NSScrollView

Inherits From: NSView : NSResponder : NSObject

Conforms To: NSCoding (NSResponder)
NSObject (NSObject)

Declared In: AppKit/NSScrollView.h

Class at a Glance

Purpose
An NSScrollView allows the user to scroll a document view that's too large to display in its entirety. In addition to the document view, it displays horizontal and vertical scrollers and rulers (depending on which it's configured to have).

Principal Attributes
- Configurable scrollers
- Configurable rulers
- Displays a special cursor over its document view
- Small and large increment scrolling
- Dynamic (continuous) scrolling

Creation
Interface Builder
– initWithFrame: Designated initializer.

Commonly Used Methods
– setDocumentView: Sets the cursor used over the document view.
– setLineScroll: Sets the amount by which the document view moves during scrolling.
– setRulersVisible: Displays or hides rulers.

Class Description

The NSScrollView class is the central coordinator for the Application Kit's scrolling machinery, composed of this class, NSClipView, and NSScroller. An NSScrollView displays a portion of a *document view* that's too large to be displayed whole, and provides NSScrollers that allow the user to move the document view within the NSScrollView. An NSScrollView can be configured with a vertical scroller, a horizontal scroller,

or both. In addition to the basic accoutrements, an NSScrollView keeps a cursor that it sets whenever the mouse is over its document view, and maintains both horizontal and vertical ruler objects that can be hidden and displayed.

An NSScrollView encloses its document view within an NSClipView, using this view to actually position and monitor the document view. Because the NSClipView manages the content of the NSScrollView, it's also called the *content view*. The content view positions the document view by altering its bounds rectangle, which determines where the document view's frame lies. The content view also monitors changes in the document view's size and notifies the NSScrollView so that the scrollers can be updated to reflect the new size. The **documentView** and **contentView** methods return an NSScrollView's major component views.

NSScrollView defines three levels of scrolling: by line, by page, and direct. Scrolling by line moves the document view by a small amount, typically when the user clicks the scroll buttons of a scroller. Scrolling by page moves the document view by a larger amount, typically near the size of the content view, when the user Alternate-clicks the scroll buttons, and on some platforms in the slot of the scroller. You set these amounts using **setLineScroll:** and **setPageScroll:**, respectively (Interface Builder also lets you set these directly). Direct scrolling moves the document view to the position of the scroller's knob as the user drags it. This either displays the document view continuously as it scrolls or displays it only when the user releases the mouse, as configured with the **setScrollsDynamically:** method.

When created programmatically, an NSScrollView has no scrollers. You can set them up using the **setHasVerticalScroller:** and **setHasHorizontalScroller:** methods with an argument of YES, which cause the NSScrollView to allocate and maintain instances of the NSScroller class. You can substitute specialized scrollers using the **setVerticalScroller:** and **setHorizontalScroller:** methods. Note that in any case you must use the **setHas...** methods to make sure the NSScrollView displays its scrollers.

Rulers

An NSScrollView can be set to hold both horizontal and vertical rulers using the **setHasHorizontalRuler:** and **setHasVerticalRuler:** methods. These allocate instances of the NSRulerView class, but unlike with scrollers don't immediately display the rulers. To do this, use the **setRulersVisible:** method. You can substitute custom ruler objects using **setHorizontalRuler:** and **setVerticalRuler:**, and to customize the rulers for all instances of NSScrollView you can set the class used with **setRulerViewClass:**. This causes all subsequent rulers created by NSScrollViews to be of the class you specify.

An NSScrollView's rulers don't automatically establish the document view as their client. The document view itself (or a subview) is responsible, as its selection and other state changes, for retrieving the rulers using NSScrollView's **horizontalRuler** and **verticalRuler** methods and for establishing itself as the client using NSRulerView's **setClientView:** method.

How Scrolling Works

As indicated above, an NSScrollView's document view is actually positioned by the content view, which sets its bounds rectangle in such a way that the document view's frame moves relative to it. However, the

action sequence between the scrollers and the NSScrollView and the manner in which scrolling is performed involve a bit more detail than this.

Scrolling typically occurs because of user actions on an NSScroller object, which sends the NSScrollView a private action message telling it to scroll based on the NSScroller's state. This process is described in the class description for the NSScroller class under "Interaction with a Container View." If you plan to implement your own kind of scrolling view or scroller object, you should read that section.

NSClipView's **scrollToPoint:** is the method that actually scrolls the document view. It essentially translates the origin of the content view's bounds rectangle, but it also optimizes redisplay by copying as much of the rendered document view as remains visible, and only asking the document view to draw newly exposed regions. This usually improves scrolling performance, but may not always be appropriate behavior. You can turn it off using NSClipView's **setCopiesOnScroll:** method. If you do leave copy-on-scroll active, be sure to scroll the document view programmatically using **scrollToPoint:** rather than **translateOriginToPoint:**.

Whether the document view scrolls explicitly through a user action or an NSClipView message, or implicitly through a **setFrame:** or other such message, the content view monitors it closely. Whenever the document view's frame or bounds rectangle changes, it informs the NSScrollView of the change with a **reflectScrolledClipView:** message. This method updates the NSScroller objects to reflect the position and size of the visible portion of the document view. You may find on occasion that you need to invoke this method explicitly when manipulating the document view directly.

Autoscrolling

In addition to user-driven and programmatic scrolling, you can program any NSView to automatically scroll when the user drags the mouse outside the enclosing NSClipView. This allows the user to drag an item in order to move it, and have the document view automatically shift itself in the appropriate direction when the user drags the item past the visible area. NSClipView's **autoscroll:** method takes an NSEvent object of the mouse-dragged type and scrolls its document view in the opposite direction from the mouse location, making the portion of the document view that would be under the mouse become visible. NSView also implements **autoscroll:** to forward the message to its superview. This allows any NSView to simply send the message to itself during a mouse-dragging loop without checking whether it's contained in an NSClipView (though it does need to check whether the mouse is outside of its visible portion, as returned by **visibleRect**).

Method Types

Calculating layout	+ contentSizeForFrameSize:hasHorizontalScroller: hasVerticalScroller:borderType: + frameSizeForContentSize:hasHorizontalScroller: hasVerticalScroller:borderType:

Determining component sizes	– contentSize – documentVisibleRect
Managing graphic attributes	– setBackgroundColor: – backgroundColor – setBorderType: – borderType
Managing the scrolled views	– setContentView: – contentView – setDocumentView: – documentView – setDocumentCursor:
Managing scrollers	– setHorizontalScroller: – horizontalScroller – setHasHorizontalScroller: – hasHorizontalScroller – setVerticalScroller: – verticalScroller – setHasVerticalScroller: – hasVerticalScroller
Managing rulers	+ setRulerViewClass: + rulerViewClass – setHasHorizontalRuler: – hasHorizontalRuler – setHorizontalRulerView: – horizontalRulerView – setHasVerticalRuler: – hasVerticalRuler – setVerticalRulerView: – verticalRulerView – setRulersVisible: – rulersVisible
Setting scrolling behavior	– setLineScroll: – lineScroll – setPageScroll: – pageScroll – setScrollsDynamically: – scrollsDynamically
Updating display after scrolling	– reflectScrolledClipView:
Arranging components	– tile

Class Methods

contentSizeForFrameSize:hasHorizontalScroller:hasVerticalScroller:borderType:

+ (NSSize)**contentSizeForFrameSize:**(NSSize)*frameSize*
hasHorizontalScroller:(BOOL)*hFlag*
hasVerticalScroller:(BOOL)*vFlag*
borderType:(NSBorderType)*borderType*

Returns the size of a content view for an NSScrollView whose frame size is *frameSize*. *hFlag* and *vFlag* indicate whether a horizontal or vertical scroller, respectively, is present. If the flag is YES then the content size is reduced in the appropriate dimension by the width of an NSScroller. *borderType* indicates the appearance of the NSScrollView's edge, which also affects the content size; see the description of **setBorderType:** for a list of possible values.

For an existing NSScrollView, you can simply use the **contentSize** method.

See also: + **frameSizeForContentSize:hasHorizontalScroller:hasVerticalScroller:borderType:**,
+ **scrollerWidth** (NSScroller)

frameSizeForContentSize:hasHorizontalScroller:hasVerticalScroller:borderType:

+ (NSSize)**frameSizeForContentSize:**(NSSize)*contentSize*
hasHorizontalScroller:(BOOL)*hFlag*
hasVerticalScroller:(BOOL)*vFlag*
borderType:(NSBorderType)*borderType*

Returns the frame size of an NSScrollView that contains a content view whose size is *contentSize*. *hFlag* and *vFlag* indicate whether a horizontal or vertical scroller, respectively, is present. If the flag is YES then the frame size is increased in the appropriate dimension by the width of an NSScroller. *borderType* indicates the appearance of the NSScrollView's edge, which also affects the frame size; see the description of **setBorderType:** for a list of possible values.

For an existing NSScrollView, you can simply use the **frame** method and extract its size.

See also: + **contentSizeForFrameSize:hasHorizontalScroller:hasVerticalScroller:borderType:**,
+ **scrollerWidth** (NSScroller)

rulerViewClass

+ (Class)**rulerViewClass**

Returns the default class to be used for ruler objects in NSScrollViews. This is normally NSRulerView.

See also: + **setRulerViewClass:**

setRulerViewClass:

+ (void)**setRulerViewClass:**(Class)*aClass*

Sets the default class to be used for ruler objects in NSScrollViews to *aClass*. This is normally NSRulerView, but you can use this method to set it to a custom subclass of NSRulerView.

Note: This method simply sets a global variable private to NSScrollView.Subclasses of NSScrollView should override both this method and **rulerViewClass** to store their ruler view classes in private variables.

See also: + **rulerViewClass**

Instance Methods

backgroundColor

– (NSColor *)**backgroundColor**

Returns the content view's background color.

See also: – **setBackgroundColor:**, – **backgroundColor** (NSClipView)

borderType

– (NSBorderType)**borderType**

Returns a value that represents the type of border surrounding the receiver; see the description of **setBorderType:** for a list of possible values.

contentSize

– (NSSize)**contentSize**

Returns the size of the receiver's content view.

See also: + **contentSizeForFrameSize:hasHorizontalScroller:hasVerticalScroller:borderType:**

contentView

– (NSClipView *)**contentView**

Returns the receiver's content view, the view that clips the document view.

See also: – **setContentView:**, – **documentView**

documentCursor

– (NSCursor *)**documentCursor**

Returns the content view's document cursor.

See also: – **setDocumentCursor:**, – **documentCursor** (NSClipView)

documentView

– (id)**documentView**

Returns the view that the receiver scrolls within its content view.

See also: – **setDocumentView:**, – **documentView** (NSClipView)

documentVisibleRect

– (NSRect)**documentVisibleRect**

Returns the portion of the document view, in its own coordinate system, that's visible through the receiver's content view.

See also: – **documentVisibleRect** (NSClipView), – **visibleRect** (NSView)

hasHorizontalRuler

– (BOOL)**hasHorizontalRuler**

Returns YES if the receiver maintains a horizontal ruler view, NO if it doesn't. Display of rulers is controlled using the **setRulersVisible:** method.

See also: – **horizontalRulerView**, – **setHasHorizontalRuler:**, – **hasVerticalRuler**, + **rulerClass:**

hasHorizontalScroller

– (BOOL)**hasHorizontalScroller**

Returns YES if the receiver displays a horizontal scroller, NO if it doesn't.

See also: – **horizontalScroller**, – **setHasHorizontalScroller:**, – **hasVerticalScroller**

hasVerticalRuler

– (BOOL)**hasVerticalRuler**

Returns YES if the receiver maintains a vertical ruler view, NO if it doesn't. Display of rulers is controlled using the **setRulersVisible:** method.

See also: – **verticalRulerView**, – **setHasVerticalRuler:**, – **hasHorizontalRuler**, + **rulerClass:**

hasVerticalScroller

– (BOOL)**hasVerticalScroller**

Returns YES if the receiver displays a vertical scroller, NO if it doesn't.

See also: – **verticalScroller**, – **setHasVerticalScroller:**, – **hasHorizontalScroller**

horizontalRulerView

– (NSRulerView *)**horizontalRulerView**

Returns the receiver's horizontal ruler view, whether or not the receiver is currently displaying it, or **nil** if the receiver has none. If the receiver is set to display a horizontal ruler view and doesn't yet have one, this method creates an instance of the ruler view class set using the class method **setRulerViewClass:**. Display of rulers is controlled using the **setRulersVisible:** method.

See also: – **hasHorizontalRulerView**, – **verticalRulerView**

horizontalScroller

– (NSScroller *)**horizontalScroller**

Returns the receiver's horizontal scroller, whether or not the receiver is currently displaying it, or **nil** if the receiver has none.

See also: – **hasHorizontalScroller**, – **setHorizontalScroller:**, – **verticalScroller**

lineScroll

– (float)**lineScroll**

Returns the amount by which the receiver scrolls itself when scrolling line-by-line, expressed in the content view's coordinate system. This amount is used when the user clicks the scroll arrows without holding a modifier key.

See also: – **setLineScroll:**, – **pageScroll**

pageScroll

– (float)**pageScroll**

Returns the amount of the document view kept visible when scrolling page-by-page, expressed in the content view's coordinate system. This amount is used when the user clicks the scroll arrows while holding the Alternate key.

Note: This amount expresses the context that remains when the receiver scrolls by one page, allowing the user to orient himself to the new display. It differs from the line scroll amount, which indicates how far the document view moves. The page scroll amount is the amount *common to* the content view before and after the document view is scrolled by one page.

See also: – **setLineScroll:**, – **pageScroll**

reflectScrolledClipView:

– (void)**reflectScrolledClipView:**(NSClipView *)*aClipView*

If *aClipView* is the receiver's content view, adjusts the receiver's scrollers to reflect the size and positioning of its document view. Does nothing if *aClipView* is any other view object (in particular, if it's an NSClipView that isn't the content view).

This method is invoked automatically during scrolling and when an NSClipView's relationship to its document view changes; you should rarely need to invoke it yourself, but may wish to override it for custom updating or other behavior.

See also: – **contentView**, – **documentView**

rulersVisible

– (BOOL)**rulersVisible**

Returns YES if the receiver was set to show rulers using **setRulersVisible:** (whether or not it has rulers at all), NO if it was set to hide them.

See also: – **hasHorizontalRuler**, – **hasVerticalRuler**

scrollsDynamically

– (BOOL)**scrollsDynamically**

Returns YES if the receiver redraws its document view while tracking the knob, NO if it redraws only when the scroller knob is released. NSScrollView scrolls dynamically by default.

See also: – **setScrollsDynamically:**

setBackgroundColor:

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

Sets the color of the content view's background to *aColor*. This color is used to paint areas inside the content view that aren't covered by the document view.

See also: – **backgroundColor**, – **setBackgroundColor:** (NSClipView)

setBorderType:

– (void)**setBorderType:**(NSBorderType)*borderType*

Sets the border type of the receiver to *borderType*, which may be one of:

NSNoBorder
NSLineBorder
NSBezelBorder
NSGrooveBorder

See also: – **borderType**

setContentView:

– (void)**setContentView:**(NSClipView *)*aView*

Sets the receiver's content view, the view that clips the document view, to *aView*. *aView*'s document view, if any, also becomes the receiver's document view, while the original content view's document view remains with it.

See also: – **contentView**, – **setDocumentView:**

setDocumentCursor:

– (void)**setDocumentCursor:**(NSCursor *)*aCursor*

Sets the cursor used when the mouse is over the content view to *aCursor*, by sending **setDocumentCursor:** to the content view.

See also: – **documentCursor**

setDocumentView:

– (void)**setDocumentView:**(NSView *)*aView*

Sets the receiver's document view to *aView*.

See also: – **documentView**, – **setDocumentView:** (NSClipView)

setHasHorizontalRuler:

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

Determines whether the receiver keeps a horizontal ruler object. If *flag* is YES, the receiver allocates a horizontal ruler the first time it's needed. Display of rulers is handled independently with the **setRulersVisible:** method.

See also: – **hasHorizontalRuler**, – **horizontalRuler**, – **setHasVerticalRuler:**

setHasHorizontalScroller:

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

Determines whether the receiver keeps a horizontal scroller. If *flag* is YES, the receiver allocates and displays a horizontal scroller as needed. An NSScrollView by default has neither a horizontal nor a vertical scroller.

See also: – **hasHorizontalScroller**, – **horizontalScroller**, – **setHasVerticalScroller:**

setHasVerticalRuler:

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

Determines whether the receiver keeps a vertical ruler object. If *flag* is YES, the receiver allocates a vertical ruler the first time it's needed. Display of rulers is handled independently with the **setRulersVisible:** method.

See also: – **hasVerticalRuler**, – **verticalRuler**, – **setHasHorizontalRuler:**, – **setRulersVisible:**

setHasVerticalScroller:

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

Determines whether the receiver keeps a vertical scroller. If *flag* is YES, the receiver allocates and displays a vertical scroller as needed. An NSScrollView by default has neither a vertical nor a horizontal scroller.

See also: – **hasVerticalScroller**, – **verticalScroller**, – **setHasHorizontalScroller:**

setHorizontalRulerView:

– (void)**setHorizontalRulerView:**(NSRulerView *)*aRulerView*

Sets the receiver's horizontal ruler view to *aRulerView*. You can use this method to override the default ruler class set using the class method **setRulerClass:**. Display of rulers is controlled using the **setRulersVisible:** method.

See also: – **horizontalRulerView:**,– **setHasHorizontalRuler:**, – **setVerticalRulerView:**, – **setRulersVisible:**

setHorizontalScroller:

– (void)**setHorizontalScroller:**(NSScroller *)*aScroller*

Sets the receiver's horizontal scroller to *aScroller*, establishing the appropriate target-action relationships between them. To make sure the scroller is visible, invoke the **setHasHorizontalScroller:** method with an argument of YES.

See also: – **horizontalScroller**, – **setVerticalScroller:**

setLineScroll:

– (void)**setLineScroll:**(float)*aFloat*

Sets the amount by which the receiver scrolls itself when scrolling line-by-line to *aFloat*, expressed in the content view's coordinate system. This is the amount used when the user clicks the scroll arrows without holding a modifier key. When displaying text in an NSScrollView, for example, you might set this to the height of a single line of text in the default font.

See also: – **lineScroll**, – **setPageScroll:**

setPageScroll:

– (void)**setPageScroll:**(float)*aFloat*

Sets the amount of the document view kept visible when scrolling page-by-page to *aFloat*, expressed in the content view's coordinate system. This amount is used when the user clicks the scroll arrows while holding the Alternate key.

Note: This amount expresses the context that remains when the receiver scrolls by one page, allowing the user to orient himself to the new display. It differs from the line scroll amount, which indicates how far the document view moves. The page scroll amount is the amount *common to* the content view

before and after the document view is scrolled by one page. Thus, setting the page scroll amount to 0.0 implies that the entire visible portion of the document view is replaced when a page scroll occurs.

See also: – **pageScroll**, – **setLineScroll:**

setRulersVisible:

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

Determines whether the receiver displays its rulers. If *flag* is YES, the receiver displays its rulers (creating them if needed). If *flag* is NO, the receiver doesn't display its rulers.

See also: – **rulersVisible**, – **hasHorizontalRuler**, – **hasVerticalRuler**

setScrollsDynamically:

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

Determines whether the receiver redraws its document view while scrolling continuously. If *flag* is YES it does, if *flag* is NO it redraws only when the scroller knob is released. NSScrollView scrolls dynamically by default.

See also: – **scrollsDynamically**

setVerticalRulerView:

– (void)**setVerticalRulerView:**(NSRulerView *)*aRulerView*

Sets the receiver's vertical ruler view to *aRulerView*. You can use this method to override the default ruler class set using the class method **setRulerClass:**. Display of rulers is controlled using the **setRulersVisible:** method.

See also: – **verticalRulerView:**, – **setHasVerticalRuler:**, – **setHorizontalRulerView:**, – **setRulersVisible:**

setVerticalScroller:

– (void)**setVerticalScroller:**(NSScroller *)*aScroller*

Sets the receiver's vertical scroller to *aScroller*, establishing the appropriate target-action relationships between them. To make sure the scroller is visible, invoke the **setHasVerticalScroller:** method with an argument of YES.

See also: – **verticalScroller**, – **setHorizontalScroller:**

tile

– (void)**tile**

Lays out the components of the receiver: the content view, the scrollers, and the ruler views. You rarely need to invoke this method, but subclasses may override it to manage additional components.

verticalRulerView

– (NSRulerView *)**verticalRulerView**

Returns the receiver's vertical ruler view, whether or not the receiver is currently displaying it, or **nil** if the receiver has none. If the receiver is set to display a vertical ruler view and doesn't yet have one, this method creates an instance of the ruler view class set using the class method **setRulerViewClass:**. Display of rulers is controlled using the **setRulersVisible:** method.

See also: – **hasVerticalRulerView**, – **horizontalRulerView**

verticalScroller

– (NSScroller *)**verticalScroller**

Returns the receiver's vertical scroller, whether or not the receiver is currently displaying it, or **nil** if the receiver has none.

See also: – **hasVerticalScroller**, – **setVerticalScroller:**, – **horizontalScroller**