInLength:actualCharacterRange:` (page 52)
  Invalidates the cached glyphs for the characters in the given character range, adjusts the character indices of all the subsequent glyphs by the change in length, and invalidates the new character range.
- `invalidateGlyphsOnLayoutInvalidationForGlyphRange:` (page 53)
  Specifies explicitly when portions of the glyph stream depend on layout.
- `invalidateLayoutForCharacterRange:isSoft:actualCharacterRange:` (page 54)
  Invalidates the layout information for the glyphs mapped to the given range of characters.
- `invalidateLayoutForCharacterRange:actualCharacterRange:` (page 53)
  Invalidates the layout information for the glyphs mapped to the given range of characters.
- `invalidateDisplayForCharacterRange:` (page 52)
  Invalidates display for the given character range.
- `invalidateDisplayForGlyphRange:` (page 52)
  Marks the glyphs in the given glyph range as needing display, as well as the appropriate regions of the `NSTextView` objects that display those glyphs (using the `NSView` method `setNeedsDisplayInRect:`).
- `layoutManagerDidInvalidateLayout:` (page 97) *delegate method*
  Informs the delegate that the given layout manager has invalidated layout information (not glyph information).
- `textContainerChangedGeometry:` (page 88)
  Invalidates the layout information, and possibly glyphs, for the given text container and all subsequent `NSTextContainer` objects.
- `textContainerChangedTextView:` (page 89)
  Updates information needed to manage `NSTextView` objects in the given text container.
- `textStorage:edited:range:changeInLength:invalidatedRange:` (page 91)
  Invalidates glyph and layout information for a portion of the text in the given text storage object.

Enabling Background Layout

- `setBackgroundLayoutEnabled:` (page 69)
  Specifies whether the receiver generates glyphs and lays them out when the application's run loop is idle.
- `backgroundLayoutEnabled` (page 23)
  Indicates whether the receiver generates glyphs and lays out text when the application's run loop is idle.

Accessing Glyphs

- `insertGlyph:atGlyphIndex:characterIndex:` (page 49)
  Inserts a single glyph into the glyph stream at the given index and maps it to the character at the given character index.
- `insertGlyphs:length:forStartingGlyphAtIndex:characterIndex:` (page 50)
  Inserts the given glyphs into the glyph cache at the given index and maps them to characters beginning at the given character index.

- `isValidGlyphIndex:` (page 55)
  Indicates whether the specified index refers to a valid glyph, otherwise `NO`.
- `glyphAtIndex:` (page 42)
  Returns the glyph at *`glyphIndex`*.
- `glyphAtIndex:isValidIndex:` (page 42)
  If the given index is valid, returns the glyph at that location and optionally returns a flag indicating whether the requested index is in range.
- `replaceGlyphAtIndex:withGlyph:` (page 66)
  Replaces the glyph at the given index with a new glyph.
- `getGlyphs:range:` (page 39)
  Fills the passed-in buffer with a sequence of glyphs
- `getGlyphsInRange:glyphs:characterIndexes:glyphInscriptions:elasticBits:` (page 39)
  Returns the glyphs and information needed to perform layout for the given glyph range.
- `getGlyphsInRange:glyphs:characterIndexes:glyphInscriptions:elasticBits:bidiLevels:` (page 40)
  Returns the glyphs and information needed to perform layout for the given glyph range.
- `deleteGlyphsInRange:` (page 29)
  Deletes the glyphs in the given range from the receiver's glyph store.
- `numberOfGlyphs` (page 61)
  Returns the number of glyphs in the receiver.

Mapping Characters to Glyphs

- `setCharacterIndex:forGlyphAtIndex:` (page 70)
  Sets the index of the character corresponding to the glyph at the given glyph index.
- `characterIndexForGlyphAtIndex:` (page 26)
  Returns the index in the text storage for the first character associated with the given glyph.
- `glyphIndexForCharacterAtIndex:` (page 43)
  Returns the index of the first glyph associated with the character at the specified index.
- `characterRangeForGlyphRange:actualGlyphRange:` (page 26)
  Returns the range of characters that generated the glyphs in the given glyph range.
- `glyphRangeForCharacterRange:actualCharacterRange:` (page 47)
  Returns the range of glyphs that are generated from the characters in the given character range.

Setting Glyph Attributes

- `intAttribute:forGlyphAtIndex:` (page 51)
  Returns the value of the attribute identified by the given attribute tag for the glyph at the given index.
- `setIntAttribute:value:forGlyphAtIndex:` (page 74)
  Sets a custom attribute value for a given glyph.
- `setAttachmentSize:forGlyphRange:` (page 69)
  Sets the size at which the given glyph (assumed to be an attachment) is asked to draw in the given glyph range.

- `attachmentSizeForGlyphAtIndex:` (page 23)
  For a glyph corresponding to an attachment, returns the size for the attachment cell to occupy.
- `setDefaultAttachmentScaling:` (page 71)
  Sets the default scaling behavior to the given scaling if an attachment image is too large to fit in a text container.
- `defaultAttachmentScaling` (page 27)
  Returns the default behavior desired if an attachment image is too large to fit in a text container.
- `showAttachmentCell:inRect:characterIndex:` (page 82)
  Draws an attachment cell.

Handling Layout for Text Containers

- `setTextContainer:forGlyphRange:` (page 79)
  Sets text container where the glyphs in the given range are laid out.
- `glyphRangeForTextContainer:` (page 47)
  Returns the range of glyphs laid out within the given text container.
- `textContainerForGlyphAtIndex:effectiveRange:` (page 89)
  Returns the container in which the given glyph is laid out and (optionally) by reference the whole range of glyphs that are in that container.
- `textContainerForGlyphAtIndex:effectiveRange:withoutAdditionalLayout:` (page 90)
  Returns the container in which the given glyph is laid out and (optionally) by reference the whole range of glyphs that are in that container.
- `usedRectForTextContainer:` (page 94)
  Returns the bounding rectangle for the glyphs laid out in the given text container.
- `layoutManager:didCompleteLayoutForTextContainer:atEnd:` (page 96) *delegate method*
  Informs the delegate that the given layout manager has finished laying out text in the given text container.

Handling Line Fragment Rectangles

- `setLineFragmentRect:forGlyphRange:usedRect:` (page 75)
  Associates the given line fragment bounds with the given range of glyphs.
- `lineFragmentRectForGlyphAtIndex:effectiveRange:` (page 57)
  Returns the rectangle for the line fragment in which the given glyph is laid out and (optionally), by reference, the whole range of glyphs that are in that fragment.
- `lineFragmentRectForGlyphAtIndex:effectiveRange:withoutAdditionalLayout:` (page 58)
  Returns the line fragment rectangle containing the glyph at the given glyph index.
- `lineFragmentUsedRectForGlyphAtIndex:effectiveRange:` (page 59)
  Returns the usage rectangle for the line fragment in which the given glyph is laid and (optionally) by reference the whole range of glyphs that are in that fragment.
- `lineFragmentUsedRectForGlyphAtIndex:effectiveRange:withoutAdditionalLayout:` (page 59)
  Returns the usage rectangle for the line fragment in which the given glyph is laid and (optionally) by reference the whole range of glyphs that are in that fragment.

- `setExtraLineFragmentRect:usedRect:textContainer:` (page 72)
  Sets the bounds and container for the extra line fragment.
- `extraLineFragmentRect` (page 36)
  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 36)
  Returns the rectangle enclosing the insertion point drawn in the extra line fragment rectangle.
- `extraLineFragmentTextContainer` (page 36)
  Returns the text container that contains the extra line fragment rectangle.
- `setDrawsOutsideLineFragment:forGlyphAtIndex:` (page 72)
  Specifies whether the given glyph exceeds the bounds of the line fragment where it's laid out.
- `drawsOutsideLineFragmentForGlyphAtIndex:` (page 31)
  Indicates whether the glyph draws outside of its line fragment rectangle.

Laying Out Glyphs

- `setLocation:forStartOfGlyphRange:` (page 76)
  Sets the location for the first glyph of the given range.
- `setLocations:startingGlyphIndexes:count:forGlyphRange:` (page 77)
  Sets locations for many glyph ranges at once.
- `locationForGlyphAtIndex:` (page 60)
  Returns the location for the given glyph within its line fragment.
- `rangeOfNominallySpacedGlyphsContainingIndex:` (page 62)
  Returns the range for the glyphs around the given glyph that can be displayed using only their advancements from the font, without pairwise kerning or other adjustments to spacing.
- `getLineFragmentInsertionPointsForCharacterAtIndex:alternatePositions:inDisplayOrder:positions:characterIndexes:` (page 41)
  Returns insertion points in bulk for a given line fragment.
- `rectArrayForCharacterRange:withinSelectedCharacterRange:inTextContainer:rectCount:` (page 62)
  Returns an array of rectangles and, by reference, the number of such rectangles, that define the region in the given container enclosing the given character range.
- `rectArrayForGlyphRange:withinSelectedGlyphRange:inTextContainer:rectCount:` (page 63)
  Returns an array of rectangles and, by reference, the number of such rectangles, that define the region in the given container enclosing the given glyph range.
- `boundingRectForGlyphRange:inTextContainer:` (page 24)
  Returns a single bounding rectangle (in container coordinates) enclosing all glyphs and other marks drawn in the given text container for the given glyph range, including glyphs that draw outside their line fragment rectangles and text attributes such as underlining.
- `glyphRangeForBoundingRect:inTextContainer:` (page 45)
  Returns the smallest contiguous range for glyphs that are laid out wholly or partially within the given rectangle in the given text container.
- `glyphRangeForBoundingRectWithoutAdditionalLayout:inTextContainer:` (page 46)
  Returns the smallest contiguous range for glyphs that are laid out wholly or partially within the given rectangle in the given text container.

- `glyphIndexForPoint:inTextContainer:fractionOfDistanceThroughGlyph:` (page 44)
  Returns the index of the glyph falling under the given point, expressed in the given container's coordinate system.
- `fractionOfDistanceThroughGlyphForPoint:inTextContainer:` (page 38)
  This method is a primitive for `glyphIndexForPoint:inTextContainer:fractionOfDistanceThroughGlyph:` (page 44). You should always call the main method, not the primitives.
- `glyphIndexForPoint:inTextContainer:` (page 44)
  This method is a primitive for `glyphIndexForPoint:inTextContainer:fractionOfDistanceThroughGlyph:` (page 44). You should always call the main method, not the primitives.

Handling Layout for Text Blocks

- `setLayoutRect:forTextBlock:glyphRange:` (page 75)
  Sets the layout rectangle enclosing the given text block containing the given glyph range.
- `layoutRectForTextBlock:glyphRange:` (page 57)
  Returns the layout rectangle within which the given text block containing the given glyph range is to be laid out.
- `setBoundsRect:forTextBlock:glyphRange:` (page 70)
  Sets the bounding rectangle enclosing a given text block containing the given glyph range.
- `boundsRectForTextBlock:glyphRange:` (page 25)
  Returns the bounding rectangle enclosing the given text block containing the given glyph range.
- `layoutRectForTextBlock:atIndex:effectiveRange:` (page 56)
  Returns the layout rectangle within which the given text block containing the glyph at the given index is to be laid out.
- `boundsRectForTextBlock:atIndex:effectiveRange:` (page 24)
  Returns the bounding rectangle within which the given text block containing the glyph at the given index is to be laid out.

Displaying Special Glyphs

- `setNotShownAttribute:forGlyphAtIndex:` (page 77)
  Sets the glyph at the given index to be one that isn't shown.
- `notShownAttributeForGlyphAtIndex:` (page 61)
  Indicates whether the glyph at the given index is one that isn't shown.
- `setShowsInvisibleCharacters:` (page 78)
  Specifies whether to substitute visible glyphs for whitespace and other typically invisible characters in layout.
- `showsInvisibleCharacters` (page 84)
  Indicates whether the receiver substitutes visible glyphs for whitespace and other typically invisible characters in layout.
- `setShowsControlCharacters:` (page 78)
  Specifies whether to substitute visible glyphs for control characters in layout.

- `showsControlCharacters` (page 83)
  Indicates whether the receiver substitutes visible glyphs for control characters.
- `layoutOptions` (page 55)
  Returns the layout manager's current layout options.

Controlling Hyphenation

- `setHyphenationFactor:` (page 73)
  Sets the threshold controlling when hyphenation is done.
- `hyphenationFactor` (page 48)
  Returns the current hyphenation threshold.

Finding Characters and Glyphs Not Laid Out

- `getFirstUnlaidCharacterIndex:glyphIndex:` (page 38)
  Returns the indexes for the first character and glyph that have invalid layout information.
- `firstUnlaidCharacterIndex` (page 37)
  Returns the index for the first character in the layout manager that has not been laid out.
- `firstUnlaidGlyphIndex` (page 38)
  Returns the index for the first glyph in the layout manager that has not been laid out.

Using Screen Fonts

- `setUsesScreenFonts:` (page 82)
  Controls using screen fonts to calculate layout and display text.
- `usesScreenFonts` (page 95)
  Indicates whether the receiver uses screen fonts to calculate layout and display text.
- `substituteFontForFont:` (page 85)
  Returns a screen font suitable for use in place of the given font, if one is available.

Handling Rulers

- `rulerAccessoryViewForTextView:paragraphStyle:ruler:enabled:` (page 67)
  Returns the the accessory view that the text system uses for its ruler.
- `rulerMarkersForTextView:paragraphStyle:ruler:` (page 68)
  Returns an array of text ruler objects for the current selection.

Managing the Responder Chain

- `layoutManagerOwnsFirstResponderInWindow:` (page 55)
  Indicates whether the first responder in the given window is a text view associated with the receiver.

`- firstTextView` (page 37)
    Returns the first text view in the receiver's series of text views.

`- textViewForBeginningOfSelection` (page 92)
    Returns the text view containing the first glyph in the selection.

Drawing

`- drawBackgroundForGlyphRange:atPoint:` (page 29)
    Draws background marks for the given glyph range, which must lie completely within a single text container.

`- drawGlyphsForGlyphRange:atPoint:` (page 30)
    Draws the glyphs in the given glyph range, which must lie completely within a single text container.

`- drawUnderlineForGlyphRange:underlineType:baselineOffset:lineFragmentRect:lineFragmentGlyphRange:containerOrigin:` (page 32)
    Draws underlining for the glyphs in a given range.

`- underlineGlyphRange:underlineType:lineFragmentRect:lineFragmentGlyphRange:containerOrigin:` (page 94)
    Calculates subranges to be underlined for the glyphs in a given range and draws the underlining as appropriate.

`- showPackedGlyphs:length:glyphRange:atPoint:font:color:printingAdjustment:` (page 83)
    Draws a range of glyphs.

`- drawStrikethroughForGlyphRange:strikethroughType:baselineOffset:lineFragmentRect:lineFragmentGlyphRange:containerOrigin:` (page 31)
    Draws a strikethrough for the glyphs in a given range.

`- strikethroughGlyphRange:strikethroughType:lineFragmentRect:lineFragmentGlyphRange:containerOrigin:` (page 84)
    Calculates and draws strikethrough for the glyphs in the given range.

Accessing the Delegate

`- setDelegate:` (page 71)
    Sets the receiver's delegate.

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

Accessing the Typesetter

`- setTypesetter:` (page 80)
    Sets the current typesetter.

`- typesetter` (page 93)
    Returns the receiver's typesetter.

Managing Typesetter Compatibility

`- setTypesetterBehavior:` (page 81)
: Sets the default typesetter behavior.

`- typesetterBehavior` (page 93)
: Returns the current typesetter behavior.

`- defaultLineHeightForFont:` (page 28)
: Returns the default line height for a line of text drawn using a given font.

`- defaultBaselineOffsetForFont:` (page 28)
: Returns the default baseline offset specified by the layout manager's typesetter behavior for the given font.

Managing Temporary Attribute Support

`- addTemporaryAttributes:forCharacterRange:` (page 21)
: Appends one or more temporary attributes to the attributes dictionary of the specified character range.

`- addTemporaryAttribute:value:forCharacterRange:` (page 20)
: Adds a temporary attribute with the given name and value to the characters in the specified range.

`- setTemporaryAttributes:forCharacterRange:` (page 79)
: Sets one or more temporary attributes for the specified character range.

`- removeTemporaryAttribute:forCharacterRange:` (page 64)
: Removes a temporary attribute from the list of attributes for the specified character range.

`- temporaryAttribute:atCharacterIndex:effectiveRange:` (page 85)
: Returns the value for the temporary attribute with a given name of the character at a given index, and by reference the range over which the attribute applies.

`- temporaryAttribute:atCharacterIndex:longestEffectiveRange:inRange:` (page 86)
: Returns the value for the temporary attribute with a given name of the character at a given index, and by reference the maximum range over which the attribute applies.

`- temporaryAttributesAtCharacterIndex:effectiveRange:` (page 87)
: Returns the dictionary of temporary attributes for the character range specified in *`effectiveCharRange`* at character index *`charIndex`*.

`- temporaryAttributesAtCharacterIndex:longestEffectiveRange:inRange:` (page 88)
: Returns the temporary attributes for the character at a given index, and by reference the maximum range over which the attributes apply.

`- layoutManager:shouldUseTemporaryAttributes:forDrawingToScreen:atCharacterIndex:effectiveRange:` (page 96) *delegate method*
: Sent when the layout manager is drawing and needs to decide whether or not to use temporary attributes.

Managing Noncontiguous Layout

`- setAllowsNonContiguousLayout:` (page 68)
: Enables or disables noncontiguous layout.

- `allowsNonContiguousLayout` (page 22)
  Indicates whether noncontiguous layout is enabled or disabled.
- `hasNonContiguousLayout` (page 48)
  Indicates whether the layout manager currently has any areas of noncontiguous layout.
- `ensureGlyphsForCharacterRange:` (page 33)
  Forces the receiver to generate glyphs for the specified character range, if it has not already done so.
- `ensureGlyphsForGlyphRange:` (page 34)
  Forces the receiver to generate glyphs for the specified glyph range, if it has not already done so.
- `ensureLayoutForCharacterRange:` (page 34)
  Forces the receiver to perform layout for the specified character range, if it has not already done so.
- `ensureLayoutForGlyphRange:` (page 35)
  Forces the receiver to perform layout for the specified glyph range, if it has not already done so.
- `ensureLayoutForTextContainer:` (page 35)
  Forces the receiver to perform layout for the specified text container, if it has not already done so.
- `ensureLayoutForBoundingRect:inTextContainer:` (page 34)
  Forces the receiver to perform layout for the specified area in the specified text container, if it has not already done so.

Accessing the Font Leading

- `usesFontLeading` (page 95)
  Indicates whether the receiver uses the leading provided in the font.
- `setUsesFontLeading:` (page 81)
  Specifies whether or not the receiver uses the leading provided in the font.

Instance Methods

addTemporaryAttribute:value:forCharacterRange:

Adds a temporary attribute with the given name and value to the characters in the specified range.

```
- (void)addTemporaryAttribute:(NSString *)attrName value:(id)value
    forCharacterRange:(NSRange)charRange
```

Parameters

attrName
  The name of a temporary attribute.

value
  The temporary attribute value associated with *attrName*.

charRange
  The range of characters to which the specified attribute-value pair applies.

Discussion

Raises an `NSInvalidArgumentException` if *attrName* or *value* is `nil`.

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

See Also
`– addTemporaryAttributes:forCharacterRange:` (page 21)
`– setTemporaryAttributes:forCharacterRange:` (page 79)
`– removeTemporaryAttribute:forCharacterRange:` (page 64)
`– temporaryAttributesAtCharacterIndex:effectiveRange:` (page 87)

Declared In
`NSLayoutManager.h`

addTemporaryAttributes:forCharacterRange:

Appends one or more temporary attributes to the attributes dictionary of the specified character range.

```
- (void)addTemporaryAttributes:(NSDictionary *)attrs
    forCharacterRange:(NSRange)charRange
```

Parameters

attrs
Attributes dictionary containing the temporary attributes to add.

charRange
The range of characters to which the specified attributes apply.

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 recognized are those that do not affect layout (colors, underlines, and so on).

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

See Also
`– setTemporaryAttributes:forCharacterRange:` (page 79)
`– removeTemporaryAttribute:forCharacterRange:` (page 64)
`– temporaryAttributesAtCharacterIndex:effectiveRange:` (page 87)

Related Sample Code
LayoutManagerDemo

Declared In
`NSLayoutManager.h`

addTextContainer:

Appends the given text container to the series of text containers where the receiver arranges text.

```
- (void)addTextContainer:(NSTextContainer *)aTextContainer
```

Parameters

`aTextContainer`
: The text container to append.

Discussion

Invalidates glyphs and layout as needed, but doesn't perform glyph generation or layout.

Availability

Available in Mac OS X v10.0 and later.

See Also

`– insertTextContainer:atIndex:` (page 50)
`– removeTextContainerAtIndex:` (page 65)
`– textContainers` (page 91)
`– invalidateGlyphsForCharacterRange:changeInLength:actualCharacterRange:` (page 52)
`– invalidateLayoutForCharacterRange:isSoft:actualCharacterRange:` (page 54)

Related Sample Code

DockTile
Sketch-112
SpeedometerView
TextLayoutDemo
TextViewConfig

Declared In

`NSLayoutManager.h`

allowsNonContiguousLayout

Indicates whether noncontiguous layout is enabled or disabled.

```
- (BOOL)allowsNonContiguousLayout
```

Return Value

`YES` if noncontiguous layout is enabled; otherwise, `NO`.

Discussion

For more information about noncontiguous layout, see "Noncontiguous Layout" (page 10).

Availability

Available in Mac OS X v10.5 and later.

See Also

`– setAllowsNonContiguousLayout:` (page 68)
`– hasNonContiguousLayout` (page 48)

Declared In

`NSLayoutManager.h`

attachmentSizeForGlyphAtIndex:

For a glyph corresponding to an attachment, returns the size for the attachment cell to occupy.

```
- (NSSize)attachmentSizeForGlyphAtIndex:(NSUInteger)glyphIndex
```

Parameters

glyphIndex

The index of the attachment glyph.

Return Value

The size for the attachment cell to occupy. Returns `{-1.0, -1.0}` if there is no attachment laid for the specified glyph.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- setAttachmentSize:forGlyphRange:` (page 69)
`- defaultAttachmentScaling` (page 27)

Declared In

`NSLayoutManager.h`

attributedString

Returns the text storage object from which the `NSGlyphGenerator` object procures characters for glyph generation.

```
- (NSAttributedString *)attributedString
```

Return Value

The receiver's text storage object.

Discussion

This method is part of the `NSGlyphStorage` protocol, for use by the glyph generator. For `NSLayoutManager` the attributed string is equivalent to the text storage.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`NSLayoutManager.h`

backgroundLayoutEnabled

Indicates whether the receiver generates glyphs and lays out text when the application's run loop is idle.

```
- (BOOL)backgroundLayoutEnabled
```

Return Value

`YES` if the receiver generates glyphs and lays out text when the application's run loop is idle, `NO` if it performs glyph generation and layout only when necessary.

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

See Also
`- setBackgroundLayoutEnabled:` (page 69)

Declared In
`NSLayoutManager.h`

boundingRectForGlyphRange:inTextContainer:

Returns a single bounding rectangle (in container coordinates) enclosing all glyphs and other marks drawn in the given text container for the given glyph range, including glyphs that draw outside their line fragment rectangles and text attributes such as underlining.

```
- (NSRect)boundingRectForGlyphRange:(NSRange)glyphRange
    inTextContainer:(NSTextContainer *)container
```

Parameters

glyphRange
The range of glyphs for which to return the bounding rectangle.

container
The text container in which the glyphs are laid out.

Return Value
The bounding rectangle enclosing the given range of glyphs.

Discussion
The range is intersected with the container's range before computing the bounding rectangle. This method can be used to translate glyph ranges into display rectangles for invalidation and redrawing when a range of glyphs changes. Bounding rectangles are always in container coordinates.

Performs glyph generation and layout if needed.

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

See Also
`- glyphRangeForBoundingRect:inTextContainer:` (page 45)
`- glyphRangeForTextContainer:` (page 47)
`- drawsOutsideLineFragmentForGlyphAtIndex:` (page 31)

Related Sample Code
LayoutManagerDemo

Declared In
`NSLayoutManager.h`

boundsRectForTextBlock:atIndex:effectiveRange:

Returns the bounding rectangle within which the given text block containing the glyph at the given index is to be laid out.

```
- (NSRect)boundsRectForTextBlock:(NSTextBlock *)block
    atIndex:(NSUInteger)glyphIndex
    effectiveRange:(NSRangePointer)effectiveGlyphRange
```

Parameters

block

The text block whose bounding rectangle is returned.

glyphIndex

Index of the glyph.

effectiveGlyphRange

If not `NULL`, on output, the range for all glyphs in the text block.

Return Value

The bounding rectangle of the text block, or `NSZeroRect` if no rectangle has been set for the specified block since the last invalidation.

Discussion

This method causes glyph generation but not layout. Block layout rectangles and bounds rectangles are always in container coordinates.

Availability

Available in Mac OS X v10.4 and later.

See Also

`- setBoundsRect:forTextBlock:glyphRange:` (page 70)

Declared In

`NSLayoutManager.h`

boundsRectForTextBlock:glyphRange:

Returns the bounding rectangle enclosing the given text block containing the given glyph range.

```
- (NSRect)boundsRectForTextBlock:(NSTextBlock *)block glyphRange:(NSRange)glyphRange
```

Parameters

block

The text block whose bounds rectangle is returned.

glyphRange

The range of glyphs in the text block.

Return Value

The bounding rectangle, or `NSZeroRect` if no rectangle has been set for the specified block since the last invalidation

Discussion

This method causes glyph generation but not layout. Block layout rectangles and bounds rectangles are always in container coordinates.

Availability

Available in Mac OS X v10.4 and later.

See Also

`- setBoundsRect:forTextBlock:glyphRange:` (page 70)

Declared In
`NSLayoutManager.h`

characterIndexForGlyphAtIndex:

Returns the index in the text storage for the first character associated with the given glyph.

```
- (NSUInteger)characterIndexForGlyphAtIndex:(NSUInteger)glyphIndex
```

Parameters

glyphIndex
The index of the glyph for which to return the associated character.

Return Value
The index of the first character associated with the glyph at the specified index.

Discussion
If noncontiguous layout is not enabled, this method causes generation of all glyphs up to and including *glyphIndex*. This method accepts an index beyond the last glyph, returning an index extrapolated from the last actual glyph index.

In many cases it's better to use the range-mapping methods, `characterRangeForGlyphRange:actualGlyphRange:` (page 26) and `glyphRangeForCharacterRange:actualCharacterRange:` (page 47), which provide more comprehensive information.

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

See Also
`- glyphIndexForCharacterAtIndex:` (page 43)

Related Sample Code
LayoutManagerDemo

Declared In
`NSLayoutManager.h`

characterRangeForGlyphRange:actualGlyphRange:

Returns the range of characters that generated the glyphs in the given glyph range.

```
- (NSRange)characterRangeForGlyphRange:(NSRange)glyphRange
    actualGlyphRange:(NSRangePointer)actualGlyphRange
```

Parameters

glyphRange
The glyph range for which to return the character range.

actualGlyphRange
: If not `NULL`, on output, points to the full range of glyphs generated by the character range returned. This range may be identical or slightly larger than the requested glyph range. For example, if the text storage contains the character "Ö" and the glyph cache contains the two atomic glyphs "O" and "¨", and if *glyphRange* encloses only the first or second glyph, then *actualGlyphRange* is set to enclose both glyphs.

Return Value
The range of characters that generated the glyphs in *glyphRange*.

Discussion
If the length of *glyphRange* is `0`, the resulting character range is a zero-length range just after the character(s) corresponding to the preceding glyph, and *actualGlyphRange* is also zero-length. If *glyphRange* extends beyond the text length, the method truncates the result to the number of characters in the text.

If noncontiguous layout is not enabled, this method forces the generation of glyphs for all characters up to and including the end of the returned range.

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

See Also
`– characterIndexForGlyphAtIndex:` (page 26)
`– glyphRangeForCharacterRange:actualCharacterRange:` (page 47)

Related Sample Code
LayoutManagerDemo
TipWrapper

Declared In
`NSLayoutManager.h`

defaultAttachmentScaling

Returns the default behavior desired if an attachment image is too large to fit in a text container.

```
- (NSImageScaling)defaultAttachmentScaling
```

Discussion
Attachment cells control their own size and drawing, so this setting is only advisory to them, but Application Kit–supplied attachment cells respect it.

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

See Also
`– setDefaultAttachmentScaling:` (page 71)

Declared In
`NSLayoutManager.h`

defaultBaselineOffsetForFont:

Returns the default baseline offset specified by the layout manager's typesetter behavior for the given font.

```
- (CGFloat)defaultBaselineOffsetForFont:(NSFont *)theFont
```

Parameters

theFont

The font for which to return the default baseline offset.

Return Value

The default baseline offset for a line of text drawn using *theFont*.

Discussion

The value returned may vary according to the layout manager's typesetter behavior.

Availability

Available in Mac OS X v10.5 and later.

See Also

`- setTypesetterBehavior:` (page 81)
`- defaultLineHeightForFont:` (page 28)

Declared In

`NSLayoutManager.h`

defaultLineHeightForFont:

Returns the default line height for a line of text drawn using a given font.

```
- (CGFloat)defaultLineHeightForFont:(NSFont *)theFont
```

Parameters

theFont

The font for which to determine the default line height.

Return Value

The default line height for a line of text drawn using *theFont*.

Discussion

The value returned may vary according to the layout manager's typesetter behavior.

Availability

Available in Mac OS X v10.2 and later.

See Also

`- setTypesetterBehavior:` (page 81)
`- defaultBaselineOffsetForFont:` (page 28)

Declared In

`NSLayoutManager.h`

delegate

Returns the receiver's delegate.

```
- (id)delegate
```

Return Value
The receiver's delegate.

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

See Also
`- setDelegate:` (page 71)

Declared In
`NSLayoutManager.h`

deleteGlyphsInRange:

Deletes the glyphs in the given range from the receiver's glyph store.

```
- (void)deleteGlyphsInRange:(NSRange)glyphRange
```

Parameters

glyphRange
    The range of glyphs to delete.

Discussion
This method is for use by the glyph-generation mechanism and doesn't perform any invalidation or generation of the glyphs or layout. This method should be invoked only during glyph generation and typesetting, in almost all cases only by the glyph generator or typesetter. For example, a custom glyph generator or typesetter might invoke it.

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

See Also
`- insertGlyph:atGlyphIndex:characterIndex:` (page 49)

Declared In
`NSLayoutManager.h`

drawBackgroundForGlyphRange:atPoint:

Draws background marks for the given glyph range, which must lie completely within a single text container.

```
- (void)drawBackgroundForGlyphRange:(NSRange)glyphsToShow atPoint:(NSPoint)origin
```

Parameters

glyphsToShow
    The range of glyphs for which the background is drawn.

origin
: The position of the text container in the coordinate system of the currently focused view.

Discussion

This method is called by `NSTextView` for drawing. You can override it to perform additional drawing, or to replace text drawing entirely, but not to change layout. You can call this method directly, but focus must already be locked on the destination view or image.

Background marks are such things as selection highlighting, text background color, and any background for marked text, along with block decoration such as table backgrounds and borders.

Performs glyph generation and layout if needed.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `drawGlyphsForGlyphRange:atPoint:` (page 30)
- `glyphRangeForTextContainer:` (page 47)
- `textContainerOrigin` (`NSTextView`)

Related Sample Code

Sketch-112

Declared In

`NSLayoutManager.h`

drawGlyphsForGlyphRange:atPoint:

Draws the glyphs in the given glyph range, which must lie completely within a single text container.

```
- (void)drawGlyphsForGlyphRange:(NSRange)glyphsToShow atPoint:(NSPoint)origin
```

Parameters

glyphsToShow
: The range of glyphs that are drawn.

origin
: The position of the text container in the coordinate system of the currently focused view.

Discussion

This method is called by `NSTextView` for drawing. You can override it to perform additional drawing, or to replace text drawing entirely, but not to change layout. You can call this method directly, but focus must already be locked on the destination view or image. This method expects the coordinate system of the view to be flipped.

This method draws the actual glyphs, including attachments, as well as any underlines or strikethoughs.

Performs glyph generation and layout if needed.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `drawBackgroundForGlyphRange:atPoint:` (page 29)

– `glyphRangeForTextContainer:` (page 47)
– `textContainerOrigin` (`NSTextView`)

Related Sample Code
DockTile
Sketch-112
SpeedometerView
WebKitPluginStarter
WebKitPluginWithJavaScript

Declared In
`NSLayoutManager.h`

drawsOutsideLineFragmentForGlyphAtIndex:

Indicates whether the glyph draws outside of its line fragment rectangle.

```
- (BOOL)drawsOutsideLineFragmentForGlyphAtIndex:(NSUInteger)glyphIndex
```

Parameters

`glyphIndex`
Index of the glyph.

Return Value
`YES` if the glyph at *`glyphIndex`* exceeds the bounds of the line fragment where it's laid out, `NO` otherwise.

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.

This method causes glyph generation and layout for the line fragment containing the specified glyph, or if noncontiguous layout is not enabled, up to and including that line fragment.

Glyphs that draw outside their line fragment rectangles aren't considered when calculating enclosing rectangles with the `rectArrayForCharacterRange:withinSelectedCharacterRange:inTextContainer:rectCount:` (page 62) and `rectArrayForGlyphRange:withinSelectedGlyphRange:inTextContainer:rectCount:` (page 63) methods. They are, however, considered by `boundingRectForGlyphRange:inTextContainer:` (page 24).

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

Declared In
`NSLayoutManager.h`

drawStrikethroughForGlyphRange:strikethroughType:baselineOffset:lineFragmentRect:lineFragmentGlyphRange:containerOrigin:

Draws a strikethrough for the glyphs in a given range.

```
- (void)drawStrikethroughForGlyphRange:(NSRange)glyphRange
    strikethroughType:(NSInteger)strikethroughVal
    baselineOffset:(CGFloat)baselineOffset lineFragmentRect:(NSRect)lineRect
    lineFragmentGlyphRange:(NSRange)lineGlyphRange
    containerOrigin:(NSPoint)containerOrigin
```

Parameters

`glyphRange`
: The range of glyphs for which to draw a strikethrough. The range must belong to a single line fragment rectangle (as returned by `lineFragmentRectForGlyphAtIndex:effectiveRange:` (page 57)).

`strikethroughVal`
: The style of strikethrough to draw. This value is a mask derived from the value for `NSUnderlineStyleAttributeName`—for example, `(NSUnderlinePatternDash | NSUnderlineStyleThick)`. Subclasses can define custom strikethrough styles.

`baselineOffset`
: Indicates how far above the text baseline the underline should be drawn.

`lineRect`
: The line fragment rectangle containing the glyphs to draw strikethrough for.

`lineGlyphRange`
: The range of all glyphs within *`lineRect`*.

`containerOrigin`
: The origin of the line fragment rectangle's `NSTextContainer` in its `NSTextView`.

Discussion

This method is invoked automatically by `strikethroughGlyphRange:strikethroughType:lineFragmentRect:lineFragmentGlyphRange:containerOrigin:` (page 84); you should rarely need to invoke it directly. This method's *`strikethroughVal`* parameter does not take account of any setting for`NSUnderlineByWordMask` because that's taken care of by `underlineGlyphRange:underlineType:lineFragmentRect:lineFragmentGlyphRange:containerOrigin:` (page 94).

Availability

Available in Mac OS X v10.3 and later.

Declared In

`NSLayoutManager.h`

drawUnderlineForGlyphRange:underlineType:baselineOffset:lineFragmentRect:lineFragmentGlyphRange:containerOrigin:

Draws underlining for the glyphs in a given range.

```
- (void)drawUnderlineForGlyphRange:(NSRange)glyphRange
    underlineType:(NSInteger)underlineVal baselineOffset:(CGFloat)baselineOffset
    lineFragmentRect:(NSRect)lineRect lineFragmentGlyphRange:(NSRange)lineGlyphRange
     containerOrigin:(NSPoint)containerOrigin
```

Parameters

glyphRange
: A range of glyphs, which must belong to a single line fragment rectangle (as returned by `lineFragmentRectForGlyphAtIndex:effectiveRange:` (page 57)).

underlineVal
: The style of underlining to draw. This value is a mask derived from the value for `NSUnderlineStyleAttributeName`—for example, `(NSUnderlinePatternDash | NSUnderlineStyleThick)`. Subclasses can define custom underlining styles.

baselineOffset
: Specifies the distance from the bottom of the bounding box of the specified glyphs in the specified range to their baseline.

lineRect
: The line fragment rectangle containing the glyphs to draw underlining for.

lineGlyphRange
: The range of all glyphs within *lineRect*.

containerOrigin
: The origin of the *lineRect* `NSTextContainer` in its `NSTextView`.

Discussion
This method is invoked automatically by `underlineGlyphRange:underlineType:lineFragmentRect:lineFragmentGlyphRange:containerOrigin:` (page 94); you should rarely need to invoke it directly. This method's *underlineVal* parameter does not take account of any setting for`NSUnderlineByWordMask` because that's taken care of by `underlineGlyphRange:underlineType:lineFragmentRect:lineFragmentGlyphRange:containerOrigin:` (page 94).

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

See Also
– `textContainerForGlyphAtIndex:effectiveRange:` (page 89)
– `textContainerOrigin` (`NSTextView`)

Declared In
`NSLayoutManager.h`

ensureGlyphsForCharacterRange:

Forces the receiver to generate glyphs for the specified character range, if it has not already done so.

```
- (void)ensureGlyphsForCharacterRange:(NSRange)charRange
```

Parameters

charRange
: The character range for which glyphs are generated.

Discussion
The layout manager reserves the right to perform glyph generation for larger ranges. If noncontiguous layout is disabled, then the affected range is always effectively extended to start at the beginning of the text.

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

Declared In
`NSLayoutManager.h`

ensureGlyphsForGlyphRange:

Forces the receiver to generate glyphs for the specified glyph range, if it has not already done so.

```
- (void)ensureGlyphsForGlyphRange:(NSRange)glyphRange
```

Parameters

glyphRange
  The glyph range for which glyphs are generated.

Discussion
The layout manager reserves the right to perform glyph generation for larger ranges. If noncontiguous layout is disabled, then the affected range is always effectively extended to start at the beginning of the text.

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

Declared In
`NSLayoutManager.h`

ensureLayoutForBoundingRect:inTextContainer:

Forces the receiver to perform layout for the specified area in the specified text container, if it has not already done so.

```
- (void)ensureLayoutForBoundingRect:(NSRect)bounds inTextContainer:(NSTextContainer
    *)container
```

Parameters

bounds
  The area for which layout is performed.

container
  The text container containing the area for which layout is performed.

Discussion
The layout manager reserves the right to perform layout for larger ranges. If noncontiguous layout is disabled, then the affected range is always effectively extended to start at the beginning of the text.

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

Declared In
`NSLayoutManager.h`

ensureLayoutForCharacterRange:

Forces the receiver to perform layout for the specified character range, if it has not already done so.

```
- (void)ensureLayoutForCharacterRange:(NSRange)charRange
```

Parameters

charRange
: The character range for which layout is performed.

Discussion

The layout manager reserves the right to perform layout for larger ranges. If noncontiguous layout is disabled, then the affected range is always effectively extended to start at the beginning of the text.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`NSLayoutManager.h`

ensureLayoutForGlyphRange:

Forces the receiver to perform layout for the specified glyph range, if it has not already done so.

```
- (void)ensureLayoutForGlyphRange:(NSRange)glyphRange
```

Parameters

glyphRange
: The glyph range for which layout is performed.

Discussion

The layout manager reserves the right to perform layout for larger ranges. If noncontiguous layout is disabled, then the affected range is always effectively extended to start at the beginning of the text.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`NSLayoutManager.h`

ensureLayoutForTextContainer:

Forces the receiver to perform layout for the specified text container, if it has not already done so.

```
- (void)ensureLayoutForTextContainer:(NSTextContainer *)container
```

Parameters

container
: The text container for which layout is performed.

Discussion

The layout manager reserves the right to perform layout for larger ranges. If noncontiguous layout is disabled, then the affected range is always effectively extended to start at the beginning of the text.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`NSLayoutManager.h`

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).

```
- (NSRect)extraLineFragmentRect
```

Return Value
The rectangle defining the extra line fragment for the insertion point.

Discussion
The rectangle is defined in the coordinate system of its `NSTextContainer`. Returns `NSZeroRect` if there is no such rectangle.

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

See Also
`- extraLineFragmentUsedRect` (page 36)
`- extraLineFragmentTextContainer` (page 36)
`- setExtraLineFragmentRect:usedRect:textContainer:` (page 72)

Declared In
`NSLayoutManager.h`

extraLineFragmentTextContainer

Returns the text container that contains the extra line fragment rectangle.

```
- (NSTextContainer *)extraLineFragmentTextContainer
```

Return Value
The text container that contains the extra line fragment rectangle, or `nil` if there is no extra line fragment rectangle.

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).

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

See Also
`- extraLineFragmentRect` (page 36)
`- extraLineFragmentUsedRect` (page 36)
`- setExtraLineFragmentRect:usedRect:textContainer:` (page 72)

Declared In
`NSLayoutManager.h`

extraLineFragmentUsedRect

Returns the rectangle enclosing the insertion point drawn in the extra line fragment rectangle.

```
- (NSRect)extraLineFragmentUsedRect
```

Return Value
The rectangle enclosing the insertion point.

Discussion
The rectangle is defined in the coordinate system of its `NSTextContainer`. Returns `NSZeroRect` if there is no extra line fragment rectangle.

The extra line fragment used rectangle is twice as wide (or tall) as the text container's line fragment padding, with the insertion point itself in the middle.

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

See Also
`- extraLineFragmentRect` (page 36)
`- extraLineFragmentTextContainer` (page 36)
`- setExtraLineFragmentRect:usedRect:textContainer:` (page 72)

Declared In
`NSLayoutManager.h`

firstTextView

Returns the first text view in the receiver's series of text views.

```
- (NSTextView *)firstTextView
```

Return Value
The receiver's first text view.

Discussion
This `NSTextView` object is the recipient of various `NSText` and `NSTextView` notifications.

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

Declared In
`NSLayoutManager.h`

firstUnlaidCharacterIndex

Returns the index for the first character in the layout manager that has not been laid out.

```
- (NSUInteger)firstUnlaidCharacterIndex
```

Return Value
The character index.

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

Declared In
`NSLayoutManager.h`

firstUnlaidGlyphIndex

Returns the index for the first glyph in the layout manager that has not been laid out.

```
- (NSUInteger)firstUnlaidGlyphIndex
```

Return Value
The glyph index.

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

Declared In
`NSLayoutManager.h`

fractionOfDistanceThroughGlyphForPoint:inTextContainer:

This method is a primitive for `glyphIndexForPoint:inTextContainer:fractionOfDistanceThroughGlyph:` (page 44). You should always call the main method, not the primitives.

```
- (CGFloat)fractionOfDistanceThroughGlyphForPoint:(NSPoint)point
    inTextContainer:(NSTextContainer *)container
```

Discussion
Overriding should be done for the primitive methods. Existing subclasses that do not do this overriding will not have their implementations available to Java developers.

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

See Also
`- glyphIndexForPoint:inTextContainer:` (page 44)

Declared In
`NSLayoutManager.h`

getFirstUnlaidCharacterIndex:glyphIndex:

Returns the indexes for the first character and glyph that have invalid layout information.

```
- (void)getFirstUnlaidCharacterIndex:(NSUInteger *)charIndex glyphIndex:(NSUInteger
     *)glyphIndex
```

Parameters

charIndex

On return, if not `NULL`, the index of the first character that has invalid layout information *glyphIndex*

On return, if not `NULL`, the index of the first glyph that has invalid layout information.

Discussion

Either parameter may be `NULL`, in which case the receiver simply ignores it.

As part of its implementation, this method calls `firstUnlaidCharacterIndex` (page 37) and `firstUnlaidGlyphIndex` (page 38). To change this method's behavior, override those two methods instead of this one.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSLayoutManager.h`

getGlyphs:range:

Fills the passed-in buffer with a sequence of glyphs

```
- (NSUInteger)getGlyphs:(NSGlyph *)glyphArray
    range:(NSRange)glyphRange
```

Parameters

glyphArray

On output, the displayable glyphs from *glyphRange*, null-terminated. Does not include in the result any `NSNullGlyph` or other glyphs that are not shown. The memory passed in should be large enough for at least `glyphRange.length+1` elements.

glyphRange

The range of glyphs from which to return the displayable glyphs.

Return Value

The actual number of glyphs filled into the array is returned (not counting the null-termination).

Discussion

Raises an `NSRangeException` if the range specified exceeds the bounds of the actual glyph range for the receiver. Performs glyph generation if needed.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `glyphAtIndex:` (page 42)
- `glyphAtIndex:isValidIndex:` (page 42)
- `notShownAttributeForGlyphAtIndex:` (page 61)

Declared In

`NSLayoutManager.h`

getGlyphsInRange:glyphs:characterIndexes:glyphInscriptions:elasticBits:

Returns the glyphs and information needed to perform layout for the given glyph range.

```
- (NSUInteger)getGlyphsInRange:(NSRange)glyphRange
    glyphs:(NSGlyph *)glyphBuffer
    characterIndexes:(NSUInteger *)charIndexBuffer
    glyphInscriptions:(NSGlyphInscription *)inscribeBuffer
    elasticBits:(BOOL *)elasticBuffer
```

Discussion
This is a convenience method for `getGlyphsInRange:glyphs:characterIndexes:glyphInscriptions:elasticBits:bidiLevels:` (page 40) that does not return a *bidiLevelBuffer*.

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

Declared In
`NSLayoutManager.h`

getGlyphsInRange:glyphs:characterIndexes:glyphInscriptions:elasticBits:bidiLevels:

Returns the glyphs and information needed to perform layout for the given glyph range.

```
- (NSUInteger)getGlyphsInRange:(NSRange)glyphRangeglyphs:(NSGlyph
    *)glyphBuffercharacterIndexes:(NSUInteger
    *)charIndexBufferglyphInscriptions:(NSGlyphInscription
    *)inscribeBufferelasticBits:(BOOL *)elasticBufferbidiLevels:(unsigned char
    *)bidiLevelBuffer
```

Parameters

glyphRange
: The range of glyphs to lay out.

glyphBuffer
: On output, the sequence of glyphs needed to lay out the given glyph range.

charIndexBuffer
: On output, the indexes of the original characters corresponding to the given glyph range. Note that a glyph at index 1 is not necessarily mapped to the character at index 1, since a glyph may be for a ligature or accent.

inscribeBuffer
: On output, the inscription attributes for each glyph, which are used to lay out characters that are combined together. The possible values are described in "Constants" (page 97).

elasticBuffer
: On output, values indicating whether a glyph is elastic for each glyph. An elastic glyph can be made longer at the end of a line or when needed for justification.

bidiLevelBuffer
: On output, the direction of each glyph for bidirectional text. The values range from 0 to 61 as defined by Unicode Standard Annex #9. An even value means the glyph goes left-to-right, and an odd value means the glyph goes right-to-left.

Return Value
The number of glyphs returned in *glyphBuffer*.

Discussion
This method and
`getGlyphsInRange:glyphs:characterIndexes:glyphInscriptions:elasticBits:` (page 39) are intended primarily to enable the typesetter to obtain in bulk the glyphs and other information that it needs to perform layout. These methods return all glyphs in the range, including `NSNullGlyph` and not-shown glyphs. They do not null-terminate the results. Each pointer passed in should either be `NULL`, or else point to sufficient memory to hold `glyphRange.length` elements.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`NSLayoutManager.h`

getLineFragmentInsertionPointsForCharacterAtIndex:alternatePositions:inDisplayOrder:positions:characterIndexes:

Returns insertion points in bulk for a given line fragment.

```
- (NSUInteger)getLineFragmentInsertionPointsForCharacterAtIndex:(NSUInteger)charIndex
    alternatePositions:(BOOL)aFlag inDisplayOrder:(BOOL)dFlag positions:(CGFloat
    *)positions characterIndexes:(NSUInteger *)charIndexes
```

Parameters

charIndex
The character index of one character within the line fragment.

aFlag
If `YES`, returns alternate, rather than primary, insertion points.

dFlag
If `YES`, returns insertion points in display, rather than logical, order.

positions
On output, the positions of the insertion points, in the order specified.

charIndexes
On output, the indexes of the characters corresponding to the returned insertion points.

Return Value
The number of insertion points returned.

Discussion
The method allows clients to obtain all insertion points for a line fragment in one call. Each pointer passed in should either be `NULL` or else point to sufficient memory to hold as many elements as there are insertion points in the line fragment (which cannot be more than the number of characters + 1). The returned positions indicate a transverse offset relative to the line fragment rectangle's origin. Internal caching is used to ensure that repeated calls to this method for the same line fragment (possibly with differing values for other arguments) are not significantly more expensive than a single call.

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

See Also
`- rectArrayForCharacterRange:withinSelectedCharacterRange:inTextContainer:rectCount:` (page 62)

`- rectArrayForGlyphRange:withinSelectedGlyphRange:inTextContainer:rectCount:` (page 63)

Declared In
`NSLayoutManager.h`

glyphAtIndex:

Returns the glyph at *`glyphIndex`*.

```
- (NSGlyph)glyphAtIndex:(NSUInteger)glyphIndex
```

Parameters

`glyphIndex`
The index of a glyph in the receiver. This value must not exceed the bounds of the receiver's glyph array.

Return Value
The glyph at *`glyphIndex`*.

Discussion
Raises an `NSRangeException` 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. Another method, `glyphAtIndex:isValidIndex:` (page 42), generates glyphs only up to the one requested, so using it can be more efficient.

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

See Also
`- getGlyphs:range:` (page 39)

Declared In
`NSLayoutManager.h`

glyphAtIndex:isValidIndex:

If the given index is valid, returns the glyph at that location and optionally returns a flag indicating whether the requested index is in range.

```
- (NSGlyph)glyphAtIndex:(NSUInteger)glyphIndex
    isValidIndex:(BOOL *)isValidIndex
```

Parameters

`glyphIndex`
The index of the glyph to be returned.

`isValidIndex`
If not `NULL`, on output, `YES` if the requested index is in range; otherwise `NO`.

Return Value
The glyph at the requested index, or `NSNullGlyph` if the requested index is out of the range `{0, numberOfGlyphs` (page 61)`}`.

Discussion
If noncontiguous layout is not enabled, this method causes generation of all glyphs up to and including *`glyphIndex`*.

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

See Also
`- getGlyphs:range:` (page 39)
`- glyphAtIndex:` (page 42)

Declared In
`NSLayoutManager.h`

glyphGenerator

Returns the glyph generator used by this layout manager.

```
- (NSGlyphGenerator *)glyphGenerator
```

Return Value
The glyph generator.

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

See Also
`- setGlyphGenerator:` (page 73)

Declared In
`NSLayoutManager.h`

glyphIndexForCharacterAtIndex:

Returns the index of the first glyph associated with the character at the specified index.

```
- (NSUInteger)glyphIndexForCharacterAtIndex:(NSUInteger)charIndex
```

Parameters

`charIndex`
The index of the character for which to return the associated glyph.

Return Value
The index of the first glyph associated with the character at the specified index.

Discussion
If noncontiguous layout is not enabled, this method causes generation of all glyphs up to and including those associated with the specified character. This method accepts an index beyond the last character, returning an index extrapolated from the last actual character index.

In many cases it's better to use the range-mapping methods, `characterRangeForGlyphRange:actualGlyphRange:` (page 26) and `glyphRangeForCharacterRange:actualCharacterRange:` (page 47), which provide more comprehensive information.

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

See Also
– `characterIndexForGlyphAtIndex:` (page 26)

Declared In
`NSLayoutManager.h`

glyphIndexForPoint:inTextContainer:

This method is a primitive for `glyphIndexForPoint:inTextContainer:fractionOfDistanceThroughGlyph:` (page 44). You should always call the main method, not the primitives.

```
- (NSUInteger)glyphIndexForPoint:(NSPoint)point inTextContainer:(NSTextContainer
    *)container
```

Discussion
Overriding should be done for the primitive methods. Existing subclasses that do not do this overriding will not have their implementations available to Java developers.

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

See Also
– `fractionOfDistanceThroughGlyphForPoint:inTextContainer:` (page 38)

Related Sample Code
LayoutManagerDemo

Declared In
`NSLayoutManager.h`

glyphIndexForPoint:inTextContainer:fractionOfDistanceThroughGlyph:

Returns the index of the glyph falling under the given point, expressed in the given container's coordinate system.

```
- (NSUInteger)glyphIndexForPoint:(NSPoint)point
    inTextContainer:(NSTextContainer *)container
    fractionOfDistanceThroughGlyph:(CGFloat *)partialFraction
```

Parameters

`point`
    The point for which to return the glyph, in coordinates of *`container`*.

container
: The container in which the returned glyph is laid out.

partialFraction
: If not `NULL`, on output, the fraction of the distance between the location of the glyph returned and the location of the next glyph.

Return Value
The index of the glyph falling under the given point, expressed in the given container's coordinate system.

Discussion
If no glyph is under *point*, the nearest glyph is returned, where nearest is defined according to the requirements of selection by mouse. Clients who wish to determine whether the the point actually lies within the bounds of the glyph returned should follow this with a call to `boundingRectForGlyphRange:inTextContainer:` (page 24) and test whether the point falls in the rectangle returned by that method. If *partialFraction* is non-NULL, it returns by reference the fraction of the distance between the location of the glyph returned and the location of the next glyph.

For purposes such as dragging out a selection or placing the insertion point, a partial percentage less than or equal to 0.5 indicates that *point* should be considered as falling before the glyph index returned; a partial percentage greater than 0.5 indicates that it should be considered as falling after the glyph index returned. If the nearest glyph doesn't lie under *point* at all (for example, if *point* is beyond the beginning or end of a line), this ratio is 0 or 1.

If the glyph stream contains the glyphs "A" and "b", with the width of "A" being 13 points, and the user clicks at a location 8 points into "A", *partialFraction* 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.

As part of its implementation, this method calls `fractionOfDistanceThroughGlyphForPoint:inTextContainer:` (page 38) and `glyphIndexForPoint:inTextContainer:` (page 44). To change this method's behavior, override those two methods instead of this one.

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

Declared In
`NSLayoutManager.h`

glyphRangeForBoundingRect:inTextContainer:

Returns the smallest contiguous range for glyphs that are laid out wholly or partially within the given rectangle in the given text container.

```
- (NSRange)glyphRangeForBoundingRect:(NSRect)bounds inTextContainer:(NSTextContainer
    *)container
```

Parameters

bounds
: The bounding rectangle for which to return glyphs.

`container`

The text container in which the glyphs are laid out.

Return Value

The range of glyphs that would need to be displayed in order to draw all glyphs that fall (even partially) within the given bounding rectangle. The range returned can include glyphs that don't fall inside or intersect *`bounds`*, although the first and last glyphs in the range always do. At most this method returns the glyph range for the whole container.

Discussion

This method is used to determine which glyphs need to be displayed within a given rectangle.

Performs glyph generation and layout if needed. Bounding rectangles are always in container coordinates.

Availability

Available in Mac OS X v10.0 and later.

See Also

`– glyphRangeForBoundingRectWithoutAdditionalLayout:inTextContainer:` (page 46)

Declared In

`NSLayoutManager.h`

glyphRangeForBoundingRectWithoutAdditionalLayout:inTextContainer:

Returns the smallest contiguous range for glyphs that are laid out wholly or partially within the given rectangle in the given text container.

```
- (NSRange)glyphRangeForBoundingRectWithoutAdditionalLayout:(NSRect)bounds
    inTextContainer:(NSTextContainer *)container
```

Parameters

`bounds`

The bounding rectangle for which to return glyphs.

`container`

The text container in which the glyphs are laid out.

Return Value

The range of glyphs that would need to be displayed in order to draw all glyphs that fall (even partially) within the given bounding rectangle. The range returned can include glyphs that don't fall inside or intersect *`bounds`*, although the first and last glyphs in the range always do. At most this method returns the glyph range for the whole container.

Discussion

Unlike `glyphRangeForBoundingRect:inTextContainer:` (page 45), this variant of the 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.

Bounding rectangles are always in container coordinates.

Availability

Available in Mac OS X v10.0 and later.

See Also

`– glyphRangeForBoundingRect:inTextContainer:` (page 45)

Declared In
`NSLayoutManager.h`

glyphRangeForCharacterRange:actualCharacterRange:

Returns the range of glyphs that are generated from the characters in the given character range.

```
- (NSRange)glyphRangeForCharacterRange:(NSRange)charRange
    actualCharacterRange:(NSRangePointer)actualCharRange
```

Parameters

charRange
: The character range for which to return the generated glyph range.

actualCharRange
: If not `NULL`, on output, points to the actual range of characters that fully define the glyph range returned. This range may be identical to or slightly larger than the requested character range. For example, if the text storage contains the characters "`O`" and "¨", and the glyph store contains the single precomposed glyph "`Ö`", and if *charRange* encloses only the first or second character, then *actualCharRange* is set to enclose both characters.

Return Value
The range of glyphs generated by *charRange*.

Discussion
If the length of *charRange* is `0`, the resulting glyph range is a zero-length range just after the glyph(s) corresponding to the preceding character, and *actualCharRange* will also be zero-length. If *charRange* extends beyond the text length, the method truncates the result to the number of glyphs in the text.

If noncontiguous layout is not enabled, this method forces the generation of glyphs for all characters up to and including the end of the specified range.

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

See Also
`- characterIndexForGlyphAtIndex:` (page 26)

Related Sample Code
TipWrapper

Declared In
`NSLayoutManager.h`

glyphRangeForTextContainer:

Returns the range of glyphs laid out within the given text container.

```
- (NSRange)glyphRangeForTextContainer:(NSTextContainer *)aTextContainer
```

Discussion
This is a less efficient method than the similar `textContainerForGlyphAtIndex:effectiveRange:` (page 89).

Performs glyph generation and layout if needed.

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

See Also
`- textContainerForGlyphAtIndex:effectiveRange:` (page 89)

Related Sample Code
DockTile
Sketch-112
SpeedometerView
WebKitPluginStarter
WebKitPluginWithJavaScript

Declared In
`NSLayoutManager.h`

hasNonContiguousLayout

Indicates whether the layout manager currently has any areas of noncontiguous layout.

```
- (BOOL)hasNonContiguousLayout
```

Return Value
`YES` if noncontiguous layout exists; otherwise, `NO`.

Discussion
There may be times at which there is no noncontiguous layout, such as when layout is complete; this method enables the layout manager to report that to clients.

For more information about noncontiguous layout, see "Noncontiguous Layout" (page 10).

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

See Also
`- allowsNonContiguousLayout` (page 22)
`- setAllowsNonContiguousLayout:` (page 68)

Declared In
`NSLayoutManager.h`

hyphenationFactor

Returns the current hyphenation threshold.

```
- (float)hyphenationFactor
```

Return Value
The hyphenation factor ranging 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.

Discussion
Whenever (width of the real contents of the line) / (the line fragment width) is less than `hyphenationFactor`, hyphenation is attempted when laying out the line. Hyphenation slows down text layout and increases memory usage, so it should be used sparingly.

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

See Also
`- setHyphenationFactor:` (page 73)

Declared In
`NSLayoutManager.h`

init

Initializes the receiver, a newly created `NSLayoutManager` object.

```
- (id)init
```

Discussion
This method is the designated initializer for the `NSLayoutManager` class. Returns an initialized object.

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

See Also
`- addLayoutManager:` (NSTextStorage)
`- addTextContainer:` (page 21)

Declared In
`NSLayoutManager.h`

insertGlyph:atGlyphIndex:characterIndex:

Inserts a single glyph into the glyph stream at the given index and maps it to the character at the given character index.

```
- (void)insertGlyph:(NSGlyph)glyph
    atGlyphIndex:(NSUInteger)glyphIndex
    characterIndex:(NSUInteger)charIndex
```

Parameters
glyph
The glyph to insert.

glyphIndex
The index at which to insert the glyph.

charIndex
The index of the character to which the glyph is mapped.

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. This method should be invoked only during glyph generation and typesetting, in almost all cases only by the glyph generator or typesetter. For example, a custom glyph generator or typesetter might invoke it.

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

See Also
– `deleteGlyphsInRange:` (page 29)
– `replaceGlyphAtIndex:withGlyph:` (page 66)

Declared In
`NSLayoutManager.h`

insertGlyphs:length:forStartingGlyphAtIndex:characterIndex:

Inserts the given glyphs into the glyph cache at the given index and maps them to characters beginning at the given character index.

```
- (void)insertGlyphs:(const NSGlyph *)glyphs length:(NSUInteger)length
    forStartingGlyphAtIndex:(NSUInteger)glyphIndex
    characterIndex:(NSUInteger)charIndex
```

Parameters

glyphs
The glyphs to insert.

glyphIndex
The index in the glyph cache to begin inserting glyphs.

length
The number of glyphs to insert.

charIndex
Index of first character to be mapped.

Discussion
This method is part of the `NSGlyphStorage` protocol, for use by the glyph generator. It enables bulk insertion of glyphs into the glyph cache.

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

Declared In
`NSLayoutManager.h`

insertTextContainer:atIndex:

Inserts the given text container into the series of text containers at the given index.

```
- (void)insertTextContainer:(NSTextContainer *)aTextContainer
    atIndex:(NSUInteger)index
```

Parameters

aTextContainer
: The text container to insert.

index
: The index in the series of text containers at which to insert *aTextContainer*.

Discussion

This method invalidates layout for all subsequent `NSTextContainer` objects, and invalidates glyph information as needed.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `addTextContainer:` (page 21)
- `removeTextContainerAtIndex:` (page 65)
- `textContainers` (page 91)

Declared In

`NSLayoutManager.h`

intAttribute:forGlyphAtIndex:

Returns the value of the attribute identified by the given attribute tag for the glyph at the given index.

```
- (NSInteger)intAttribute:(NSInteger)attributeTag
    forGlyphAtIndex:(NSUInteger)glyphIndex
```

Parameters

attributeTag
: The attribute whose value is returned.

glyphIndex
: Index of the glyph whose attribute value is returned.

Return Value

The value of the attribute identified by *attributeTag* and *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 `setIntAttribute:value:forGlyphAtIndex:` (page 74).

If noncontiguous layout is not enabled, this method causes generation of all glyphs up to and including *glyphIndex*. This method is primarily for the use of the glyph generator and typesetter.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `setIntAttribute:value:forGlyphAtIndex:` (page 74)

Declared In

`NSLayoutManager.h`

invalidateDisplayForCharacterRange:

Invalidates display for the given character range.

```
- (void)invalidateDisplayForCharacterRange:(NSRange)charRange
```

Parameters

charRange

The character range for which display is invalidated.

Discussion

Parts of the range that are not laid out are remembered and redisplayed later when the layout is available. Does not actually cause layout.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSLayoutManager.h`

invalidateDisplayForGlyphRange:

Marks the glyphs in the given glyph range as needing display, as well as the appropriate regions of the `NSTextView` objects that display those glyphs (using the `NSView` method `setNeedsDisplayInRect:`).

```
- (void)invalidateDisplayForGlyphRange:(NSRange)glyphRange
```

Parameters

glyphRange

The range of glyphs to invalidate.

Discussion

You should rarely need to invoke this method.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSLayoutManager.h`

invalidateGlyphsForCharacterRange:changeInLength:actualCharacterRange:

Invalidates the cached glyphs for the characters in the given character range, adjusts the character indices of all the subsequent glyphs by the change in length, and invalidates the new character range.

```
- (void)invalidateGlyphsForCharacterRange:(NSRange)charRange
    changeInLength:(NSInteger)lengthChange
    actualCharacterRange:(NSRangePointer)actualCharRange
```

Parameters

charRange

The range of characters for which to invalidate glyphs.

lengthChange

The number of characters added or removed.

actualCharRange

If not `NULL`, on output, the actual range invalidated after any necessary expansion. This range can be larger than the range of characters given due to the effect of context on glyphs and layout.

Discussion

This method 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:actualCharacterRange:` (page 53), passing *charRange* as the first argument.

This method is used by the layout mechanism and should be invoked only during typesetting, in almost all cases only by the typesetter. For example, a custom typesetter might invoke it.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSLayoutManager.h`

invalidateGlyphsOnLayoutInvalidationForGlyphRange:

Specifies explicitly when portions of the glyph stream depend on layout.

```
- (void)invalidateGlyphsOnLayoutInvalidationForGlyphRange:(NSRange)glyphRange
```

Parameters

glyphRange

The range of glyphs to invalidate.

Discussion

This method is for the use of the typesetter, to allow it to specify explicitly when portions of the glyph stream depend on layout, for example, because they have had hyphens inserted. Therefore, the glyphs are invalidated the next time their layout is invalidated, so that they will be regenerated before being laid out again.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`NSLayoutManager.h`

invalidateLayoutForCharacterRange:actualCharacterRange:

Invalidates the layout information for the glyphs mapped to the given range of characters.

```
- (void)invalidateLayoutForCharacterRange:(NSRange)charRange
    actualCharacterRange:(NSRangePointer)actualCharRange
```

Parameters

charRange

The range of characters to invalidate.

actualCharRange

If not `NULL`, on output, the actual range invalidated after any necessary expansion.

Discussion

This method has the same effect as `invalidateLayoutForCharacterRange:isSoft:actualCharacterRange:` (page 54) with *`flag`* set to `NO`.

This method only invalidates information; it performs no glyph generation or layout. You should rarely need to invoke this method.

Availability

Available in Mac OS X v10.5 and later.

See Also

`- invalidateGlyphsForCharacterRange:changeInLength:actualCharacterRange:` (page 52)

Declared In

`NSLayoutManager.h`

invalidateLayoutForCharacterRange:isSoft:actualCharacterRange:

Invalidates the layout information for the glyphs mapped to the given range of characters.

```
- (void)invalidateLayoutForCharacterRange:(NSRange)charRange isSoft:(BOOL)flag
    actualCharacterRange:(NSRangePointer)actualCharRange
```

Parameters

`charRange`
: The character range for which glyphs are invalidated.

`flag`
: If `YES`, invalidates internal caches in the layout manager; if `NO`, invalidates layout. See the discussion section.

`actualCharRange`
: If not `NULL`, on output, the range of characters mapped to the glyphs whose layout information is invalidated. This range can be larger than the range of characters given due to the effect of context on glyphs and layout.

Discussion

This method only invalidates information; it performs no glyph generation or layout. You should rarely need to invoke this method.

For code that needs to work on both Mac OS X v10.5 and previous releases, the following procedures should be used. For Mac OS X v10.4 and before, invalidation should consist of

1. Calling this method with the *`flag`* set to `YES`, for the range that has actually become invalid.

2. Calling this method with the *`flag`* set to `NO`, for the range (if any) that follows that range, usually extending to the end of the text, that might need to be moved due to relayout of the invalidated range.

As of Mac OS X v10.5, the semantics of the *`flag`* parameter are slightly different. Soft layout holes are obsolete in Mac OS X v10.5 and later, so the flag is no longer necessary. If the method is called with *`flag`* set to `NO`, then it has the effect of invalidating layout. If it's called with the *`flag`* set to `YES`, then it does not actually invalidate layout; it invalidates a number of internal caches, but otherwise has no effect, and in general is unnecessary.

This method is superseded by `invalidateLayoutForCharacterRange:actualCharacterRange:` (page 53) and will be deprecated in a future release.

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

See Also
`- invalidateGlyphsForCharacterRange:changeInLength:actualCharacterRange:` (page 52)

Declared In
`NSLayoutManager.h`

isValidGlyphIndex:

Indicates whether the specified index refers to a valid glyph, otherwise `NO`.

```
- (BOOL)isValidGlyphIndex:(NSUInteger)glyphIndex
```

Parameters

glyphIndex
The index of a glyph in the receiver.

Return Value
`YES` if the specified *glyphIndex* refers to a valid glyph, otherwise `NO`.

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

Declared In
`NSLayoutManager.h`

layoutManagerOwnsFirstResponderInWindow:

Indicates whether the first responder in the given window is a text view associated with the receiver.

```
- (BOOL)layoutManagerOwnsFirstResponderInWindow:(NSWindow *)window
```

Parameters

window
The window whose first responder is tested.

Return Value
`YES` if the first responder in *window* is a text view associated with the receiver; otherwise, `NO`.

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

Declared In
`NSLayoutManager.h`

layoutOptions

Returns the layout manager's current layout options.

```
- (NSUInteger)layoutOptions
```

Return Value
A bit mask representing the current layout options as defined in `Layout_Options` in *NSGlyphStorage Protocol Reference*.

Discussion
This method is part of the `NSGlyphStorage` protocol, for use by the glyph generator. It enables the glyph generator to ask which options the layout manager requests.

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

Declared In
`NSLayoutManager.h`

layoutRectForTextBlock:atIndex:effectiveRange:

Returns the layout rectangle within which the given text block containing the glyph at the given index is to be laid out.

```
- (NSRect)layoutRectForTextBlock:(NSTextBlock *)block
    atIndex:(NSUInteger)glyphIndex
    effectiveRange:(NSRangePointer)effectiveGlyphRange
```

Parameters

block
The text block whose layout rectangle is returned.

glyphIndex
Index of the glyph.

effectiveGlyphRange
If not `NULL`, on output, the range for all glyphs in the text block.

Return Value
The layout rectangle of the text block, or `NSZeroRect` if no rectangle has been set for the specified block since the last invalidation.

Discussion
This method causes glyph generation but not layout. Block layout rectangles and bounds rectangles are always in container coordinates.

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

See Also
`- setLayoutRect:forTextBlock:glyphRange:` (page 75)

Declared In
`NSLayoutManager.h`

layoutRectForTextBlock:glyphRange:

Returns the layout rectangle within which the given text block containing the given glyph range is to be laid out.

```
- (NSRect)layoutRectForTextBlock:(NSTextBlock *)block glyphRange:(NSRange)glyphRange
```

Return Value
The layout rectangle, or `NSZeroRect` if no rectangle has been set for the specified block since the last invalidation.

Discussion
This method causes glyph generation but not layout. Block layout rectangles and bounds rectangles are always in container coordinates.

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

See Also
`- setLayoutRect:forTextBlock:glyphRange:` (page 75)

Declared In
`NSLayoutManager.h`

lineFragmentRectForGlyphAtIndex:effectiveRange:

Returns the rectangle for the line fragment in which the given glyph is laid out and (optionally), by reference, the whole range of glyphs that are in that fragment.

```
- (NSRect)lineFragmentRectForGlyphAtIndex:(NSUInteger)glyphIndex
    effectiveRange:(NSRangePointer)effectiveGlyphRange
```

Parameters

glyphIndex
The glyph for which to return the line fragment rectangle.

effectiveGlyphRange
If not `NULL`, on output, the range for all glyphs in the line fragment.

Return Value
The line fragment in which the given glyph is laid out.

Discussion
This method causes glyph generation and layout for the line fragment containing the specified glyph, or if noncontiguous layout is not enabled, for all of the text up to and including that line fragment.

Line fragment rectangles are always in container coordinates.

Overriding this method is not recommended. If the the line fragment rectangle needs to be modified, that should be done at the typesetter level or by calling `setLineFragmentRect:forGlyphRange:usedRect:` (page 75).

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

See Also

`- lineFragmentUsedRectForGlyphAtIndex:effectiveRange:` (page 59)

`- setLineFragmentRect:forGlyphRange:usedRect:` (page 75)

Declared In

`NSLayoutManager.h`

lineFragmentRectForGlyphAtIndex:effectiveRange:withoutAdditionalLayout:

Returns the line fragment rectangle containing the glyph at the given glyph index.

```
- (NSRect)lineFragmentRectForGlyphAtIndex:(NSUInteger)glyphIndex
    effectiveRange:(NSRangePointer)effectiveGlyphRange
    withoutAdditionalLayout:(BOOL)flag
```

Parameters

glyphIndex
: The glyph for which to return the line fragment rectangle.

effectiveGlyphRange
: If not `NULL`, on output, the range for all glyphs in the line fragment.

flag
: If `YES`, glyph generation and layout are not performed, so this option should not be used unless layout is known to be complete for the range in question, or unless noncontiguous layout is enabled; if `NO`, both are performed as needed.

Return Value

The line fragment in which the given glyph is laid out.

Discussion

This method is primarily for use from within `NSTypesetter`, after layout is complete for the range in question, but before the layout manager's call to `NSTypesetter` has returned. In that case glyph and layout holes have not yet been recalculated, so the layout manager does not yet know that layout is complete for that range, and this variant must be used.

Overriding this method is not recommended. If the the line fragment rectangle needs to be modified, that should be done at the typesetter level or by calling `setLineFragmentRect:forGlyphRange:usedRect:` (page 75).

Availability

Available in Mac OS X v10.4 and later.

See Also

`- setLineFragmentRect:forGlyphRange:usedRect:` (page 75)

`- lineFragmentUsedRectForGlyphAtIndex:effectiveRange:withoutAdditionalLayout:` (page 59)

Declared In

`NSLayoutManager.h`

lineFragmentUsedRectForGlyphAtIndex:effectiveRange:

Returns the usage rectangle for the line fragment in which the given glyph is laid and (optionally) by reference the whole range of glyphs that are in that fragment.

```
- (NSRect)lineFragmentUsedRectForGlyphAtIndex:(NSUInteger)glyphIndex
    effectiveRange:(NSRangePointer)effectiveGlyphRange
```

Parameters

glyphIndex

The glyph for which to return the line fragment used rectangle.

effectiveGlyphRange

If not `NULL`, on output, the range for all glyphs in the line fragment.

Return Value

The used rectangle for the line fragment in which the given glyph is laid out.

Discussion

This method causes glyph generation and layout for the line fragment containing the specified glyph, or if noncontiguous layout is not enabled, up to and including that line fragment.

Line fragment used rectangles are always in container coordinates.

Overriding this method is not recommended. If the the line fragment used rectangle needs to be modified, that should be done at the typesetter level or by calling `setLineFragmentRect:forGlyphRange:usedRect:` (page 75).

Availability

Available in Mac OS X v10.0 and later.

See Also

`- lineFragmentRectForGlyphAtIndex:effectiveRange:` (page 57)
`- setLineFragmentRect:forGlyphRange:usedRect:` (page 75)

Declared In

`NSLayoutManager.h`

lineFragmentUsedRectForGlyphAtIndex:effectiveRange:withoutAdditionalLayout:

Returns the usage rectangle for the line fragment in which the given glyph is laid and (optionally) by reference the whole range of glyphs that are in that fragment.

```
- (NSRect)lineFragmentUsedRectForGlyphAtIndex:(NSUInteger)glyphIndex
    effectiveRange:(NSRangePointer)effectiveGlyphRange
    withoutAdditionalLayout:(BOOL)flag
```

Parameters

glyphIndex

The glyph for which to return the line fragment used rectangle.

effectiveGlyphRange

If not `NULL`, on output, the range for all glyphs in the line fragment.

flag

If `YES`, glyph generation and layout are not performed, so this option should not be used unless layout is known to be complete for the range in question, or unless noncontiguous layout is enabled; if `NO`, both are performed as needed.

Return Value

The used rectangle for the line fragment in which the given glyph is laid out.

Discussion

This method causes glyph generation and layout for the line fragment containing the specified glyph, or if noncontiguous layout is not enabled, up to and including that line fragment.

Line fragment used rectangles are always in container coordinates.

Overriding this method is not recommended. If the the line fragment used rectangle needs to be modified, that should be done at the typesetter level or by calling `setLineFragmentRect:forGlyphRange:usedRect:` (page 75).

Availability

Available in Mac OS X v10.4 and later.

See Also

`- setLineFragmentRect:forGlyphRange:usedRect:` (page 75)
`- lineFragmentRectForGlyphAtIndex:effectiveRange:withoutAdditionalLayout:` (page 58)

Declared In

`NSLayoutManager.h`

locationForGlyphAtIndex:

Returns the location for the given glyph within its line fragment.

```
- (NSPoint)locationForGlyphAtIndex:(NSUInteger)glyphIndex
```

Parameters

glyphIndex

The glyph whose location is returned.

Return Value

The location of the given glyph.

Discussion

If the given glyph does not have an explicit location set for it (for example, if it is part of (but not first in) a sequence of nominally spaced characters), the location is calculated by glyph advancements from the location of the most recent preceding glyph with a location set.

Glyph locations are relative to their line fragment rectangle's origin. The line fragment rectangle in turn is defined in the coordinate system of the text container where it resides.

This method causes glyph generation and layout for the line fragment containing the specified glyph, or if noncontiguous layout is not enabled, up to and including that line fragment.

Availability

Available in Mac OS X v10.0 and later.

See Also
`- lineFragmentRectForGlyphAtIndex:effectiveRange:` (page 57)
`- lineFragmentUsedRectForGlyphAtIndex:effectiveRange:` (page 59)

Declared In
`NSLayoutManager.h`

notShownAttributeForGlyphAtIndex:

Indicates whether the glyph at the given index is one that isn't shown.

```
- (BOOL)notShownAttributeForGlyphAtIndex:(NSUInteger)glyphIndex
```

Parameters

`glyphIndex`
Index of the glyph.

Return Value
`YES` if the glyph at *`glyphIndex`* is not shown; otherwise `NO`.

Discussion
Some glyphs are not shown. For example, a tab, newline, or attachment glyph is not 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, although they leave no visible marks.

This method causes glyph generation and layout for the line fragment containing the specified glyph, or if noncontiguous layout is not enabled, up to and including that line fragment.

Raises an `NSRangeException` if *`glyphIndex`* is out of bounds.

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

See Also
`- setNotShownAttribute:forGlyphAtIndex:` (page 77)

Declared In
`NSLayoutManager.h`

numberOfGlyphs

Returns the number of glyphs in the receiver.

```
- (NSUInteger)numberOfGlyphs
```

Return Value
The number of glyphs.

Discussion
If noncontiguous layout is not enabled, this method forces generation of glyphs for all characters.

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

Declared In
`NSLayoutManager.h`

rangeOfNominallySpacedGlyphsContainingIndex:

Returns the range for the glyphs around the given glyph that can be displayed using only their advancements from the font, without pairwise kerning or other adjustments to spacing.

```
- (NSRange)rangeOfNominallySpacedGlyphsContainingIndex:(NSUInteger)glyphIndex
```

Parameters

glyphIndex
Index of the glyph to test.

Return Value
The range of nominally spaced glyphs.

Discussion
The range returned begins with the first glyph, counting back from *glyphIndex*, that has a location set, and it continues up to, but does not include, the next glyph that has a location set.

Performs glyph generation and layout if needed.

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

Declared In
`NSLayoutManager.h`

rectArrayForCharacterRange:withinSelectedCharacterRange:inTextContainer:rectCount:

Returns an array of rectangles and, by reference, the number of such rectangles, that define the region in the given container enclosing the given character range.

```
- (NSRectArray)rectArrayForCharacterRange:(NSRange)charRange
    withinSelectedCharacterRange:(NSRange)selCharRange
    inTextContainer:(NSTextContainer *)container
    rectCount:(NSUInteger *)rectCount
```

Parameters

charRange
The character range for which to return rectangles.

selCharRange
Selected characters within *charRange*, which can affect the size of the rectangles; it must be equal to or contain *charRange*. If the caller is interested in this more from an enclosing point of view rather than a selection point of view, pass `{NSNotFound, 0}` as the selected range.

container
The text container in which the text is laid out.

rectCount
The number of rectangles returned.

Return Value
The array of rectangles enclosing the given range.

Discussion
These rectangles can be used to draw the text background or highlight for the given range of characters. If a selected range is given in *`selCharRange`*, the rectangles returned are correct for drawing the selection. Selection rectangles are generally more complicated than enclosing rectangles and supplying a selected range is the clue this method uses to determine whether to go to the trouble of doing this special work.

This method will do the minimum amount of work required to answer the question. The resulting array is owned by the layout manager and will be reused when this method, `rectArrayForGlyphRange:withinSelectedGlyphRange:inTextContainer:rectCount:` (page 63), or `boundingRectForGlyphRange:inTextContainer:` (page 24) is called. One of these methods may be called indirectly. If you aren't going to use the rectangles right away, you should copy them to another location. These rectangles are always in container coordinates.

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.

These rectangles don’t necessarily enclose glyphs that draw outside their line fragment rectangles; use `boundingRectForGlyphRange:inTextContainer:` (page 24) to determine the area that contains all drawing performed for a range of glyphs.

Performs glyph generation and layout if needed.

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

See Also
`– glyphRangeForTextContainer:` (page 47)
`– characterRangeForGlyphRange:actualGlyphRange:` (page 26)
`– drawsOutsideLineFragmentForGlyphAtIndex:` (page 31)

Declared In
`NSLayoutManager.h`

rectArrayForGlyphRange:withinSelectedGlyphRange:inTextContainer:rectCount:

Returns an array of rectangles and, by reference, the number of such rectangles, that define the region in the given container enclosing the given glyph range.

```
- (NSRectArray)rectArrayForGlyphRange:(NSRange)glyphRange
    withinSelectedGlyphRange:(NSRange)selGlyphRange
    inTextContainer:(NSTextContainer *)container
    rectCount:(NSUInteger *)rectCount
```

Parameters

`glyphRange`
The glyph range for which to return rectangles.

`selGlyphRange`
: Selected glyphs within `glyphRange`, which can affect the size of the rectangles; it must be equal to or contain `glyphRange`. If the caller is interested in this more from an enclosing point of view rather than a selection point of view, pass `{NSNotFound, 0}` as the selected range.

`container`
: The text container in which the text is laid out.

`rectCount`
: The number of rectangles returned.

Return Value
The array of rectangles enclosing the given range.

Discussion
These rectangles can be used to draw the text background or highlight for the given range of characters. If a selected range is given in `selGlyphRange`, the rectangles returned are correct for drawing the selection. Selection rectangles are generally more complicated than enclosing rectangles and supplying a selected range is the clue this method uses to determine whether to go to the trouble of doing this special work.

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.

This method will do the minimum amount of work required to answer the question. The resulting array is owned by the layout manager and will be reused when this method, `rectArrayForCharacterRange:withinSelectedCharacterRange:inTextContainer:rectCount:` (page 62), or `boundingRectForGlyphRange:inTextContainer:` (page 24) is called. One of these methods may be called indirectly. If you aren't going to use the rectangles right away, you should copy them to another location. These rectangles are always in container coordinates.

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:inTextContainer:` (page 24) to determine the area that contains all drawing performed for a range of glyphs.

Performs glyph generation and layout if needed.

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

See Also
- `glyphRangeForTextContainer:` (page 47)
- `glyphRangeForCharacterRange:actualCharacterRange:` (page 47)
- `drawsOutsideLineFragmentForGlyphAtIndex:` (page 31)

Declared In
`NSLayoutManager.h`

removeTemporaryAttribute:forCharacterRange:

Removes a temporary attribute from the list of attributes for the specified character range.

```
- (void)removeTemporaryAttribute:(NSString *)attrName
    forCharacterRange:(NSRange)charRange
```

Parameters

`attrName`
The name of a temporary attribute.

`charRange`
The range of characters from which to remove the specified temporary attribute.

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 recognized are those that do not affect layout (colors, underlines, and so on).

Availability

Available in Mac OS X v10.0 and later.

See Also

`– setTemporaryAttributes:forCharacterRange:` (page 79)
`– addTemporaryAttributes:forCharacterRange:` (page 21)
`– temporaryAttributesAtCharacterIndex:effectiveRange:` (page 87)

Related Sample Code

LayoutManagerDemo

Declared In

`NSLayoutManager.h`

removeTextContainerAtIndex:

Removes the text container at the given index and invalidates the layout as needed.

```
- (void)removeTextContainerAtIndex:(NSUInteger)index
```

Parameters

`index`
The index of the text container to remove.

Discussion

This method invalidates glyph information as needed.

Availability

Available in Mac OS X v10.0 and later.

See Also

`– addTextContainer:` (page 21)
`– insertTextContainer:atIndex:` (page 50)
`– textContainers` (page 91)
`– invalidateGlyphsForCharacterRange:changeInLength:actualCharacterRange:` (page 52)
`– invalidateLayoutForCharacterRange:isSoft:actualCharacterRange:` (page 54)

Related Sample Code

Quartz Composer WWDC 2005 TextEdit
TextEditPlus

Declared In
`NSLayoutManager.h`

replaceGlyphAtIndex:withGlyph:

Replaces the glyph at the given index with a new glyph.

```
- (void)replaceGlyphAtIndex:(NSUInteger)glyphIndex
    withGlyph:(NSGlyph)newGlyph
```

Parameters

`glyphIndex`
: Index of the glyph to replace.

`newGlyph`
: The new glyph.

Discussion
Doesn't alter the glyph-to-character mapping or invalidate layout information. The character index of the glyph is assumed to remain the same (although it can, of course, be set explicitly if needed).

This method is for use by the glyph-generation mechanism and doesn't perform any invalidation or generation of the glyphs or layout. This method should be invoked only during glyph generation and typesetting, in almost all cases only by the glyph generator or typesetter. For example, a custom glyph generator or typesetter might invoke it.

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

See Also
- `setCharacterIndex:forGlyphAtIndex:` (page 70)
- `invalidateGlyphsForCharacterRange:changeInLength:actualCharacterRange:` (page 52)
- `invalidateLayoutForCharacterRange:isSoft:actualCharacterRange:` (page 54)

Declared In
`NSLayoutManager.h`

replaceTextStorage:

Replaces the `NSTextStorage` object for the group of text-system objects containing the receiver with the given text storage object.

```
- (void)replaceTextStorage:(NSTextStorage *)newTextStorage
```

Parameters

`newTextStorage`
: The text storage object to set.

Discussion
All `NSLayoutManager` objects sharing the original `NSTextStorage` object then share the new one. This method makes all the adjustments necessary to keep these relationships intact, unlike `setTextStorage:` (page 80).

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

Related Sample Code
TextLayoutDemo

Declared In
`NSLayoutManager.h`

rulerAccessoryViewForTextView:paragraphStyle:ruler:enabled:

Returns the the accessory view that the text system uses for its ruler.

```
- (NSView *)rulerAccessoryViewForTextView:(NSTextView *)view
    paragraphStyle:(NSParagraphStyle *)style
    ruler:(NSRulerView *)ruler
    enabled:(BOOL)isEnabled
```

Parameters

view
: The text view using the layout manager.

style
: Sets the state of the controls in the accessory view; must not be `nil`.

ruler
: The ruler view whose accessory view is returned.

isEnabled
: If `YES`, the accessory view is enabled and accepts mouse and keyboard events; if `NO` it's disabled.

Return Value
The accessory view containing tab wells, text alignment buttons, and so on.

Discussion
If you have turned off automatic ruler updating through the use of `setUsesRuler:` so that you can do more complex things, but you still want to display the appropriate accessory view, you can use this 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 customize ruler support. If you do use this method directly, note that it neither installs the ruler accessory view nor sets the markers for the `NSRulerView` object. You must install the accessory view into the ruler using the `NSRulerView` method `setAccessoryView:`. To set the markers, use `rulerMarkersForTextView:paragraphStyle:ruler:` (page 68) to get the markers needed, and then send `setMarkers:` to the ruler.

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

See Also
`– horizontalRulerView` (NSScrollView)

Declared In
`NSLayoutManager.h`

rulerMarkersForTextView:paragraphStyle:ruler:

Returns an array of text ruler objects for the current selection.

```
- (NSArray *)rulerMarkersForTextView:(NSTextView *)view
    paragraphStyle:(NSParagraphStyle *)style ruler:(NSRulerView *)ruler
```

Parameters

view

The text view using the layout manager.

style

Sets the state of the controls in the accessory view; must not be `nil`.

ruler

The ruler view whose ruler markers are returned.

Return Value

An array of `NSRulerMarker` objects representing such things as left and right margins, first-line indent, and tab stops.

Discussion

If you have turned off automatic ruler updating through the use of `setUsesRuler:` so that you can do more complex things, but you still want to display the appropriate accessory view, you can use this 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.

You can set the returned ruler markers with the `NSRulerView` method `setMarkers:`.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- rulerAccessoryViewForTextView:paragraphStyle:ruler:enabled:` (page 67)

Declared In

`NSLayoutManager.h`

setAllowsNonContiguousLayout:

Enables or disables noncontiguous layout.

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

Parameters

flag

If `YES`, noncontiguous layout is enabled; if `NO`, noncontiguous layout is disabled.

Discussion

Passing `YES` in *flag* allows but does not require the layout manager to use noncontiguous layout, and the layout manager may in fact not do so, depending on its configuration.

For more information about noncontiguous layout, see "Noncontiguous Layout" (page 10).

Availability

Available in Mac OS X v10.5 and later.

See Also
`- allowsNonContiguousLayout` (page 22)
`- hasNonContiguousLayout` (page 48)

Declared In
`NSLayoutManager.h`

setAttachmentSize:forGlyphRange:

Sets the size at which the given glyph (assumed to be an attachment) is asked to draw in the given glyph range.

```
- (void)setAttachmentSize:(NSSize)attachmentSize forGlyphRange:(NSRange)glyphRange
```

Parameters

`attachmentSize`
The glyph size to set.

`glyphRange`
The attachment glyph's position in the glyph stream.

Discussion
For a glyph corresponding to an attachment, this method should be called to set the size for the attachment cell to occupy. The glyph's value should be `NSControlGlyph`.

This method is used by the layout mechanism and should be invoked only during typesetting, in almost all cases only by the typesetter. For example, a custom typesetter might invoke it.

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

See Also
`- attachmentSizeForGlyphAtIndex:` (page 23)
`- setDefaultAttachmentScaling:` (page 71)

Declared In
`NSLayoutManager.h`

setBackgroundLayoutEnabled:

Specifies whether the receiver generates glyphs and lays them out when the application's run loop is idle.

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

Parameters

`flag`
If `YES`, background layout is enabled; if `NO`, the receiver performs glyph generation and layout only when necessary.

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

See Also
`- backgroundLayoutEnabled` (page 23)

Declared In
`NSLayoutManager.h`

setBoundsRect:forTextBlock:glyphRange:

Sets the bounding rectangle enclosing a given text block containing the given glyph range.

```
- (void)setBoundsRect:(NSRect)rect forTextBlock:(NSTextBlock *)block
    glyphRange:(NSRange)glyphRange
```

Parameters

rect
: The bounding rectangle to set.

block
: The text block whose bounding rectangle is set.

glyphRange
: The range of glyphs in the text block.

Discussion
This method causes glyph generation but not layout. Block layout rectangles and bounds rectangles are always in container coordinates.

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

See Also
`– boundingRectForGlyphRange:inTextContainer:` (page 24)
`– boundsRectForTextBlock:atIndex:effectiveRange:` (page 24)
`– boundsRectForTextBlock:glyphRange:` (page 25)

Declared In
`NSLayoutManager.h`

setCharacterIndex:forGlyphAtIndex:

Sets the index of the character corresponding to the glyph at the given glyph index.

```
- (void)setCharacterIndex:(NSUInteger)charIndex
    forGlyphAtIndex:(NSUInteger)glyphIndex
```

Parameters

charIndex
: The index to set.

glyphIndex
: The glyph corresponding to the character whose index is set. The glyph must already be present.

Discussion
This method is for use by the glyph-generation mechanism and doesn't perform any invalidation or generation of the glyphs or layout. This method should be invoked only during glyph generation and typesetting, in almost all cases only by the glyph generator or typesetter. For example, a custom glyph generator or typesetter might invoke it.

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

See Also
- `characterIndexForGlyphAtIndex:` (page 26)
- `characterRangeForGlyphRange:actualGlyphRange:` (page 26)
- `glyphRangeForCharacterRange:actualCharacterRange:` (page 47)

Declared In
`NSLayoutManager.h`

setDefaultAttachmentScaling:

Sets the default scaling behavior to the given scaling if an attachment image is too large to fit in a text container.

```
- (void)setDefaultAttachmentScaling:(NSImageScaling)scaling
```

Parameters

scaling
The scaling behavior to set. See `NSImageScaling` for possible values. The default is `NSScaleNone`, meaning that images clip rather than scaling.

Discussion
Attachment cells control their own size and drawing, so this setting is only advisory to them, but Application Kit–supplied attachment cells respect it.

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

See Also
- `defaultAttachmentScaling` (page 27)

Declared In
`NSLayoutManager.h`

setDelegate:

Sets the receiver's delegate.

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

Parameters

anObject
The delegate for the receiver.

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

See Also
- `delegate` (page 29)

Related Sample Code
Quartz Composer WWDC 2005 TextEdit

TextEditPlus

Declared In
`NSLayoutManager.h`

setDrawsOutsideLineFragment:forGlyphAtIndex:

Specifies whether the given glyph exceeds the bounds of the line fragment where it's laid out.

```
- (void)setDrawsOutsideLineFragment:(BOOL)flag
    forGlyphAtIndex:(NSUInteger)glyphIndex
```

Parameters

flag
If `YES`, sets the given glyph to draw outside its line fragment; if `NO`, the glyph does not draw outside.

glyphIndex
Index of the glyph to set.

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 information is important for determining whether additional lines need to be redrawn as a result of changes to any given line fragment.

This method is used by the layout mechanism and should be invoked only during typesetting, in almost all cases only by the typesetter. For example, a custom typesetter might invoke it.

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

See Also
`- drawsOutsideLineFragmentForGlyphAtIndex:` (page 31)

Declared In
`NSLayoutManager.h`

setExtraLineFragmentRect:usedRect:textContainer:

Sets the bounds and container for the extra line fragment.

```
- (void)setExtraLineFragmentRect:(NSRect)aRect usedRect:(NSRect)usedRect
    textContainer:(NSTextContainer *)aTextContainer
```

Parameters

aRect
The rectangle to set.

usedRect
Indicates where the insertion point is drawn.

aTextContainer
    The text container where the rectangle is to be laid out.

Discussion
The extra line fragment is used when the text backing ends with a hard line break or when the text backing is totally empty, to define the extra line which needs to be displayed at the end of the text. If the text backing is not empty and does not end with a hard line break, this should be set to `NSZeroRect` and `nil`.

Line fragment rectangles and line fragment used rectangles are always in container coordinates.

This method is used by the layout mechanism and should be invoked only during typesetting, in almost all cases only by the typesetter. For example, a custom typesetter might invoke it.

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

See Also
- `extraLineFragmentRect` (page 36)
- `extraLineFragmentUsedRect` (page 36)
- `textContainers` (page 91)

Declared In
`NSLayoutManager.h`

setGlyphGenerator:

Sets the glyph generator used by this layout manager.

```
- (void)setGlyphGenerator:(NSGlyphGenerator *)glyphGenerator
```

Parameters

glyphGenerator
    The new glyph generator to set.

Discussion
Setting the glyph generator invalidates all glyphs and layout in the layout manager.

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

See Also
- `glyphGenerator` (page 43)

Declared In
`NSLayoutManager.h`

setHyphenationFactor:

Sets the threshold controlling when hyphenation is done.

```
- (void)setHyphenationFactor:(float)factor
```

Parameters

`factor`

The hyphenation factor, ranging from 0.0 to 1.0. By default, the value is 0.0, meaning hyphenation is off. A *`factor`* of 1.0 causes hyphenation to be attempted always.

Discussion

Whenever (width of the real contents of the line) / (the line fragment width) is below *`factor`*, hyphenation is attempted when laying out the line. Hyphenation slows down text layout and increases memory usage, so it should be used sparingly.

May be overridden on a per-paragraph basis by the `NSParagraphStyle` method `hyphenationFactor`.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- hyphenationFactor` (page 48)

Declared In

`NSLayoutManager.h`

setIntAttribute:value:forGlyphAtIndex:

Sets a custom attribute value for a given glyph.

```
- (void)setIntAttribute:(NSInteger)attributeTag value:(NSInteger)val
    forGlyphAtIndex:(NSUInteger)glyphIndex
```

Parameters

`attributeTag`

The custom attribute.

`val`

The new attribute value.

`glyphIndex`

Index of the glyph whose attribute is set.

Discussion

Custom attributes are glyph attributes such as `NSGlyphInscription` or attributes defined by subclasses. Nonnegative tags are reserved by Apple; you can define your own attributes with negative tags and set values using this method.

This method is part of the `NSGlyphStorage` protocol, for use by the glyph generator to set attributes. It is not usually necessary for anyone but the glyph generator (and perhaps the typesetter) to call it. It is provided as a public method so subclasses can extend it to accept other glyph attributes. To add new glyph attributes to the text system you must do two things. First, you need to arrange for the glyph generator or typesetter to generate and interpret it. Second, you need to subclass `NSLayoutManager` to provide someplace to store the new attribute, overriding this method and `intAttribute:forGlyphAtIndex:` (page 51) to recognize the new attribute tags and respond to them, while passing any other attributes to the superclass implementation. The `NSLayoutManager` implementation understands the glyph attributes which it is prepared to store, as enumerated in "Glyph Attributes" (page 97).

Availability

Available in Mac OS X v10.0 and later.

See Also

`- intAttribute:forGlyphAtIndex:` (page 51)

Declared In

`NSLayoutManager.h`

setLayoutRect:forTextBlock:glyphRange:

Sets the layout rectangle enclosing the given text block containing the given glyph range.

```
- (void)setLayoutRect:(NSRect)rect forTextBlock:(NSTextBlock *)block
    glyphRange:(NSRange)glyphRange
```

Parameters

rect

The layout rectangle to set.

block

The text block whose layout rectangle is set.

glyphRange

The range of glyphs in the text block.

Discussion

This method causes glyph generation but not layout. Block layout rectangles and bounds rectangles are always in container coordinates.

Availability

Available in Mac OS X v10.4 and later.

See Also

`- layoutRectForTextBlock:atIndex:effectiveRange:` (page 56)

`- layoutRectForTextBlock:glyphRange:` (page 57)

Declared In

`NSLayoutManager.h`

setLineFragmentRect:forGlyphRange:usedRect:

Associates the given line fragment bounds with the given range of glyphs.

```
- (void)setLineFragmentRect:(NSRect)fragmentRect forGlyphRange:(NSRange)glyphRange
     usedRect:(NSRect)usedRect
```

Parameters

fragmentRect

The rectangle of the line fragment.

glyphRange

The range of glyphs to be associated with *fragmentRect*.

usedRect

The portion of *fragmentRect* that actually contains glyphs or other marks that are drawn (including the text container's line fragment padding. Must be equal to or contained within *fragmentRect*.

Discussion

The typesetter must specify the text container first with `setTextContainer:forGlyphRange:` (page 79), and it sets the exact positions of the glyphs afterwards with `setLocation:forStartOfGlyphRange:` (page 76).

In the course of layout, all glyphs should end up being included in a range passed to this method, but only glyphs that start a new line fragment should be at the start of such ranges.

Line fragment rectangles and line fragment used rectangles are always in container coordinates.

This method is used by the layout mechanism and should be invoked only during typesetting, in almost all cases only by the typesetter. For example, a custom typesetter might invoke it.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- lineFragmentRectForGlyphAtIndex:effectiveRange:withoutAdditionalLayout:` (page 58)
`- lineFragmentRectForGlyphAtIndex:effectiveRange:` (page 57)
`- lineFragmentUsedRectForGlyphAtIndex:effectiveRange:withoutAdditionalLayout:` (page 59)
`- lineFragmentUsedRectForGlyphAtIndex:effectiveRange:` (page 59)

Declared In

`NSLayoutManager.h`

setLocation:forStartOfGlyphRange:

Sets the location for the first glyph of the given range.

```
- (void)setLocation:(NSPoint)aPoint forStartOfGlyphRange:(NSRange)glyphRange
```

Parameters

aPoint
The location to which the first glyph is set, relative to the origin of the glyph's line fragment origin.

glyphRange
The glyphs whose location is set.

Discussion

Setting the location for a glyph range implies that its first glyph is not nominally spaced with respect to the previous glyph. In the course of layout, all glyphs should end up being included in a range passed to this method, but only glyphs that start a new nominal range should be at the start of such ranges. The first glyph in a line fragment should always start a new nominal range. Glyph locations are given relative to their line fragment rectangle's origin.

Before setting the location for a glyph range, you must specify the text container with `setTextContainer:forGlyphRange:` (page 79) and the line fragment rectangle with `setLineFragmentRect:forGlyphRange:usedRect:` (page 75).

This method is used by the layout mechanism and should be invoked only during typesetting, in almost all cases only by the typesetter. For example, a custom typesetter might invoke it.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- rangeOfNominallySpacedGlyphsContainingIndex:` (page 62)

Declared In

`NSLayoutManager.h`

setLocations:startingGlyphIndexes:count:forGlyphRange:

Sets locations for many glyph ranges at once.

```
- (void)setLocations:(NSPointArray)locations startingGlyphIndexes:(NSUInteger
    *)glyphIndexes count:(NSUInteger)count forGlyphRange:(NSRange)glyphRange
```

Parameters

locations

The locations to which the first glyph in each range is set, relative to the origin of the glyph's line fragment origin.

glyphIndexes

Indexes in *glyphRange* of the glyphs whose locations are set.

count

The number of glyphs whose locations are set.

glyphRange

The entire glyph range containing all the glyphs whose locations are set.

Discussion

This method enables the typesetter to set locations for glyph ranges in bulk. All of the specified glyph indexes should lie within the specified glyph range. The first of them should be equal to `glyphRange.location`, and the remainder should increase monotonically. Each location is set as the location for the range beginning at the corresponding glyph index, and continuing until the subsequent glyph index, or until the end of the glyph range for the last location. Thus this method is equivalent to calling `setLocation:forStartOfGlyphRange:` (page 76) for a set of ranges covering all of the glyphs in *glyphRange*.

This method is used by the layout mechanism and should be invoked only during typesetting, in almost all cases only by the typesetter. For example, a custom typesetter might invoke it.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`NSLayoutManager.h`

setNotShownAttribute:forGlyphAtIndex:

Sets the glyph at the given index to be one that isn't shown.

```
- (void)setNotShownAttribute:(BOOL)flag forGlyphAtIndex:(NSUInteger)glyphIndex
```

Parameters

flag

If `YES`, the glyph is not shown; if `NO`, it is shown.

glyphIndex
Index of the glyph whose attribute is set.

Discussion
The typesetter decides which glyphs are not shown and sets this attribute in the layout manager to ensure that those glyphs are not displayed. For example, a tab or newline character doesn't leave any marks; it just indicates where following glyphs are laid out.

Raises an `NSRangeException` if *glyphIndex* is out of bounds.

This method is used by the layout mechanism and should be invoked only during typesetting, in almost all cases only by the typesetter. For example, a custom typesetter might invoke it.

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

See Also
`- notShownAttributeForGlyphAtIndex:` (page 61)

Declared In
`NSLayoutManager.h`

setShowsControlCharacters:

Specifies whether to substitute visible glyphs for control characters in layout.

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

Parameters
flag
If `YES`, the receiver substitutes visible glyphs for control characters if the font and script support it; if `NO`, it doesn't. The default is `NO`.

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

See Also
`- setShowsInvisibleCharacters:` (page 78)
`- showsControlCharacters` (page 83)

Declared In
`NSLayoutManager.h`

setShowsInvisibleCharacters:

Specifies whether to substitute visible glyphs for whitespace and other typically invisible characters in layout.

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

Parameters
flag
If `YES`, the receiver substitutes visible glyphs for invisible characters if the font and script support it; if `NO`, it doesn't. The default is `NO`.

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

See Also
- `setShowsControlCharacters:` (page 78)
- `showsInvisibleCharacters` (page 84)

Declared In
`NSLayoutManager.h`

setTemporaryAttributes:forCharacterRange:

Sets one or more temporary attributes for the specified character range.

```
- (void)setTemporaryAttributes:(NSDictionary *)attrs
    forCharacterRange:(NSRange)charRange
```

Parameters

attrs
Attributes dictionary containing the temporary attributes to set.

charRange
The range of characters to which the specified attributes apply.

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 recognized are those that do not affect layout (colors, underlines, and so on).

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

See Also
- `addTemporaryAttributes:forCharacterRange:` (page 21)
- `removeTemporaryAttribute:forCharacterRange:` (page 64)
- `temporaryAttributesAtCharacterIndex:effectiveRange:` (page 87)

Declared In
`NSLayoutManager.h`

setTextContainer:forGlyphRange:

Sets text container where the glyphs in the given range are laid out.

```
- (void)setTextContainer:(NSTextContainer *)aTextContainer
    forGlyphRange:(NSRange)glyphRange
```

Parameters

aTextContainer
The text container to set.

glyphRange
The range of glyphs to lay out.

Discussion
The layout within the container is specified with the `setLineFragmentRect:forGlyphRange:usedRect:` (page 75) and `setLocation:forStartOfGlyphRange:` (page 76) methods.

This method is used by the layout mechanism and should be invoked only during typesetting, in almost all cases only by the typesetter. For example, a custom typesetter might invoke it.

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

See Also
`– textContainerForGlyphAtIndex:effectiveRange:` (page 89)

Declared In
`NSLayoutManager.h`

setTextStorage:

Sets the receiver's `NSTextStorage` object.

```
- (void)setTextStorage:(NSTextStorage *)textStorage
```

Parameters

`textStorage`
The text storage object to set.

Discussion
This method is invoked automatically when you add an `NSLayoutManager` to an `NSTextStorage` object; you should never need to invoke it directly, but you might want to override it. If you want to replace the `NSTextStorage` object for an established group of text-system objects containing the receiver, use `replaceTextStorage:` (page 66).

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

See Also
`– addLayoutManager:` (`NSTextStorage`)

Declared In
`NSLayoutManager.h`

setTypesetter:

Sets the current typesetter.

```
- (void)setTypesetter:(NSTypesetter *)typesetter
```

Parameters

`typesetter`
The typesetter for the receiver.

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

See Also

- `typesetter` (page 93)

Declared In

`NSLayoutManager.h`

setTypesetterBehavior:

Sets the default typesetter behavior.

```
- (void)setTypesetterBehavior:(NSTypesetterBehavior)theBehavior
```

Parameters

`theBehavior`

An `NSTypesetterBehavior` (page 99) constant that specifies the behavior for the receiver.

Discussion

The typesetter behavior affects glyph spacing and line height.

If the application was linked on a system prior to Mac OS X v10.2, `NSLayoutManager` uses `NSTypesetterOriginalBehavior` by default.

Availability

Available in Mac OS X v10.2 and later.

See Also

- `typesetterBehavior` (page 93)

Declared In

`NSLayoutManager.h`

setUsesFontLeading:

Specifies whether or not the receiver uses the leading provided in the font.

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

Parameters

`flag`

If `YES`, the receiver uses the font's leading; if `NO`, it does not.

Discussion

By default, a layout manager uses leading as specified by the font. However, this is not appropriate for most user-interface text, for which a fixed leading is usually specified by user-interface layout guidelines. This method enables the use of the font's leading to be turned off.

Availability

Available in Mac OS X v10.5 and later.

See Also

- `usesFontLeading` (page 95)
- `setLineSpacing:` (`NSMutableParagraphStyle`)

Declared In
`NSLayoutManager.h`

setUsesScreenFonts:

Controls using screen fonts to calculate layout and display text.

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

Parameters

flag
If `YES`, the receiver uses screen fonts; if `NO`, it doesn't.

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

See Also
`- usesScreenFonts` (page 95)
`- substituteFontForFont:` (page 85)

Related Sample Code
TextLayoutDemo

Declared In
`NSLayoutManager.h`

showAttachmentCell:inRect:characterIndex:

Draws an attachment cell.

```
- (void)showAttachmentCell:(NSCell *)cell inRect:(NSRect)rect
    characterIndex:(NSUInteger)attachmentIndex
```

Parameters

cell
The attachment cell to draw.

rect
The rectangle within which to draw *cell*.

attachmentIndex
The location of the attachment cell.

Discussion
The *attachmentIndex* parameter is provided for cells that alter their appearance based on their location.

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

Declared In
`NSLayoutManager.h`

showPackedGlyphs:length:glyphRange:atPoint:font:color:printingAdjustment:

Draws a range of glyphs.

```
- (void)showPackedGlyphs:(char *)glyphs length:(NSUInteger)glyphLen
    glyphRange:(NSRange)glyphRange atPoint:(NSPoint)point font:(NSFont *)font
    color:(NSColor *)color printingAdjustment:(NSSize)printingAdjustment
```

Parameters

glyphs
: The glyphs to draw; may contain embedded `NULL` bytes.

glyphLen
: The number of bytes pointed at by *glyphs*; this is twice the number of glyphs contained.

glyphRange
: The range of glyphs to draw.

point
: The point at which to draw the glyphs.

font
: The font of the glyphs to draw.

color
: Color of the glyphs to draw.

printingAdjustment
: `NSZeroSize` when drawing to the screen, but when printing may contain values by which the nominal spacing between the characters should be adjusted.

Discussion
The *glyphRange*, *point*, *font*, and *color* parameters are passed in merely for information purposes. They are already set in the graphics state. If for any reason you modify the set color or font, you must restore it before returning from this method.

You should never call this method, but you might override it.

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

Declared In
`NSLayoutManager.h`

showsControlCharacters

Indicates whether the receiver substitutes visible glyphs for control characters.

```
- (BOOL)showsControlCharacters
```

Return Value
`YES` if the receiver substitutes visible glyphs for control characters if the font and script support it; `NO` if it doesn't.

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

See Also

- `showsInvisibleCharacters` (page 84)
- `setShowsControlCharacters:` (page 78)

Declared In

`NSLayoutManager.h`

showsInvisibleCharacters

Indicates whether the receiver substitutes visible glyphs for whitespace and other typically invisible characters in layout.

```
- (BOOL)showsInvisibleCharacters
```

Return Value

`YES` if the receiver substitutes visible glyphs for invisible characters if the font and script support it; otherwise `NO`. The default is `NO`.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `showsControlCharacters` (page 83)
- `setShowsInvisibleCharacters:` (page 78)

Declared In

`NSLayoutManager.h`

strikethroughGlyphRange:strikethroughType:lineFragmentRect: lineFragmentGlyphRange:containerOrigin:

Calculates and draws strikethrough for the glyphs in the given range.

```
- (void)strikethroughGlyphRange:(NSRange)glyphRange
    strikethroughType:(NSInteger)strikethroughVal lineFragmentRect:(NSRect)lineRect
     lineFragmentGlyphRange:(NSRange)lineGlyphRange
    containerOrigin:(NSPoint)containerOrigin
```

Parameters

glyphRange
: The range of glyphs for which to draw a strikethrough. The range must belong to a single line fragment rectangle (as returned by `lineFragmentRectForGlyphAtIndex:effectiveRange:` (page 57)).

strikethroughVal
: The style of underlining to draw. This value is a mask derived from the value for `NSUnderlineStyleAttributeName`—for example, `(NSUnderlinePatternDash | NSUnderlineStyleThick | NSUnderlineByWordMask)`. Subclasses can define custom underlining styles.

lineRect
: The line fragment rectangle containing the glyphs to draw strikethrough for.

lineGlyphRange
: The range of all glyphs within *lineRect*.

containerOrigin
: The origin of the line fragment rectangle's `NSTextContainer` in its `NSTextView`.

Discussion
This method determines which glyphs actually need to have a strikethrough drawn based on *strikethroughVal*. After determining which glyphs to draw strikethrough on, this method invokes `drawStrikethroughForGlyphRange:strikethroughType:baselineOffset:lineFragmentRect:lineFragmentGlyphRange:containerOrigin:` (page 31) for each contiguous range of glyphs that requires it.

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

Declared In
`NSLayoutManager.h`

substituteFontForFont:

Returns a screen font suitable for use in place of the given font, if one is available.

```
- (NSFont *)substituteFontForFont:(NSFont *)originalFont
```

Parameters

originalFont
: The font to replace.

Return Value
A screen font suitable for use in place of *originalFont*, or simply *originalFont* if a screen font can't be used or isn't available.

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.

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

See Also
`- usesScreenFonts` (page 95)

Declared In
`NSLayoutManager.h`

temporaryAttribute:atCharacterIndex:effectiveRange:

Returns the value for the temporary attribute with a given name of the character at a given index, and by reference the range over which the attribute applies.

```
- (id)temporaryAttribute:(NSString *)attrName atCharacterIndex:(NSUInteger)location
      effectiveRange:(NSRangePointer)range
```

Parameters

`attrName`

The name of a temporary attribute.

`location`

The index for which to return attributes. This value must not exceed the bounds of the receiver.

`range`

If non-`NULL`:

- If the named attribute exists at *`location`*, on output, contains the range over which the named attribute's value applies.
- If the named attribute does not exist at *`location`*, on output, contains the range over which the attribute does not exist.

The range isn't necessarily the maximum range covered by *`attrName`*, and its extent is implementation-dependent. If you need the maximum range, use `temporaryAttribute:atCharacterIndex:longestEffectiveRange:inRange:` (page 86). If you don't need this value, pass `NULL`.

Return Value

The value for the temporary attribute named *`attrName`* of the character at index *`location`*, or `nil` if there is no such attribute.

Availability

Available in Mac OS X v10.5 and later.

See Also

`– temporaryAttributesAtCharacterIndex:effectiveRange:` (page 87)
`– temporaryAttribute:atCharacterIndex:longestEffectiveRange:inRange:` (page 86)

Declared In

`NSLayoutManager.h`

temporaryAttribute:atCharacterIndex:longestEffectiveRange:inRange:

Returns the value for the temporary attribute with a given name of the character at a given index, and by reference the maximum range over which the attribute applies.

```
- (id)temporaryAttribute:(NSString *)attrName atCharacterIndex:(NSUInteger)location
    longestEffectiveRange:(NSRangePointer)range inRange:(NSRange)rangeLimit
```

Parameters

`attrName`

The name of a temporary attribute.

`location`

The index for which to return attributes. This value must not exceed the bounds of the receiver.

range

If non-`NULL`:

- If the named attribute exists at *location*, on output, contains the maximum range over which the named attribute's value applies, clipped to *rangeLimit*.
- If the named attribute does not exist at *location*, on output, contains the maximum range over which the attribute does not exist.

If you don't need this value, pass `NULL`.

rangeLimit

The range over which to search for continuous presence of *attrName*. This value must not exceed the bounds of the receiver.

Return Value
The value for the attribute named *attrName* of the character at *location*, or `nil` if there is no such attribute.

Discussion
If you don't need the longest effective range, it's far more efficient to use the `temporaryAttribute:atCharacterIndex:effectiveRange:` (page 85) method to retrieve the attribute value.

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

See Also
`– temporaryAttributesAtCharacterIndex:effectiveRange:` (page 87)
`– temporaryAttribute:atCharacterIndex:effectiveRange:` (page 85)

Declared In
`NSLayoutManager.h`

temporaryAttributesAtCharacterIndex:effectiveRange:

Returns the dictionary of temporary attributes for the character range specified in *effectiveCharRange* at character index *charIndex*.

```
- (NSDictionary *)temporaryAttributesAtCharacterIndex:(NSUInteger)charIndex
    effectiveRange:(NSRangePointer)effectiveCharRange
```

Return Value
The dictionary of temporary attributes for the character range specified in *effectiveCharRange* at character index *charIndex*.

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 recognized are those that do not affect layout (colors, underlines, and so on).

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

See Also
`– addTemporaryAttributes:forCharacterRange:` (page 21)
`– removeTemporaryAttribute:forCharacterRange:` (page 64)

`- setTemporaryAttributes:forCharacterRange:` (page 79)

Declared In
`NSLayoutManager.h`

temporaryAttributesAtCharacterIndex:longestEffectiveRange:inRange:

Returns the temporary attributes for the character at a given index, and by reference the maximum range over which the attributes apply.

```
- (NSDictionary *)temporaryAttributesAtCharacterIndex:(NSUInteger)location
    longestEffectiveRange:(NSRangePointer)range inRange:(NSRange)rangeLimit
```

Parameters

location
: The index for which to return attributes. This value must not exceed the bounds of the receiver.

range
: If not `NULL`, on output, contains the maximum range over which the attributes and values are the same as those at *location*, clipped to *rangeLimit*.

rangeLimit
: The range over which to search for continuous presence of the attributes at *location*. This value must not exceed the bounds of the receiver.

Return Value
The attributes for the character at *location*.

Discussion
If you don't need the longest effective range, it's far more efficient to use the `temporaryAttributesAtCharacterIndex:effectiveRange:` (page 87) method to retrieve the attribute value.

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

See Also
`- temporaryAttributesAtCharacterIndex:effectiveRange:` (page 87)
`- temporaryAttribute:atCharacterIndex:longestEffectiveRange:inRange:` (page 86)

Declared In
`NSLayoutManager.h`

textContainerChangedGeometry:

Invalidates the layout information, and possibly glyphs, for the given text container and all subsequent `NSTextContainer` objects.

```
- (void)textContainerChangedGeometry:(NSTextContainer *)aTextContainer
```

Parameters

aTextContainer
: The text container whose layout is invalidated.

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).

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

Declared In
`NSLayoutManager.h`

textContainerChangedTextView:

Updates information needed to manage `NSTextView` objects in the given text container.

```
- (void)textContainerChangedTextView:(NSTextContainer *)aTextContainer
```

Parameters

aTextContainer
The text container whose text view has changed.

Discussion
This method is called by a text container, whenever its text view changes, to keep notifications synchronized. You should rarely need to invoke it directly.

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

Declared In
`NSLayoutManager.h`

textContainerForGlyphAtIndex:effectiveRange:

Returns the container in which the given glyph is laid out and (optionally) by reference the whole range of glyphs that are in that container.

```
- (NSTextContainer *)textContainerForGlyphAtIndex:(NSUInteger)glyphIndex
    effectiveRange:(NSRangePointer)effectiveGlyphRange
```

Parameters

glyphIndex
Index of a glyph in the returned container.

effectiveGlyphRange
If not `NULL`, on output, points to the whole range of glyphs that are in the returned container.

Return Value
The text container in which the glyph at *glyphIndex* is laid out.

Discussion
This method causes glyph generation and layout for the line fragment containing the specified glyph, or if noncontiguous layout is not enabled, up to and including that line fragment. If noncontiguous layout is not enabled and *`effectiveGlyphRange`* is not `NULL`, this method additionally causes glyph generation and layout for the entire text container containing the specified glyph.

Overriding this method is not recommended. Any changes to the returned glyph range should be done at the typesetter level.

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

See Also
`- setTextContainer:forGlyphRange:` (page 79)

Declared In
`NSLayoutManager.h`

textContainerForGlyphAtIndex:effectiveRange:withoutAdditionalLayout:

Returns the container in which the given glyph is laid out and (optionally) by reference the whole range of glyphs that are in that container.

```
- (NSTextContainer *)textContainerForGlyphAtIndex:(NSUInteger)glyphIndex
    effectiveRange:(NSRangePointer)effectiveGlyphRange
    withoutAdditionalLayout:(BOOL)flag
```

Parameters

`glyphIndex`
Index of a glyph in the returned container.

`effectiveGlyphRange`
If not `NULL`, on output, points to the whole range of glyphs that are in the returned container.

`flag`
If `YES`, glyph generation and layout are not performed, so this option should not be used unless layout is known to be complete for the range in question, or unless noncontiguous layout is enabled; if `NO`, both are performed as needed.

Return Value
The text container in which the glyph at *`glyphIndex`* is laid out.

Discussion
This method is primarily for use from within `NSTypesetter`, after layout is complete for the range in question, but before the layout manager's call to `NSTypesetter` has returned. In that case glyph and layout holes have not yet been recalculated, so the layout manager does not yet know that layout is complete for that range, and this variant must be used.

Overriding this method is not recommended. Any changes to the returned glyph range should be done at the typesetter level.

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

See Also
`- setTextContainer:forGlyphRange:` (page 79)

Declared In
`NSLayoutManager.h`

textContainers

Returns the receiver's text containers.

```
- (NSArray *)textContainers
```

Return Value
The receiver's text containers.

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

See Also
`- addTextContainer:` (page 21)
`- insertTextContainer:atIndex:` (page 50)
`- removeTextContainerAtIndex:` (page 65)

Related Sample Code
Quartz Composer WWDC 2005 TextEdit
Sketch-112
TextEditPlus

Declared In
`NSLayoutManager.h`

textStorage

Returns the receiver's text storage object.

```
- (NSTextStorage *)textStorage
```

Return Value
The receiver's text storage.

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

See Also
`- setTextStorage:` (page 80)
`- replaceTextStorage:` (page 66)

Declared In
`NSLayoutManager.h`

textStorage:edited:range:changeInLength:invalidatedRange:

Invalidates glyph and layout information for a portion of the text in the given text storage object.

```
- (void)textStorage:(NSTextStorage *)aTextStorage edited:(NSUInteger)mask
    range:(NSRange)newCharRange changeInLength:(NSInteger)delta
    invalidatedRange:(NSRange)invalidatedCharRange
```

Parameters

aTextStorage
: The text storage whose information is invalidated.

mask
: Specifies the nature of the changes. Its value is made by combining with the C bitwise OR operator the constants described in "Change notifications" in `NSTextStorage` (`NSTextStorageEditedAttributes` and `NSTextStorageEditedCharacters`).

newCharRange
: Indicates the extent of characters resulting from the edits.

delta
: If the `NSTextStorageEditedCharacters` bit of *mask* is set, gives the number of characters added to or removed from the original range (otherwise its value is irrelevant).

invalidatedCharRange
: Represents the range of characters affected after attributes have been fixed. Is either equal to *newCharRange* or larger. For example, deleting a paragraph separator character invalidates the layout information for all characters in the paragraphs that precede and follow the separator.

Discussion

This message is sent from the `NSTextStorage`object's `processEditing` method to indicate that its characters or attributes have changed. This method invalidates glyphs and layout for the affected characters.

For example, after replacing "The" with "Several" to produce the string "Several files couldn't be saved", *newCharRange* is {0, 7} and *delta* 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 `textStorage:edited:range:changeInLength:invalidatedRange:` messages are sent in a series to each `NSLayoutManager` object associated with the text storage object, so the layout managers receiving them shouldn't edit *aTextStorage* while this method is executing. If one of them does, the *newCharRange*, *delta*, and *invalidatedCharRange* arguments are incorrect for all following layout managers that receive the message.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- invalidateLayoutForCharacterRange:isSoft:actualCharacterRange:` (page 54)

Declared In

`NSLayoutManager.h`

textViewForBeginningOfSelection

Returns the text view containing the first glyph in the selection.

```
- (NSTextView *)textViewForBeginningOfSelection
```

Return Value
The text view containing the first glyph in the selection, or `nil` if there's no selection or there isn't enough layout information to determine the text view.

Discussion
This method does not cause layout if the beginning of the selected range is not yet laid out.

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

Declared In
`NSLayoutManager.h`

typesetter

Returns the receiver's typesetter.

```
- (NSTypesetter *)typesetter
```

Return Value
The receiver's typesetter.

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

See Also
`- setTypesetter:` (page 80)

Declared In
`NSLayoutManager.h`

typesetterBehavior

Returns the current typesetter behavior.

```
- (NSTypesetterBehavior)typesetterBehavior
```

Return Value
The current typesetter behavior value.

Availability
Available in Mac OS X v10.2 and later.

See Also
`- setTypesetterBehavior:` (page 81)

Declared In
`NSLayoutManager.h`

underlineGlyphRange:underlineType:lineFragmentRect:lineFragmentGlyphRange:containerOrigin:

Calculates subranges to be underlined for the glyphs in a given range and draws the underlining as appropriate.

```
- (void)underlineGlyphRange:(NSRange)glyphRange underlineType:(NSInteger)underlineVal
    lineFragmentRect:(NSRect)lineRect lineFragmentGlyphRange:(NSRange)lineGlyphRange
    containerOrigin:(NSPoint)containerOrigin
```

Parameters

glyphRange

A range of glyphs, which must belong to a single line fragment rectangle (as returned by `lineFragmentRectForGlyphAtIndex:effectiveRange:` (page 57)).

underlineVal

The style of underlining to draw. This value is a mask derived from the value for `NSUnderlineStyleAttributeName`—for example, `(NSUnderlinePatternDash | NSUnderlineStyleThick | NSUnderlineByWordMask)`. Subclasses can define custom underlining styles.

lineRect

The line fragment rectangle containing the glyphs to draw underlining for.

lineGlyphRange

The range of all glyphs within that line fragment rectangle.

containerOrigin

The origin of the line fragment rectangle's `NSTextContainer` in its `NSTextView`.

Discussion

This method determines which glyphs actually need to be underlined based on *underlineVal*. With `NSUnderlineStyleSingle`, 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:underlineType:baselineOffset:lineFragmentRect:lineFragmentGlyphRange:containerOrigin:` (page 32) for each contiguous range of glyphs that requires it.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `textContainerForGlyphAtIndex:effectiveRange:` (page 89)
- `textContainerOrigin` (NSTextView)

Declared In

`NSLayoutManager.h`

usedRectForTextContainer:

Returns the bounding rectangle for the glyphs laid out in the given text container.

```
- (NSRect)usedRectForTextContainer:(NSTextContainer *)aTextContainer
```

Discussion
Returns the text container's currently used area, which determines the size that the view would need to be in order to display all the glyphs that are currently laid out in the container. This causes neither glyph generation nor layout.

Used rectangles are always in container coordinates.

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

See Also
`- containerSize` (NSTextContainer)

Related Sample Code
Sketch-112

Declared In
`NSLayoutManager.h`

usesFontLeading

Indicates whether the receiver uses the leading provided in the font.

```
- (BOOL)usesFontLeading
```

Return Value
`YES` if the receiver uses the font's leading; otherwise, `NO`.

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

See Also
`- setUsesFontLeading:` (page 81)

Declared In
`NSLayoutManager.h`

usesScreenFonts

Indicates whether the receiver uses screen fonts to calculate layout and display text.

```
- (BOOL)usesScreenFonts
```

Return Value
`YES` if the receiver calculates layout and displays text using screen fonts when possible; otherwise, `NO`.

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

See Also
`- setUsesScreenFonts:` (page 82)
`- substituteFontForFont:` (page 85)

Declared In
`NSLayoutManager.h`

Delegate Methods

layoutManager:didCompleteLayoutForTextContainer:atEnd:

Informs the delegate that the given layout manager has finished laying out text in the given text container.

```
- (void)layoutManager:(NSLayoutManager *)aLayoutManager
    didCompleteLayoutForTextContainer:(NSTextContainer *)aTextContainer
    atEnd:(BOOL)flag
```

Parameters

aLayoutManager
The layout manager doing the layout.

aTextContainer
The text container in which layout is complete. If `nil`, if there aren't enough containers to hold all the text; the delegate can use this information as a cue to add another text container.

flag
If `YES`, *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.

Discussion
This message is sent whenever a text container has been filled. This method can be useful for paginating.

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

Declared In
`NSLayoutManager.h`

layoutManager:shouldUseTemporaryAttributes:forDrawingToScreen:atCharacterIndex:effectiveRange:

Sent when the layout manager is drawing and needs to decide whether or not to use temporary attributes.

```
- (NSDictionary *)layoutManager:(NSLayoutManager *)layoutManager
    shouldUseTemporaryAttributes:(NSDictionary *)attrs
    forDrawingToScreen:(BOOL)toScreen
    atCharacterIndex:(NSUInteger)charIndex
    effectiveRange:(NSRangePointer)effectiveCharRange
```

Parameters

layoutManager
The layout manager sending the message.

attrs
The temporary attributes currently in effect for the given character range.

toScreen
: `YES` if the layout manager is drawing to the screen; otherwise, `NO`.

charIndex
: Index of the first character in the range being drawn.

effectiveCharRange
: On input and output, the effective range to which the temporary attributes apply.

Return Value
The temporary attributes for the layout manager to use, or `nil` if no temporary attributes are to be used.

Discussion
The default behavior, if this method is not implemented, is to use temporary attributes only when drawing to the screen, so an implementation to match that behavior would return *attrs* if *toScreen* is `YES` and `nil` otherwise, without changing *effectiveCharRange*.

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

Declared In
`NSLayoutManager.h`

layoutManagerDidInvalidateLayout:

Informs the delegate that the given layout manager has invalidated layout information (not glyph information).

```
- (void)layoutManagerDidInvalidateLayout:(NSLayoutManager *)sender
```

Parameters

sender
: The layout manager that invalidated layout.

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 of background layout or to enable a button that forces immediate layout of text.

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

Declared In
`NSLayoutManager.h`

Constants

Glyph Attributes

These glyph attribute constants are used only inside the glyph generation machinery, but they must be shared between components.

```
enum {
    NSGlyphAttributeSoft        = 0,
    NSGlyphAttributeElastic     = 1,
    NSGlyphAttributeBidiLevel   = 2,
    NSGlyphAttributeInscribe    = 5
};
```

Constants

`NSGlyphAttributeSoft`

The glyph is soft.

Available in Mac OS X v10.0 and later.

Declared in `NSLayoutManager.h`.

`NSGlyphAttributeElastic`

The glyph is elastic.

Available in Mac OS X v10.0 and later.

Declared in `NSLayoutManager.h`.

`NSGlyphAttributeBidiLevel`

The bidirectional level (direction) of the glyph.

Available in Mac OS X v10.2 and later.

Declared in `NSLayoutManager.h`.

`NSGlyphAttributeInscribe`

Glyph inscription attribute. See [NSGlyphInscription] for possible values.`NSGlyphInscription` (page 98)

Available in Mac OS X v10.0 and later.

Declared in `NSLayoutManager.h`.

Declared In

`NSLayoutManager.h`

NSGlyphInscription

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 `NSGlyphAttributeInscribe`. Glyph inscriptions are set during glyph generation.

```
typedef enum {
    NSGlyphInscribeBase = 0,
    NSGlyphInscribeBelow = 1,
    NSGlyphInscribeAbove = 2,
    NSGlyphInscribeOverstrike = 3,
    NSGlyphInscribeOverBelow = 4
} NSGlyphInscription;
```

Constants

`NSGlyphInscribeBase`

A base glyph; a character that the font can represent with a single glyph.

Available in Mac OS X v10.0 and later.

Declared in `NSLayoutManager.h`.

`NSGlyphInscribeBelow`
: Glyph is rendered below the previous glyph.

  Available in Mac OS X v10.0 and later.

  Declared in `NSLayoutManager.h`.

`NSGlyphInscribeAbove`
: Glyph is rendered above the previous glyph.

  Available in Mac OS X v10.0 and later.

  Declared in `NSLayoutManager.h`.

`NSGlyphInscribeOverstrike`
: Glyph is rendered on top of the previous glyph.

  Available in Mac OS X v10.0 and later.

  Declared in `NSLayoutManager.h`.

`NSGlyphInscribeOverBelow`
: Glyph is rendered on top and below the previous glyph.

  Available in Mac OS X v10.0 and later.

  Declared in `NSLayoutManager.h`.

Discussion

The only constants that the text system currently uses are `NSGlyphInscribeBase` (for most glyphs) and `NSGlyphInscribeOverstrike` (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 `NSGlyphInscribeOverstrike` for the inscribe attribute.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSLayoutManager.h`

NSTypesetterBehavior

These constants define the behavior of `NSLayoutManager` and `NSTypesetter` when laying out lines. They are used by `setTypesetterBehavior:` (page 81) and `typesetterBehavior` (page 93) to control the compatibility level of the typesetter.

```
typedef enum {
    NSTypesetterLatestBehavior = -1,
    NSTypesetterOriginalBehavior = 0,
    NSTypesetterBehavior_10_2_WithCompatibility = 1,
    NSTypesetterBehavior_10_2 = 2,
    NSTypesetterBehavior_10_3 = 3,
    NSTypesetterBehavior_10_4 = 4
} NSTypesetterBehavior;
```

Constants

`NSTypesetterLatestBehavior`

The most current typesetter behavior in the current system version. For Mac OS X v10.2, this behavior is identical to `NSTypesetterBehavior_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.

Available in Mac OS X v10.2 and later.

Declared in `NSLayoutManager.h`.

`NSTypesetterOriginalBehavior`

The original typesetter behavior, as shipped with Mac OS X v10.1 and earlier.

Available in Mac OS X v10.2 and later.

Declared in `NSLayoutManager.h`.

`NSTypesetterBehavior_10_2_WithCompatibility`

Typesetting same as `NSTypesetterBehavior_10_2` but using line widths and height metric calculations that are the same as with `NSTypesetterOriginalBehavior`.

Available in Mac OS X v10.2 and later.

Declared in `NSLayoutManager.h`.

`NSTypesetterBehavior_10_2`

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.

Available in Mac OS X v10.2 and later.

Declared in `NSLayoutManager.h`.

`NSTypesetterBehavior_10_3`

The typesetter behavior introduced in Mac OS X version 10.3.

Available in Mac OS X v10.3 and later.

Declared in `NSLayoutManager.h`.

`NSTypesetterBehavior_10_4`

The typesetter behavior introduced in Mac OS X version 10.4.

Available in Mac OS X v10.4 and later.

Declared in `NSLayoutManager.h`.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`NSLayoutManager.h`

Document Revision History

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

Date	Notes
2008-12-20	Added descriptions of NSGlyphStorage protocol methods.
2008-10-15	Added note to introduction discussing use of screen fonts. Augmented information about thread safety.
2007-04-16	Documented methods and constants added in Mac OS X v10.5. Added missing NSTypesetterBehavior_10_4 enumeration constant. Revised task groupings and corrected other minor errors.
2006-12-05	Removed references to Postscript commands from description of the showPackedGlyphs:length:glyphRange:atPoint: font:color:printingAdjustment: method.
2006-06-28	Made minor changes to conform to reference consistency guidelines.
2006-05-23	First publication of this content as a separate document.

Index