NSView Class Reference

Developer

Contents

NSView Class Reference

Inherits from	NSResponder : NSObject
Conforms to	NSAnimatablePropertyContainer NSUserInterfaceItemIdentification NSDraggingDestination NSAppearanceCustomization NSCoding (NSResponder) NSObject (NSObject)
Framework	/System/Library/Frameworks/AppKit.framework
Availability	Available in OS X v10.0 and later.
Companion guide	View Programming Guide
Declared in	NSClipView.h NSLayoutConstraint.h NSMenuItem.h NSOpenGLView.h NSView.h
Related sample code	QTCoreVideo301 QuickLookSketch Sketch Sketch+Accessibility TreeView

Class at a Glance

The `NSView` class defines the basic drawing, event-handling, and printing architecture of an app. You typically do not use `NSView` objects directly. Instead, you use objects whose classes descend from `NSView` or you subclass `NSView` yourself and override its methods to implement the behavior you need. For any view object, there are many methods that you can use as-is.

Principal Attributes

- Event handling
- Integrated display to screen and printer
- Flexible coordinate systems
- Icon dragging

Commonly Used Methods

`frame` (page 103)

Returns the location and size of the `NSView` object.

`bounds` (page 60)

Returns the internal origin and size of the `NSView` object.

`setNeedsDisplay:` (page 175)

Marks the `NSView` object as needing to be redrawn

`window` (page 212)

Returns the `NSWindow` object that contains the `NSView` object.

`drawRect:` (page 96)

Draws the `NSView` object. (All subclasses must implement this method, but it's rarely invoked explicitly.)

Overview

The `NSView` class provides the infrastructure for drawing, printing, and handling events in your app. Instances of the `NSView` class (or one of its subclasses) are commonly known as view objects, or simply as views.

Views handle the presentation and interaction with your app's visible content. You arrange one or more views inside an `NSWindow` object, which acts as a wrapper for your content. A view object defines a rectangular region for drawing and receiving mouse events. Views handle other chores as well, including the dragging of icons and working with the `NSScrollView` class to support efficient scrolling.

Most of the functionality of the `NSView` class is automatically invoked by the Application Kit. Unless you're implementing a concrete subclass of `NSView` or working intimately with the content of the view hierarchy at runtime, you don't need to know much about this class's interface. See "Commonly Used Methods" (page 13) for methods you might use regardless.

For more information on how `NSView` instances handle event and action messages, see *Cocoa Event Handling Guide*. For more information on displaying tooltips and contextual menus, see "Displaying Contextual Menus" in *NSMenu Class Reference* and "Managing Tooltips" in *NSWindow Class Reference*.

Subclassing Notes

`NSView` is perhaps the most important class in the Application Kit when it comes to subclassing and inheritance. Most user-interface objects you see in a Cocoa application are objects that inherit from `NSView`. If you want to create an object that draws itself in a special way, or that responds to mouse clicks in a special way, you would create a custom subclass of `NSView` (or of a class that inherits from `NSView`). Subclassing `NSView` is such a common and important procedure that several technical documents describe how to both draw in custom subclasses and respond to events in custom subclasses. See *Cocoa Drawing Guide* and *Cocoa Event Handling Guide* (especially ""Handling Mouse Events"" and ""Mouse Events" in *Cocoa Event Handling Guide*").

Handling Events in Your Subclass

If you subclass `NSView` directly and handle specific types of events, the implementation of your event-related methods should generally not call `super`. Views inherit their event-handling capabilities from their `NSResponder` parent class. The default behavior for responders is to pass events up the responder chain, which is not the behavior you typically want if you handle events in a custom view. Therefore, you should not call `super` if your view implements any of the following methods and handles the event:

```
mouseDown:
mouseDragged:
mouseUp:
mouseMoved:
mouseEntered:
mouseExited:
rightMouseDragged:
rightMouseUp:
otherMouseDown:
otherMouseDragged:
otherMouseUp:
scrollWheel:
keyDown:
keyUp:
flagsChanged:
tabletPoint:
tabletProximity:
```

> **Note:** Because NSView changes the default behavior of the `rightMouseDown:` (page 151) method, you should call `super` when implementing that method in your custom subclasses.

If your view descends from a class other than NSView, call `super` to let the parent view handle any events that you do not.

Tasks

Creating Instances

– `initWithFrame:` (page 110)
Initializes and returns a newly allocated NSView object with a specified frame rectangle.

– `prepareForReuse` (page 135)
Restores the view to an initial state so that it can be reused.

Managing the View Hierarchy

– `superview` (page 192)
Returns the receiver's superview, or `nil` if it has none.

– `setSubviews:` (page 181)
Sets the receiver's subviews to the specified subviews.

– `subviews` (page 191)
Return the receiver's immediate subviews.

– `window` (page 212)
Returns the receiver's window object, or `nil` if it has none.

– `addSubview:` (page 44)
Adds a view to the receiver's subviews so it's displayed above its siblings.

– `addSubview:positioned:relativeTo:` (page 45)
Inserts a view among the receiver's subviews so it's displayed immediately above or below another view.

– `didAddSubview:` (page 85)
Overridden by subclasses to perform additional actions when subviews are added to the receiver.

– `removeFromSuperview` (page 145)
Unlinks the receiver from its superview and its window, removes it from the responder chain, and invalidates its cursor rectangles.

– `removeFromSuperviewWithoutNeedingDisplay` (page 146)

Unlinks the receiver from its superview and its window and removes it from the responder chain, but does not invalidate its cursor rectangles to cause redrawing.

– `replaceSubview:with:` (page 148)

Replaces one of the receiver's subviews with another view.

– `isDescendantOf:` (page 113)

Returns `YES` if the receiver is a subview of a given view or if it's identical to that view; otherwise, it returns `NO`.

– `opaqueAncestor` (page 132)

Returns the receiver's closest opaque ancestor (including the receiver itself).

– `ancestorSharedWithView:` (page 54)

Returns the closest ancestor shared by the receiver and a given view.

– `sortSubviewsUsingFunction:context:` (page 190)

Orders the receiver's immediate subviews using the specified comparator function.

– `viewDidMoveToSuperview` (page 202)

Informs the receiver that its superview has changed (possibly to `nil`).

– `viewDidMoveToWindow` (page 202)

Informs the receiver that it has been added to a new view hierarchy.

– `viewWillMoveToSuperview:` (page 205)

Informs the receiver that its superview is about to change to the specified superview (which may be `nil`).

– `viewWillMoveToWindow:` (page 205)

Informs the receiver that it's being added to the view hierarchy of the specified window object (which may be `nil`).

– `willRemoveSubview:` (page 212)

Overridden by subclasses to perform additional actions before subviews are removed from the receiver.

– `enclosingMenuItem` (page 97)

Returns the menu item containing the receiver or any of its superviews in the view hierarchy.

Modifying the Frame Rectangle

– `setFrame:` (page 167)

Sets the receiver's frame rectangle to the specified rectangle.

– `frame` (page 103)

Returns the receiver's frame rectangle, which defines its position in its superview.

– `setFrameOrigin:` (page 169)
Sets the origin of the receiver's frame rectangle to the specified point, effectively repositioning it within its superview.

– `setFrameSize:` (page 170)
Sets the size of the receiver's frame rectangle to the specified dimensions, resizing it within its superview without affecting its coordinate system.

– `setFrameRotation:` (page 169)
Sets the rotation of the receiver's frame rectangle to a specified degree value, rotating it within its superview without affecting its coordinate system.

– `frameRotation` (page 105)
Returns the angle, in degrees, of the receiver's frame relative to its superview's coordinate system.

Modifying the Bounds Rectangle

– `setBounds:` (page 159)
Sets the receiver's bounds rectangle.

– `bounds` (page 60)
Returns the receiver's bounds rectangle, which expresses its location and size in its own coordinate system.

– `setBoundsOrigin:` (page 160)
Sets the origin of the receiver's bounds rectangle to a specified point,

– `setBoundsSize:` (page 162)
Sets the size of the receiver's bounds rectangle to specified dimensions, inversely scaling its coordinate system relative to its frame rectangle.

– `setBoundsRotation:` (page 161)
Sets the rotation of the receiver's bounds rectangle to a specific degree value.

– `boundsRotation` (page 61)
Returns the angle, in degrees, of the receiver's bounds rectangle relative to its frame rectangle.

Managing the View's Layer

– `layer` (page 118)
Returns the Core Animation layer that the receiver uses as its backing store.

– `setLayer:` (page 172)
Sets the Core Animation layer that the receiver uses for layer-backing to the specified layer.

– `wantsLayer` (page 209)
Returns a Boolean value that indicates whether the receiver is using a layer as its backing store.

– `setWantsLayer:` (page 184)
Specifies whether the receiver and its subviews use a Core Animation layer as a backing store.

– `makeBackingLayer` (page 124)
Creates the view's backing layer.

– `layerContentsPlacement` (page 119)
Returns the current layer contents placement policy.

– `setLayerContentsPlacement:` (page 173)
Sets the view's layer contents placement policy.

– `layerContentsRedrawPolicy` (page 119)
Returns the view's layer contents redraw policy.

– `setLayerContentsRedrawPolicy:` (page 174)
Sets the receiver layer contents redraw policy.

– `setCanDrawSubviewsIntoLayer:` (page 163)
Sets whether the view incorporates content from its subviews into its own layer.

– `canDrawSubviewsIntoLayer` (page 64)
Returns a Boolean value indicating whether the view incorporates content from its subviews into its own layer

– `setLayerUsesCoreImageFilters:` (page 175)
Sets whether the view's layer uses Core Image filters and should therefore be rendered in process.

– `layerUsesCoreImageFilters` (page 120)
Returns a Boolean value indicating whether the view's layer uses Core Image filters.

Managing Layer-Related Properties

– `setFrameCenterRotation:` (page 168)
Rotates the frame of the receiver about the layer's position.

– `frameCenterRotation` (page 104)
Returns the receiver's rotation about the layer's position.

– `setAlphaValue:` (page 156)
Sets the opacity of the receiver.

– `alphaValue` (page 53)
Returns the opacity of the receiver

– `setBackgroundFilters:` (page 158)
An array of CoreImage filters that are applied to the receiver's background.

– `backgroundFilters` (page 56)
Returns the array of CoreImage filters that are applied to the receiver's background

– `setCompositingFilter:` (page 164)
Sets a CoreImage filter that is used to composite the receiver's contents with the background.

– `compositingFilter` (page 65)
Returns the CoreImage filter that is used to composite the receiver's contents with the background

– `setContentFilters:` (page 165)
Sets the array of CoreImage filters that are applied to the contents of the receiver and its sublayers.

– `contentFilters` (page 67)
Returns the array of CoreImage filters that are applied to the contents of the receiver and its sublayers.

– `setShadow:` (page 180)
Sets the shadow drawn by the receiver.

– `shadow` (page 186)
Returns the shadow drawn by the receiver

Drawing

– `wantsUpdateLayer` (page 211)
Returns a Boolean value indicating which drawing path the view takes when updating its contents.

– `updateLayer` (page 198)
Updates the view's content by modifying its underlying layer.

– `drawRect:` (page 96)
Overridden by subclasses to draw the receiver's image within the specified rectangle.

– `canDraw` (page 63)
Returns `YES` if drawing commands will produce any result, `NO` otherwise.

– `canDrawConcurrently` (page 63)
Returns whether the view's `drawRect:` method can be invoked on a background thread.

– `setCanDrawConcurrently:` (page 163)
Sets whether the view's `drawRect:` method can be invoked on a background thread.

– `visibleRect` (page 207)
Returns the portion of the receiver not clipped by its superviews.

– `shouldDrawColor` (page 188)

Returns `NO` if the receiver is being drawn in an `NSWindow` object (as opposed, for example, to being printed) and the window object can't store color; otherwise returns `YES`.

– `getRectsBeingDrawn:count:` (page 106)

Returns by indirection a list of non-overlapping rectangles that define the area the receiver is being asked to draw in `drawRect:` (page 96).

– `needsToDrawRect:` (page 128)

Returns whether the specified rectangle intersects any part of the area that the receiver is being asked to draw.

– `wantsDefaultClipping` (page 209)

Returns whether the Application Kit's default clipping provided to `drawRect:` (page 96) implementations is in effect.

– `bitmapImageRepForCachingDisplayInRect:` (page 59)

Returns a bitmap-representation object suitable for caching the specified portion of the receiver.

– `cacheDisplayInRect:toBitmapImageRep:` (page 62)

Draws the specified area of the receiver, and its descendants, into a provided bitmap-representation object.

Printing

– `print:` (page 138)

This action method opens the Print panel, and if the user chooses an option other than canceling, prints the receiver and all its subviews to the device specified in the Print panel.

– `beginPageInRect:atPlacement:` (page 59)

Called at the beginning of each page, this method sets up the coordinate system so that a region inside the receiver's bounds is translated to a specified location..

– `dataWithEPSInsideRect:` (page 84)

Returns EPS data that draws the region of the receiver within a specified rectangle.

– `dataWithPDFInsideRect:` (page 85)

Returns PDF data that draws the region of the receiver within a specified rectangle.

– `printJobTitle` (page 138)

Returns the receiver's print job title.

– `pageFooter` (page 132)

Returns a default footer string that includes the current page number and page count.

– `pageHeader` (page 132)
Returns a default header string that includes the print job title and date.

– `writeEPSInsideRect:toPasteboard:` (page 213)
Writes EPS data that draws the region of the receiver within a specified rectangle onto a pasteboard.

– `writePDFInsideRect:toPasteboard:` (page 213)
Writes PDF data that draws the region of the receiver within a specified rectangle onto a pasteboard.

– `drawPageBorderWithSize:` (page 95)
Allows applications that use the Application Kit pagination facility to draw additional marks on each logical page.

– `drawSheetBorderWithSize:` (page 97)
Allows applications that use the Application Kit pagination facility to draw additional marks on each printed sheet.

Pagination

– `heightAdjustLimit` (page 109)
Returns the fraction (from 0.0 to 1.0) of the page that can be pushed onto the next page during automatic pagination to prevent items such as lines of text from being divided across pages.

– `widthAdjustLimit` (page 211)
Returns the fraction (from 0.0 to 1.0) of the page that can be pushed onto the next page during automatic pagination to prevent items such as small images or text columns from being divided across pages.

– `adjustPageWidthNew:left:right:limit:` (page 49)
Overridden by subclasses to adjust page width during automatic pagination.

– `adjustPageHeightNew:top:bottom:limit:` (page 48)
Overridden by subclasses to adjust page height during automatic pagination.

– `knowsPageRange:` (page 118)
Returns `YES` if the receiver handles page boundaries, `NO` otherwise.

– `rectForPage:` (page 139)
Implemented by subclasses to determine the portion of the receiver to be printed for the page number page.

– `locationOfPrintRect:` (page 122)
Invoked by `print:` (page 138) to determine the location of the region of the receiver being printed on the physical page.

Invalidating the View's Content

– `setNeedsDisplay:` (page 175)
Controls whether the receiver's entire bounds is marked as needing display.

– `setNeedsDisplayInRect:` (page 176)
Marks the region of the receiver within the specified rectangle as needing display, increasing the receiver's existing invalid region to include it.

– `needsDisplay` (page 127)
Returns whether the view needs to be redrawn before being displayed.

– `display` (page 86)
Displays the receiver and all its subviews if possible, invoking each of the `NSView` methods `lockFocus` (page 122), `drawRect:` (page 96), and `unlockFocus` (page 196) as necessary.

– `displayRect:` (page 89)
Acts as `display` (page 86), but confining drawing to a rectangular region of the receiver.

– `displayRectIgnoringOpacity:` (page 89)
Displays the receiver but confines drawing to a specified region and does not back up to the first opaque ancestor—it simply causes the receiver and its descendants to execute their drawing code.

– `displayRectIgnoringOpacity:inContext:` (page 89)
Causes the receiver and its descendants to be redrawn to the specified graphics context.

– `displayIfNeeded` (page 87)
Displays the receiver and all its subviews if any part of the receiver has been marked as needing display.

– `displayIfNeededInRect:` (page 88)
Acts as `displayIfNeeded` (page 87), confining drawing to a specified region of the receiver..

– `displayIfNeededIgnoringOpacity` (page 87)
Acts as `displayIfNeeded` (page 87), except that this method doesn't back up to the first opaque ancestor—it simply causes the receiver and its descendants to execute their drawing code.

– `displayIfNeededInRectIgnoringOpacity:` (page 88)
Acts as `displayIfNeeded` (page 87), but confining drawing to `aRect` and not backing up to the first opaque ancestor—it simply causes the receiver and its descendants to execute their drawing code.

– `translateRectsNeedingDisplayInRect:by:` (page 195)
Translates the display rectangles by the specified delta.

– `isOpaque` (page 116)
Overridden by subclasses to return `YES` if the receiver is opaque, `NO` otherwise.

– `viewWillDraw` (page 203)
Informs the receiver that it will be required to draw content.

Converting Coordinate Values

– `backingAlignedRect:options:` (page 56)
Returns a backing store pixel aligned rectangle in window coordinates.

– `convertPointFromBacking:` (page 70)
Converts a point from its pixel aligned backing store coordinate system to the view's interior coordinate system.

– `convertPointToBacking:` (page 72)
Converts a point from the view's interior coordinate system to its pixel aligned backing store coordinate system.

– `convertPointFromLayer:` (page 71)
Convert the point from the layer's interior coordinate system to the view's interior coordinate system.

– `convertPointToLayer:` (page 73)
Convert the size from the view's interior coordinate system to the layer's interior coordinate system.

– `convertRectFromBacking:` (page 75)
Converts a rectangle from its pixel aligned backing store coordinate system to the view's interior coordinate system.

– `convertRectToBacking:` (page 77)
Converts a rectangle from the view's interior coordinate system to its pixel aligned backing store coordinate system.

– `convertRectFromLayer:` (page 77)
Convert the rectangle from the layer's interior coordinate system to the view's interior coordinate system.

– `convertRectToLayer:` (page 78)
Convert the size from the view's interior coordinate system to the layer's interior coordinate system.

– `convertSizeFromBacking:` (page 81)
Converts a size from its pixel aligned backing store coordinate system to the view's interior coordinate system.

– `convertSizeToBacking:` (page 82)
Converts a size from the view's interior coordinate system to its pixel aligned backing store coordinate system.

– `convertSizeFromLayer:` (page 82)
Convert the size from the layer's interior coordinate system to the view's interior coordinate system.

– `convertSizeToLayer:` (page 83)
Convert the size from the view's interior coordinate system to the layer's interior coordinate system.

– `convertPoint:fromView:` (page 69)
Converts a point from the coordinate system of a given view to that of the receiver.

– `convertPoint:toView:` (page 69)
Converts a point from the receiver's coordinate system to that of a given view.

– `convertSize:fromView:` (page 79)
Converts a size from another view's coordinate system to that of the receiver.

– `convertSize:toView:` (page 80)
Converts a size from the receiver's coordinate system to that of another view.

– `convertRect:fromView:` (page 74)
Converts a rectangle from the coordinate system of another view to that of the receiver.

– `convertRect:toView:` (page 75)
Converts a rectangle from the receiver's coordinate system to that of another view.

– `centerScanRect:` (page 64)
Converts the corners of a specified rectangle to lie on the center of device pixels, which is useful in compensating for rendering overscanning when the coordinate system has been scaled.

Modifying the Coordinate System

– `translateOriginToPoint:` (page 194)
Translates the receiver's coordinate system so that its origin moves to a new location.

– `scaleUnitSquareToSize:` (page 152)
Scales the receiver's coordinate system so that the unit square scales to the specified dimensions.

– `rotateByAngle:` (page 151)
Rotates the receiver's bounds rectangle by a specified degree value around the origin of the coordinate system, (0.0, 0.0).

Examining Coordinate System Modifications

– `isFlipped` (page 114)
Returns `YES` if the receiver uses flipped drawing coordinates or `NO` if it uses native coordinates.

– `isRotatedFromBase` (page 117)
Returns `YES` if the receiver or any of its ancestors has ever received a `setFrameRotation:` (page 169) or `setBoundsRotation:` (page 161) message; otherwise returns `NO`.

– `isRotatedOrScaledFromBase` (page 117)
Returns `YES` if the receiver or any of its ancestors has ever had a nonzero frame or bounds rotation, or has been scaled from the window's base coordinate system; otherwise returns `NO`.

Resizing Subviews

– `resizeSubviewsWithOldSize:` (page 150)
Informs the receiver's subviews that the receiver's bounds rectangle size has changed.

– `resizeWithOldSuperviewSize:` (page 150)
Informs the receiver that the bounds size of its superview has changed.

– `setAutoresizesSubviews:` (page 157)
Determines whether the receiver automatically resizes its subviews when its frame size changes.

– `autoresizesSubviews` (page 54)
Returns `YES` if the receiver automatically resizes its subviews using `resizeSubviewsWithOldSize:` (page 150) whenever its frame size changes, `NO` otherwise.

– `setAutoresizingMask:` (page 157)
Determines how the receiver's `resizeWithOldSuperviewSize:` (page 150) method changes its frame rectangle.

– `autoresizingMask` (page 54)
Returns the receiver's autoresizing mask, which determines how it's resized by the `resizeWithOldSuperviewSize:` (page 150) method.

Managing Constraints

– `constraints` (page 66)
Returns the constraints held by the view.

– `addConstraint:` (page 42)
Adds a constraint on the layout of the receiving view or its subviews.

– `addConstraints:` (page 43)
Adds multiple constraints on the layout of the receiving view or its subviews.

– `removeConstraint:` (page 144)
Removes the specified constraint from the view.

– `removeConstraints:` (page 144)
Removes the specified constraints from the view.

Measuring in Constraint-Based Layout

– `fittingSize` (page 101)
Returns the minimum size of the view that satisfies the constraints it holds.

– `intrinsicContentSize` (page 112)
Returns the natural size for the receiving view, considering only properties of the view itself.

– `invalidateIntrinsicContentSize` (page 112)
Invalidates the view's intrinsic content size.

– `contentCompressionResistancePriorityForOrientation:` (page 67)
Returns the priority with which a view resists being made smaller than its intrinsic size.

– `setContentCompressionResistancePriority:forOrientation:` (page 165)
Sets the priority with which a view resists being made smaller than its intrinsic size.

– `contentHuggingPriorityForOrientation:` (page 68)
Returns the priority with which a view resists being made larger than its intrinsic size.

– `setContentHuggingPriority:forOrientation:` (page 166)
Sets the priority with which a view resists being made larger than its intrinsic size.

Aligning Views with Constraint-Based Layout

– `alignmentRectForFrame:` (page 51)
Returns the view's alignment rectangle for a given frame.

– `frameForAlignmentRect:` (page 104)
Returns the view's frame for a given alignment rectangle.

– `alignmentRectInsets` (page 52)
Returns the insets from the view's frame that define its alignment rectangle.

– `baselineOffsetFromBottom` (page 57)
Returns the distance between the bottom of the view's alignment rectangle and the view's baseline.

Triggering Constraint-Based Layout

– `needsLayout` (page 127)
Returns whether the view needs a layout pass before it can be drawn.

– `setNeedsLayout:` (page 177)
Controls whether the view's subtree needs layout.

– `layout` (page 120)
Perform layout in concert with the constraint-based layout system.

– `layoutSubtreeIfNeeded` (page 121)
Updates the layout of the receiving view and its subviews based on the current views and constraints.

– `needsUpdateConstraints` (page 129)
Returns whether the view's constraints need updating.

– `setNeedsUpdateConstraints:` (page 178)
Controls whether the view's constraints need updating.

– `updateConstraints` (page 197)
Update constraints for the view.

– `updateConstraintsForSubtreeIfNeeded` (page 198)
Updates the constraints for the receiving view and its subviews.

Opting in to Constraint-Based Layout

+ `requiresConstraintBasedLayout` (page 40)
Returns whether the receiver depends on the constraint-based layout system.

– `translatesAutoresizingMaskIntoConstraints` (page 196)
Returns a Boolean value that indicates whether the view's autoresizing mask is translated into constraints for the constraint-based layout system.

– `setTranslatesAutoresizingMaskIntoConstraints:` (page 182)
Sets whether the view's autoresizing mask should be translated into constraints for the constraint-based layout system.

Debugging Constraint-Based Layout

See Auto Layout Guide for more details on debugging constraint-based layout.

– `constraintsAffectingLayoutForOrientation:` (page 66)
Returns the constraints impacting the layout of the view for a given orientation.

– `hasAmbiguousLayout` (page 108)
Returns whether the constraints impacting the layout of the view incompletely specify the location of the view.

– `exerciseAmbiguityInLayout` (page 100)
Randomly changes the frame of a view with an ambiguous layout between the different valid values.

Managing the Graphics State

– `allocateGState` (page 52)
Causes the receiver to maintain a private graphics state object, which encapsulates all parameters of the graphics environment.

– `gState` (page 107)
Returns the identifier for the receiver's graphics state object, or 0 if the receiver doesn't have a graphics state object.

– `setUpGState` (page 183)
Overridden by subclasses to (re)initialize the receiver's graphics state object.

– `renewGState` (page 148)
Invalidates the receiver's graphics state object, if it has one.

– `releaseGState` (page 143)
Frees the receiver's graphics state object, if it has one.

Focusing

– `lockFocus` (page 122)
Locks the focus on the receiver, so subsequent commands take effect in the receiver's window and coordinate system.

– `lockFocusIfCanDraw` (page 123)
Locks the focus to the receiver atomically if the `canDraw` method returns `YES` and returns the value of `canDraw`.

– `lockFocusIfCanDrawInContext:` (page 123)
Locks the focus to the receiver atomically if drawing can occur in the specified graphics context.

– `unlockFocus` (page 196)
Unlocks focus from the current view.

+ `focusView` (page 39)
Returns the currently focused `NSView` object, or `nil` if there is none.

Focus Ring Drawing

– `drawFocusRingMask` (page 94)
Draws the focus ring mask for the view.

– `focusRingMaskBounds` (page 102)
Returns the focus ring mask bounds.

– `noteFocusRingMaskChanged` (page 131)
Invoked to notify the view that the focus ring mask requires updating.

– `setKeyboardFocusRingNeedsDisplayInRect:` (page 172)
Invalidates the area around the focus ring.

+ `defaultFocusRingType` (page 38)
Returns the default focus ring type.

– `setFocusRingType:` (page 166)
Sets the type of focus ring to be drawn around the receiver.

– `focusRingType` (page 103)
Returns the type of focus ring drawn around the receiver.

Fullscreen Mode

– `enterFullScreenMode:withOptions:` (page 99)
Sets the receiver to full screen mode.

– `exitFullScreenModeWithOptions:` (page 101)
Instructs the receiver to exit full screen mode.

– `isInFullScreenMode` (page 116)
Returns whether the view is in full screen mode.

Hiding Views

– `setHidden:` (page 171)
Sets whether the view is hidden.

– `isHidden` (page 114)
Returns whether the receiver is marked as hidden.

– `isHiddenOrHasHiddenAncestor` (page 115)
Returns `YES` if the receiver is marked as hidden or has an ancestor in the view hierarchy that is marked as hidden; returns `NO` otherwise.

– `viewDidHide` (page 201)
Invoked when the receiver is hidden, either directly, or in response to an ancestor being hidden.

– `viewDidUnhide` (page 203)
Invoked when the receiver is unhidden, either directly, or in response to an ancestor being unhidden

Managing Live Resize

– `inLiveResize` (page 111)
A convenience method, expected to be called from `drawRect:` (page 96), to assist in decisions about optimized drawing.

– `preservesContentDuringLiveResize` (page 136)
Returns `YES` if the view supports the optimization of live-resize operations by preserving content that has not moved; otherwise, returns `NO`.

– `getRectsExposedDuringLiveResize:count:` (page 106)
Returns a list of rectangles indicating the newly exposed areas of the receiver.

– `rectPreservedDuringLiveResize` (page 141)
Returns the rectangle identifying the portion of your view that did not change during a live resize operation.

– `viewWillStartLiveResize` (page 206)
Informs the receiver of the start of a live resize.

– `viewDidEndLiveResize` (page 201)
Informs the receiver of the end of a live resize.

Event Handling

– `acceptsFirstMouse:` (page 41)
Overridden by subclasses to return `YES` if the receiver should be sent a `mouseDown:` message for an initial mouse-down event, `NO` if not.

– `hitTest:` (page 109)
Returns the farthest descendant of the receiver in the view hierarchy (including itself) that contains a specified point, or `nil` if that point lies completely outside the receiver.

– `mouse:inRect:` (page 125)
Returns whether a region of the receiver contains a specified point, accounting for whether the receiver is flipped or not.

– `performKeyEquivalent:` (page 133)
Implemented by subclasses to respond to key equivalents (also known as keyboard shortcuts).

– `rightMouseDown:` (page 151)

Informs the receiver that the user has pressed the right mouse button.

– `mouseDownCanMoveWindow` (page 126)

Returns `YES` if the receiver does not need to handle a mouse down and can pass it through to superviews; `NO` if it needs to handle the mouse down.

– `inputContext` (page 111)

Returns the text input context object for the receiver.

Touch Event Handling

– `acceptsTouchEvents` (page 41)

Returns whether the view will accept touch events.

– `setAcceptsTouchEvents:` (page 156)

Sets whether the view should accept touch events.

– `wantsRestingTouches` (page 210)

Returns whether the view wants resting touches.

– `setWantsRestingTouches:` (page 186)

Sets whether the view wants to receive resting touch events.

Key-view Loop Management

– `canBecomeKeyView` (page 63)

Returns whether the receiver can become key view.

– `needsPanelToBecomeKey` (page 128)

Overridden by subclasses to determine if the receiver requires its panel, which might otherwise avoid becoming key, to become the key window so that it can handle keyboard input and navigation.

– `setNextKeyView:` (page 178)

Inserts a specified view object after the receiver in the key view loop of the receiver's window.

– `nextKeyView` (page 129)

Returns the view object following the receiver in the key view loop.

– `nextValidKeyView` (page 130)

Returns the closest view object in the key view loop that follows the receiver and accepts first responder status.

– `previousKeyView` (page 137)
Returns the view object preceding the receiver in the key view loop.

– `previousValidKeyView` (page 137)
Returns the closest view object in the key view loop that precedes the receiver and accepts first responder status.

Scrolling

+ `isCompatibleWithResponsiveScrolling` (page 40)
Returns a Boolean value indicating whether the view supports responsive scrolling.

– `prepareContentInRect:` (page 135)
Prepares the overdraw region for drawing.

`preparedContentRect` (page 38) *property*
The portion of the view that has been rendered and is available for responsive scrolling.

– `scrollPoint:` (page 154)
Scrolls the receiver's closest ancestor `NSClipView` object so a point in the receiver lies at the origin of the clip view's bounds rectangle.

– `scrollRectToVisible:` (page 155)
Scrolls the receiver's closest ancestor `NSClipView` object the minimum distance needed so a specified region of the receiver becomes visible in the clip view.

– `autoscroll:` (page 55)
Scrolls the receiver's closest ancestor `NSClipView` object proportionally to the distance of an event that occurs outside of it.

– `adjustScroll:` (page 50)
Overridden by subclasses to modify a given rectangle, returning the altered rectangle.

– `scrollRect:by:` (page 154)
Copies the visible portion of the receiver's rendered image within a region and lays that portion down again at a specified offset .

– `enclosingScrollView` (page 98)
Returns the nearest ancestor `NSScrollView` object containing the receiver (not including the receiver itself); otherwise returns `nil`.

– `scrollClipView:toPoint:` (page 153)
Notifies the superview of a clip view that the clip view needs to reset the origin of its bounds rectangle.

– `reflectScrolledClipView:` (page 141)

Notifies a clip view's superview that either the clip view's bounds rectangle or the document view's frame rectangle has changed, and that any indicators of the scroll position need to be adjusted.

Dragging Operations

– `registeredDraggedTypes` (page 142)

Returns the array of pasteboard drag types that the view can accept.

– `registerForDraggedTypes:` (page 142)

Registers the pasteboard types that the receiver will accept as the destination of an image-dragging session.

– `unregisterDraggedTypes` (page 197)

Unregisters the receiver as a possible destination in a dragging session.

– `beginDraggingSessionWithItems:event:source:` (page 58)

Initiates a dragging session with a group of dragging items.

– `dragFile:fromRect:slideBack:event:` (page 90)

Initiates a dragging operation from the receiver, allowing the user to drag a file icon to any application that has window or view objects that accept files.

– `dragImage:at:offset:event:pasteboard:source:slideBack:` (page 91)

Initiates a dragging operation from the receiver, allowing the user to drag arbitrary data with a specified icon into any application that has window or view objects that accept dragged data.

– `dragPromisedFilesOfTypes:fromRect:source:slideBack:event:` (page 93)

Initiates a dragging operation from the receiver, allowing the user to drag one or more promised files (or directories) into any application that has window or view objects that accept promised file data.

– `shouldDelayWindowOrderingForEvent:` (page 187)

Overridden by subclasses to allow the user to drag images from the receiver without its window moving forward and possibly obscuring the destination and without activating the application.

Handling Smart Magnification

– `rectForSmartMagnificationAtPoint:inRect:` (page 140)

Returns the appropriate rectangle to use when magnifying around the specified point.

Controlling Notifications

– `setPostsFrameChangedNotifications:` (page 180)
Controls whether the receiver informs observers when its frame rectangle changes.

– `postsFrameChangedNotifications` (page 134)
Returns `YES` if the receiver posts notifications to the default notification center whenever its frame rectangle changes; returns `NO` otherwise.

– `setPostsBoundsChangedNotifications:` (page 179)
Controls whether the receiver informs observers when its bounds rectangle changes.

– `postsBoundsChangedNotifications` (page 134)
Returns `YES` if the receiver posts notifications to the default notification center whenever its bounds rectangle changes; returns `NO` otherwise.

Responding to Changes in Backing Store Properties

– `viewDidChangeBackingProperties` (page 200)
Invoked when the receiver's backing store properties change.

Searching by Tag

– `viewWithTag:` (page 207)
Returns the receiver's nearest descendant (including itself) with a specific tag, or `nil` if no subview has that tag.

– `tag` (page 193)
Returns the receiver's tag, an integer that you can use to identify view objects in your application.

Tool Tips

– `addToolTipRect:owner:userData:` (page 46)
Creates a tool tip for a defined area the receiver and returns a tag that identifies the tool tip rectangle.

– `removeAllToolTips` (page 143)
Removes all tool tips assigned to the receiver.

– `removeToolTip:` (page 146)
Removes the tool tip identified by specified tag.

– `setToolTip:` (page 182)
Sets the tool tip text for the view to `string`.

– `toolTip` (page 193)
Returns the text for the view's tool tip.

Managing Tracking Rectangles

– `addTrackingRect:owner:userData:assumeInside:` (page 47)
Establishes an area for tracking mouse-entered and mouse-exited events within the receiver and returns a tag that identifies the tracking rectangle.

– `removeTrackingRect:` (page 147)
Removes the tracking rectangle identified by a tag.

Managing Tracking Areas

– `addTrackingArea:` (page 47)
Adds a given tracking area to the receiver.

– `removeTrackingArea:` (page 147)
Removes a given tracking area from the receiver.

– `trackingAreas` (page 194)
Returns an array of the receiver's tracking areas.

– `updateTrackingAreas` (page 199)
Invoked automatically when the view's geometry changes such that its tracking areas need to be recalculated.

Managing Cursor Tracking

– `addCursorRect:cursor:` (page 43)
Establishes the cursor to be used when the mouse pointer lies within a specified region.

– `removeCursorRect:cursor:` (page 144)
Completely removes a cursor rectangle from the receiver.

– `discardCursorRects` (page 86)
Invalidates all cursor rectangles set up using `addCursorRect:cursor:` (page 43).

– `resetCursorRects` (page 149)
Overridden by subclasses to define their default cursor rectangles.

Contextual Menus

– `menuForEvent:` (page 125)
Overridden by subclasses to return a context-sensitive pop-up menu for a given mouse-down event.

+ `defaultMenu` (page 39)
Overridden by subclasses to return the default pop-up menu for instances of the receiving class.

Writing Conforming Rendering Instructions

– `beginDocument` (page 57)
Invoked at the beginning of the printing session, this method sets up the current graphics context.

– `endDocument` (page 98)
This method is invoked at the end of the printing session.

– `endPage` (page 99)
Writes the end of a conforming page.

Displaying Definition Windows

– `showDefinitionForAttributedString:atPoint:` (page 188)
Shows a window displaying the definition of the of the attributed string at the specified point.

– `showDefinitionForAttributedString:range:options:baselineOriginProvider:` (page 189)
Shows a window displaying the definition of the specified range of the attributed string.

Drawing Find Indicator

– `isDrawingFindIndicator` (page 113)
Returns when the view or one of its ancestors is being drawn for a find indicator.

Managing the Content Layout Direction

– `userInterfaceLayoutDirection` (page 200)
Returns the layout direction for content in the view.

– `setUserInterfaceLayoutDirection:` (page 183)
Sets the preferred layout direction for the view's content.

Specifying the OpenGL Surface Resolution

– `wantsBestResolutionOpenGLSurface` (page 208)
Returns a Boolean value indicating whether the view wants an OpenGL backing surface with resolution greater than 1 pixel per point.

– `setWantsBestResolutionOpenGLSurface:` (page 184)
Sets whether the view supports a high resolution OpenGL backing surface.

Deprecated Methods

– `convertPointToBase:` (page 73)
Converts the point from the receiver's coordinate system to the base coordinate system. (Deprecated. Use `convertPointToBacking:` (page 72) instead.)

– `convertPointFromBase:` (page 71)
Converts the point from the base coordinate system to the receiver's coordinate system. (Deprecated. Use `convertPointFromBacking:` (page 70) instead.)

– `convertSizeToBase:` (page 83)
Converts the size from the receiver's coordinate system to the base coordinate system. (Deprecated. Use `convertSizeToBacking:` (page 82) instead.)

– `convertSizeFromBase:` (page 81)
Converts the size from the base coordinate system to the receiver's coordinate system. (Deprecated. Use `convertSizeFromBacking:` (page 81) instead.)

– `convertRectToBase:` (page 78)
Converts the rectangle from the receiver's coordinate system to the base coordinate system. (Deprecated. Use `convertRectToBacking:` (page 77) instead.)

– `convertRectFromBase:` (page 76)
Converts the rectangle from the base coordinate system to the receiver's coordinate system. (Deprecated. Use `convertRectFromBacking:` (page 75) instead.)

– `performMnemonic:` (page 226) Deprecated in OS X v10.8

Implemented by subclasses to respond to mnemonics.

Properties

preparedContentRect

The portion of the view that has been rendered and is available for responsive scrolling.

```
@property NSRect preparedContentRect;
```

Discussion

During responsive scrolling, this property specifies the portion of the view that has been rendered and is ready to scroll. This rectangle always includes the visible portion of the view and may also include nonvisible portions that have been rendered and cached.

Changing the value of this property alerts AppKit that it might need to generate new overdraw content. For example, setting the value to the current visible rectangle forces AppKit to throw away any cached overdraw content and regenerate it during the next idle period. Never assign a rectangle that is smaller than the visible rectangle.

Availability

Available in OS X v10.9 and later.

Declared in

`NSView.h`

Class Methods

defaultFocusRingType

Returns the default focus ring type.

```
+ (NSFocusRingType)defaultFocusRingType
```

Return Value

The default type of focus ring for objects of the receiver's class. Possible return values are listed in `NSFocusRingType`.

Discussion

If `NSFocusRingTypeDefault` is returned from the instance method `focusRingType` (page 103), the receiver can invoke this class method to find out what type of focus ring is the default. The receiver is free to ignore the default setting.

Availability

Available in OS X v10.3 and later.

Declared in

`NSView.h`

defaultMenu

Overridden by subclasses to return the default pop-up menu for instances of the receiving class.

```
+ (NSMenu *)defaultMenu
```

Discussion

The default implementation returns `nil`.

Availability

Available in OS X v10.0 and later.

See Also

– `menuForEvent:` (page 125)
`menu` (`NSResponder`)

Declared in

`NSView.h`

focusView

Returns the currently focused `NSView` object, or `nil` if there is none.

```
+ (NSView *)focusView
```

Availability

Available in OS X v10.0 and later.

See Also

– `lockFocus` (page 122)
– `unlockFocus` (page 196)

Declared in
NSView.h

isCompatibleWithResponsiveScrolling

Returns a Boolean value indicating whether the view supports responsive scrolling.

```
+ (BOOL)isCompatibleWithResponsiveScrolling
```

Return Value
YES if the view supports responsive scrolling or NO if it does not.

Discussion
The default implementation of this method returns YES unless the class overrides the lockFocus (page 122) or scrollWheel: method. Subclasses such as NSScrollView and NSClipView override this method and perform additional checks.

AppKit enables responsive scrolling when the views involved in scrolling—the NSScrollView, NSClipView, and embedded document view—all return YES from this method. You can override this method in your custom views and return an appropriate value to reflect your view's support for the feature.

Availability
Available in OS X v10.9 and later.

Declared in
NSView.h

requiresConstraintBasedLayout

Returns whether the receiver depends on the constraint-based layout system.

```
+ (BOOL)requiresConstraintBasedLayout
```

Return Value
YES if the view must be in a window using constraint-based layout to function properly, NO otherwise.

Discussion
Custom views should override this to return YES if they can not layout correctly using autoresizing.

Availability
Available in OS X v10.7 and later.

Declared in
NSLayoutConstraint.h

Instance Methods

acceptsFirstMouse:

Overridden by subclasses to return YES if the receiver should be sent a mouseDown: message for an initial mouse-down event, NO if not.

```
- (BOOL)acceptsFirstMouse:(NSEvent *)theEvent
```

Parameters
theEvent
The initial mouse-down event, which must be over the receiver in its window.

Discussion
The receiver can either return a value unconditionally or use the location of theEvent to determine whether or not it wants the event. The default implementation ignores theEvent and returns NO.

Override this method in a subclass to allow instances to respond to click-through. This allows the user to click on a view in an inactive window, activating the view with one click, instead of clicking first to make the window active and then clicking the view. Most view objects refuse a click-through attempt, so the event simply activates the window. Many control objects, however, such as instances of NSButton and NSSlider, do accept them, so the user can immediately manipulate the control without having to release the mouse button.

Availability
Available in OS X v10.0 and later.

See Also
– hitTest: (page 109)

Declared in
NSView.h

acceptsTouchEvents

Returns whether the view will accept touch events.

```
- (BOOL)acceptsTouchEvents
```

Return Value
`YES` if the view accepts touch events, otherwise `NO`.

Discussion
The default is `NO`.

Availability
Available in OS X v10.6 and later.

See Also
– `setAcceptsTouchEvents:` (page 156)

Declared in
`NSView.h`

addConstraint:

Adds a constraint on the layout of the receiving view or its subviews.

```
– (void)addConstraint:(NSLayoutConstraint *)constraint
```

Parameters
`constraint`

The constraint to be added to the view. The constraint may only reference the view itself or its subviews.

Discussion
The constraint must involve only views that are within scope of the receiving view. Specifically, any views involved must be either the receiving view itself, or a subview of the receiving view. Constraints that are added to a view are said to be held by that view. The coordinate system used when evaluating the constraint is the coordinate system of the view that holds the constraint.

Availability
Available in OS X v10.7 and later.

Related Sample Code
InfoBarStackView

Declared in
`NSLayoutConstraint.h`

addConstraints:

Adds multiple constraints on the layout of the receiving view or its subviews.

```
- (void)addConstraints:(NSArray *)constraints
```

Parameters

`constraints`

An array of constraints to be added to the view. All constraints may only reference the view itself or its subviews.

Discussion

All constraints must involve only views that are within scope of the receiving view. Specifically, any views involved must be either the receiving view itself, or a subview of the receiving view. Constraints that are added to a view are said to be held by that view. The coordinate system used when evaluating each constraint is the coordinate system of the view that holds the constraint.

Availability

Available in OS X v10.7 and later.

Declared in

`NSLayoutConstraint.h`

addCursorRect:cursor:

Establishes the cursor to be used when the mouse pointer lies within a specified region.

```
- (void)addCursorRect:(NSRect)aRect cursor:(NSCursor *)aCursor
```

Parameters

`aRect`

A rectangle defining a region of the receiver.

`aCursor`

An object representing a cursor.

Discussion

Cursor rectangles aren't subject to clipping by superviews, nor are they intended for use with rotated views. You should explicitly confine a cursor rectangle to the view's visible rectangle to prevent improper behavior.

This method is intended to be invoked only by the `resetCursorRects` (page 149) method. If invoked in any other way, the resulting cursor rectangle will be discarded the next time the view's cursor rectangles are rebuilt.

Availability
Available in OS X v10.0 and later.

See Also
- `removeCursorRect:cursor:` (page 144)
- `discardCursorRects` (page 86)
- `resetCursorRects` (page 149)
- `visibleRect` (page 207)

Related Sample Code
DragItemAround

Declared in
`NSView.h`

addSubview:

Adds a view to the receiver's subviews so it's displayed above its siblings.

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

Parameters
`aView`

The view to add to the receiver as a subview.

Discussion
This method also sets the receiver as the next responder of `aView`.

The receiver retains `aView`. If you use `removeFromSuperview` (page 145) to remove `aView` from the view hierarchy, `aView` is released. If you want to keep using `aView` after removing it from the view hierarchy (if, for example, you are swapping through a number of views), you must retain it before invoking `removeFromSuperview` (page 145).

Availability
Available in OS X v10.0 and later.

See Also
- `addSubview:positioned:relativeTo:` (page 45)
- `subviews` (page 191)
- `removeFromSuperview` (page 145)

`setNextResponder:` (NSResponder)

- `viewWillMoveToSuperview:` (page 205)
- `viewWillMoveToWindow:` (page 205)

Related Sample Code
BlastApp
iSpend
MenuMadness
TextSizingExample
TreeView

Declared in
`NSView.h`

addSubview:positioned:relativeTo:

Inserts a view among the receiver's subviews so it's displayed immediately above or below another view.

```
- (void)addSubview:(NSView *)aView positioned:(NSWindowOrderingMode)place
relativeTo:(NSView *)otherView
```

Parameters

`aView`

The view object to add to the receiver as a subview.

`place`

An `enum` constant specifying the position of the `aView` relative to `otherView`. Valid values are `NSWindowAbove` or `NSWindowBelow`.

`otherView`

The other view `aView` is to be positioned relative to. If `otherView` is `nil` (or isn't a subview of the receiver), `aView` is added above or below all of its new siblings.

Discussion
This method also sets the receiver as the next responder of `aView`.

The receiver retains `aView`. If you use `removeFromSuperview` (page 145) to remove `aView` from the view hierarchy, `aView` is released. If you want to keep using `aView` after removing it from the view hierarchy (if, for example, you are swapping through a number of views), you must retain it before invoking `removeFromSuperview` (page 145).

Availability
Available in OS X v10.0 and later.

See Also
– `addSubview:` (page 44)
– `subviews` (page 191)
– `removeFromSuperview` (page 145)

`setNextResponder:` (`NSResponder`)

Related Sample Code
TreeView

Declared in
`NSView.h`

addToolTipRect:owner:userData:

Creates a tool tip for a defined area the receiver and returns a tag that identifies the tool tip rectangle.

```
– (NSToolTipTag)addToolTipRect:(NSRect)aRect owner:(id)anObject userData:(void
*)userData
```

Parameters

`aRect`

A rectangle defining the region of the receiver to associate the tool tip with.

`anObject`

An object from which to obtain the tool tip string. The object should either implement `view:stringForToolTip:point:userData:`, or return a suitable string from its `description` method. (It can therefore simply be an `NSString` object.)

> **Important:** The receiver maintains a weak reference to `anObject`. You are responsible for ensuring that `anObject` remains valid for as long as it may be needed.

`userData`

Any additional information you want to pass to `view:stringForToolTip:point:userData:`; it is not used if `anObject` does not implement this method.

Return Value

An integer tag identifying the tool tip; you can use this tag to remove the tool tip.

Discussion

The tool tip string is obtained dynamically from `anObject` by invoking either the `NSToolTipOwner` informal protocol method `view:stringForToolTip:point:userData:`, if implemented, or the `NSObject` protocol method `description`.

Availability

Available in OS X v10.0 and later.

See Also

– `removeToolTip:` (page 146)
– `removeAllToolTips` (page 143)

Declared in

`NSView.h`

addTrackingArea:

Adds a given tracking area to the receiver.

```
- (void)addTrackingArea:(NSTrackingArea *)trackingArea
```

Parameters

`trackingArea`

The tracking area to add to the receiver.

Availability

Available in OS X v10.5 and later.

Related Sample Code

BasicCocoaAnimations
CustomMenus
MenuItemView
TrackIt

Declared in

`NSView.h`

addTrackingRect:owner:userData:assumeInside:

Establishes an area for tracking mouse-entered and mouse-exited events within the receiver and returns a tag that identifies the tracking rectangle.

```
- (NSTrackingRectTag)addTrackingRect:(NSRect)aRect owner:(id)userObject userData:(void
 *)userData assumeInside:(BOOL)flag
```

Parameters

`aRect`

A rectangle that defines a region of the receiver for tracking mouse-entered and mouse-exited events.

`userObject`

The object that gets sent the event messages. It can be the receiver itself or some other object (such as an NSCursor or a custom drawing tool object), as long as it responds to both `mouseEntered:` and `mouseExited:`.

`userData`

Data stored in the `NSEvent` object for each tracking event.

`flag`

If `YES`, the first event will be generated when the cursor leaves `aRect`, regardless if the cursor is inside `aRect` when the tracking rectangle is added. If `NO` the first event will be generated when the cursor leaves `aRect` if the cursor is initially inside `aRect`, or when the cursor enters `aRect` if the cursor is initially outside `aRect`. You usually want to set this flag to `NO`.

Return Value

A tag that identifies the tracking rectangle. It is stored in the associated `NSEvent` objects and can be used to remove the tracking rectangle.

Discussion

Tracking rectangles provide a general mechanism that can be used to trigger actions based on the cursor location (for example, a status bar or hint field that provides information on the item the cursor lies over). To simply change the cursor over a particular area, use `addCursorRect:cursor:` (page 43). If you must use tracking rectangles to change the cursor, the `NSCursor` class specification describes the additional methods that must be invoked to change cursors by using tracking rectangles.

On OS X v10.5 and later, tracking areas provide a greater range of functionality (see `addTrackingArea:` (page 47)).

Availability

Available in OS X v10.0 and later.

See Also

– `removeTrackingRect:` (page 147)
– `addTrackingArea:` (page 47)
`userData` (`NSEvent`)

Declared in

`NSView.h`

adjustPageHeightNew:top:bottom:limit:

Overridden by subclasses to adjust page height during automatic pagination.

```
- (void)adjustPageHeightNew:(CGFloat *)newBottom top:(CGFloat)top
bottom:(CGFloat)proposedBottom limit:(CGFloat)bottomLimit
```

Parameters

`newBottom`

Returns by indirection a new `float` value for the bottom edge of the pending page rectangle in the receiver's coordinate system.

`top`

A `float` value that sets the top edge of the pending page rectangle in the receiver's coordinate system.

`proposedBottom`

A `float` value that sets the bottom edge of the pending page rectangle in the receiver's coordinate system.

`bottomLimit`

The topmost float value `newBottom` can be set to, as calculated using the return value of `heightAdjustLimit` (page 109).

Discussion

This method is invoked by `print:` (page 138). The receiver can raise the bottom edge and return the new value in `newBottom`, allowing it to prevent items such as lines of text from being divided across pages. If `bottomLimit` is exceeded, the pagination mechanism simply uses `bottomLimit` for the bottom edge.

The default implementation of this method propagates the message to its subviews, allowing nested views to adjust page height for their drawing as well. An `NSButton` object or other small view, for example, will nudge the bottom edge up if necessary to prevent itself from being cut in two (thereby pushing it onto an adjacent page). Subclasses should invoke `super`'s implementation, if desired, after first making their own adjustments.

Availability

Available in OS X v10.0 and later.

See Also

– `adjustPageWidthNew:left:right:limit:` (page 49)

Declared in

`NSView.h`

adjustPageWidthNew:left:right:limit:

Overridden by subclasses to adjust page width during automatic pagination.

```
- (void)adjustPageWidthNew:(CGFloat *)newRight left:(CGFloat)left
right:(CGFloat)proposedRight limit:(CGFloat)rightLimit
```

Parameters

`newRight`

Returns by indirection a new `float` value for the right edge of the pending page rectangle in the receiver's coordinate system.

`left`

A `float` value that sets the left edge of the pending page rectangle in the receiver's coordinate system.

`proposedRight`

A `float` value that sets the right edge of the pending page rectangle in the receiver's coordinate system.

`rightLimit`

The leftmost `float` value `newRight` can be set to, as calculated using the return value of `widthAdjustLimit` (page 211).

Discussion

This method is invoked by `print:` (page 138). The receiver can pull in the right edge and return the new value in `newRight`, allowing it to prevent items such as small images or text columns from being divided across pages. If `rightLimit` is exceeded, the pagination mechanism simply uses `rightLimit` for the right edge.

The default implementation of this method propagates the message to its subviews, allowing nested views to adjust page width for their drawing as well. An `NSButton` object or other small view, for example, will nudge the right edge out if necessary to prevent itself from being cut in two (thereby pushing it onto an adjacent page). Subclasses should invoke `super`'s implementation, if desired, after first making their own adjustments.

Availability

Available in OS X v10.0 and later.

See Also

– `adjustPageHeightNew:top:bottom:limit:` (page 48)

Declared in

`NSView.h`

adjustScroll:

Overridden by subclasses to modify a given rectangle, returning the altered rectangle.

```
- (NSRect)adjustScroll:(NSRect)proposedVisibleRect
```

Parameters

`proposedVisibleRect`

A rectangle defining a region of the receiver.

Discussion

`NSClipView` invokes this method to allow its document view to adjust its position during scrolling. For example, a custom view object that displays a table of data can adjust the origin of `proposedVisibleRect` so rows or columns aren't cut off by the edge of the enclosing `NSClipView`. `NSView`'s implementation simply returns `proposedVisibleRect`.

`NSClipView` only invokes this method during automatic or user controlled scrolling. Its `scrollToPoint:` method doesn't invoke this method, so you can still force a scroll to an arbitrary point.

Availability

Available in OS X v10.0 and later.

Declared in

`NSView.h`

alignmentRectForFrame:

Returns the view's alignment rectangle for a given frame.

```
- (NSRect)alignmentRectForFrame:(NSRect)frame
```

Parameters

`frame`

The frame whose corresponding alignment rectangle is desired.

Return Value

The alignment rectangle for the specified frame.

Discussion

The constraint-based layout system uses alignment rectangles to align views, rather than their frame. This allows custom views to be aligned based on the location of their content while still having a frame that encompasses any ornamentation they need to draw around their content, such as shadows or reflections.

The default implementation returns the view's frame modified by the view's `alignmentRectInsets` (page 52). Most custom views can override `alignmentRectInsets` to specify the location of their content within their frame. Custom views that require arbitrary transformations can override `alignmentRectForFrame:` and `frameForAlignmentRect:` (page 104) to describe the location of their content. These two methods must always be inverses of each other.

Availability
Available in OS X v10.7 and later.

Declared in
`NSLayoutConstraint.h`

alignmentRectInsets

Returns the insets from the view's frame that define its alignment rectangle.

```
– (NSEdgeInsets)alignmentRectInsets
```

Return Value
The insets from the view's frame that define its alignment rectangle.

Discussion
The default implementation of this method returns an `NSEdgeInsets` structure with zero values. Custom views that draw ornamentation around their content should override this method to return insets that align with the edges of the content, excluding the ornamentation. This allows the constraint-based layout system to align views based on their content, rather than just their frame.

Custom views whose content location can't be expressed by a simple set of insets should override `alignmentRectForFrame:` (page 51) and `frameForAlignmentRect:` (page 104) to describe their custom transform between alignment rectangle and frame.

Availability
Available in OS X v10.7 and later.

Declared in
`NSLayoutConstraint.h`

allocateGState

Causes the receiver to maintain a private graphics state object, which encapsulates all parameters of the graphics environment.

```
– (void)allocateGState
```

Discussion
If you do not invoke `allocateGState`, a graphics state object is constructed from scratch each time the NSView is focused.

The receiver builds the graphics state parameters using `setUpGState` (page 183), then automatically establishes this graphics state each time the focus is locked on it. A graphics state may improve performance for view objects that are focused often and need to set many parameters, but use of standard rendering operators is normally efficient enough.

Because graphics states occupy a fair amount of memory, they can actually degrade performance. Be sure to test application performance with and without the private graphics state before committing to its use.

Availability
Available in OS X v10.0 and later.

See Also
- `setUpGState` (page 183)
- `gState` (page 107)
- `renewGState` (page 148)
- `releaseGState` (page 143)
- `lockFocus` (page 122)

Declared in
`NSView.h`

alphaValue

Returns the opacity of the receiver

```
- (CGFloat)alphaValue
```

Return Value
The current opacity of the receiver

Discussion
This method returns the value of the `opacity` property of the receiver's layer. Possible values are between 0.0 (transparent) and 1.0 (opaque). The default is 1.0.

Sending this message to a view that is not managing a Core Animation layer causes an exception.

Availability
Available in OS X v10.5 and later.

Related Sample Code
RoundTransparentWindow

Declared in
`NSView.h`

ancestorSharedWithView:

Returns the closest ancestor shared by the receiver and a given view.

```
- (NSView *)ancestorSharedWithView:(NSView *)aView
```

Parameters

`aView`

The view to test (along with the receiver) for closest shared ancestor.

Return Value

The closest ancestor or `nil` if there's no such object. Returns `self` if `aView` is identical to the receiver.

Availability

Available in OS X v10.0 and later.

See Also

– `isDescendantOf:` (page 113)

Declared in

`NSView.h`

autoresizesSubviews

Returns `YES` if the receiver automatically resizes its subviews using `resizeSubviewsWithOldSize:` (page 150) whenever its frame size changes, `NO` otherwise.

```
- (BOOL)autoresizesSubviews
```

Availability

Available in OS X v10.0 and later.

See Also

– `setAutoresizesSubviews:` (page 157)

Declared in

`NSView.h`

autoresizingMask

Returns the receiver's autoresizing mask, which determines how it's resized by the `resizeWithOldSuperviewSize:` (page 150) method.

```
- (NSUInteger)autoresizingMask
```

Return Value
An integer bit mask specified by combining using the C bitwise OR operator any of the options described in "Resizing masks" (page 215).

Discussion
If the autoresizing mask is equal to `NSViewNotSizable` (that is, if none of the options are set), then the receiver doesn't resize at all in `resizeWithOldSuperviewSize:` (page 150).

Availability
Available in OS X v10.0 and later.

Declared in
`NSView.h`

autoscroll:

Scrolls the receiver's closest ancestor `NSClipView` object proportionally to the distance of an event that occurs outside of it.

```
- (BOOL)autoscroll:(NSEvent *)theEvent
```

Parameters
`theEvent`
An event object whose location should be expressed in the window's base coordinate system (which it normally is), not the receiving view's.

Return Value
Returns `YES` if any scrolling is performed; otherwise returns `NO`.

Discussion
View objects that track mouse-dragged events can use this method to scroll automatically when the cursor is dragged outside of the `NSClipView` object. Repeated invocations of this method (with an appropriate delay) result in continual scrolling, even when the mouse doesn't move.

Availability
Available in OS X v10.0 and later.

See Also
`autoscroll:` (NSClipView)
`- scrollPoint:` (page 154)
`- isDescendantOf:` (page 113)

Related Sample Code
DragItemAround

Rulers

Declared in
`NSView.h`

backgroundFilters

Returns the array of CoreImage filters that are applied to the receiver's background

```
- (NSArray *)backgroundFilters
```

Return Value
An array of CoreImage filters.

Discussion
This method returns the value of the `backgroundFilters` property of the receiver's layer.

Sending this message to a view that is not managing a Core Animation layer causes an exception.

Availability
Available in OS X v10.5 and later.

Declared in
`NSView.h`

backingAlignedRect:options:

Returns a backing store pixel aligned rectangle in window coordinates.

```
- (NSRect)backingAlignedRect:(NSRect)aRect options:(NSAlignmentOptions)options
```

Parameters
`aRect`
The rectangle in view coordinates.

`options`
The alignment options. See `NSAlignmentOptions` for possible values.

Return Value
A rectangle that is aligned to the backing store pixels using the specified options. The rectangle is in window coordinates.

Discussion
Uses the `NSIntegralRectWithOptions` function to produce a backing store pixel aligned rectangle from the given input rectangle in window coordinates.

Availability
Available in OS X v10.7 and later.

Declared in
`NSView.h`

baselineOffsetFromBottom

Returns the distance between the bottom of the view's alignment rectangle and the view's baseline.

```
- (CGFloat)baselineOffsetFromBottom
```

Return Value
The distance between the bottom of the view's alignment rectangle and the view's baseline.

Discussion
The default implementation of this method returns 0. Custom views with content that has the concept of a baseline, such as text, should override this method to return the correct distance between the bottom of their alignment rectangle and the baseline.

Availability
Available in OS X v10.7 and later.

Declared in
`NSLayoutConstraint.h`

beginDocument

Invoked at the beginning of the printing session, this method sets up the current graphics context.

```
- (void)beginDocument
```

Discussion
Note that this method may be invoked in a subthread.

Override it to configure printing related settings. You should store your settings in the object returned by `NSPrintInfo`'s `sharedPrintInfo` class method, which is guaranteed to return an instance specific to the thread in which you invoke this method. If you override this method, call the superclass implementation.

Availability
Available in OS X v10.0 and later.

Declared in
`NSView.h`

beginDraggingSessionWithItems:event:source:

Initiates a dragging session with a group of dragging items.

```
- (NSDraggingSession *)beginDraggingSessionWithItems:(NSArray *)items event:(NSEvent
 *)event source:(id < NSDraggingSource >)source
```

Parameters
`items`

The dragging items. The frame property of each `NSDraggingItem` must be in the view's coordinate system.

`event`

The mouse-down event object from which to initiate the drag operation. In particular, its mouse location is used for the offset of the icon being dragged.

`source`

An object that serves as the controller of the dragging operation. It must conform to the `NSDraggingSource` protocol and is typically the receiver itself or its `NSWindow` object.

Return Value
The dragging session for the drag.

Discussion
A basic drag starts by calling `beginDraggingSessionWithItems:event:source:`.

The caller can take the returned NSDraggingSession and continue to modify its properties such as -slidesBackOnCancelOrFail. When the drag actually starts, the source is sent a -draggingSession:willBeginAtPoint: message followed by multiple -draggingSession:movedToPoint: messages as the user drags.

Once the drag is ended or cancelled, the source receives a `draggingSession:endedAtPoint:operation:` method and the drag is complete.

Availability
Available in OS X v10.7 and later.

See Also
– `registerForDraggedTypes:` (page 142)

- `unregisterDraggedTypes` (page 197)
- `dragImage:at:offset:event:pasteboard:source:slideBack:` (page 91)
- `shouldDelayWindowOrderingForEvent:` (page 187)

Related Sample Code
CocoaDragAndDrop

MultiPhotoFrame

Declared in
`NSView.h`

beginPageInRect:atPlacement:

Called at the beginning of each page, this method sets up the coordinate system so that a region inside the receiver's bounds is translated to a specified location..

```
- (void)beginPageInRect:(NSRect)aRect atPlacement:(NSPoint)location
```

Parameters
`aRect`
A rectangle defining the region to be translated.

`location`
A point that is the end-point of translation.

Discussion
If you override this method, be sure to call the superclass implementation.

Availability
Available in OS X v10.0 and later.

Declared in
`NSView.h`

bitmapImageRepForCachingDisplayInRect:

Returns a bitmap-representation object suitable for caching the specified portion of the receiver.

```
- (NSBitmapImageRep *)bitmapImageRepForCachingDisplayInRect:(NSRect)aRect
```

Parameters
`aRect`
A rectangle defining the area of the receiver to be cached.

Return Value
An autoreleased `NSBitmapImageRep` object or `nil` if the object could not be created.

Discussion
Passing the visible rectangle of the receiver (`[self visibleRect]`) returns a bitmap suitable for caching the current contents of the view, including all of its descendants.

Availability
Available in OS X v10.4 and later.

See Also
`– cacheDisplayInRect:toBitmapImageRep:` (page 62)

Related Sample Code
AnimatedTableView

MultiPhotoFrame

Declared in
`NSView.h`

bounds

Returns the receiver's bounds rectangle, which expresses its location and size in its own coordinate system.

```
– (NSRect)bounds
```

Discussion
By default, the origin of the returned rectangle is (0, 0) and its size matches the size of the receiver's frame rectangle (measured in points). In OS X v10.5 and later, if the receiver is being rendered into an OpenGL graphics context (using an `NSOpenGLContext` object), the default bounds origin is still (0, 0) but the default bounds size is measured in pixels instead of points. Thus, for user space scale factors other than 1.0, the default size of the bounds rectangle may be bigger or smaller than the default size of the frame rectangle when drawing with OpenGL.

Important: Developers of OpenGL applications should not rely on this method converting coordinates to pixels automatically in future releases. Instead, you should convert coordinates to device space explicitly using the `convertPointToBase:` (page 73), `convertSizeToBase:` (page 83), or `convertRectToBase:` (page 78) methods or their earlier counterparts `convertPoint:toView:` (page 69), `convertSize:toView:` (page 80), or `convertRect:toView:` (page 75).

If you explicitly change the origin or size of the bounds rectangle, this method does not return the default rectangle and instead returns the rectangle you set. If you add a rotation factor to the view, however, that factor is also reflected in the returned bounds rectangle. You can determine if a rotation factor is in effect by calling the `boundsRotation` (page 61) method.

Availability
Available in OS X v10.0 and later.

See Also
– `frame` (page 103)
– `setBounds:` (page 159)

Related Sample Code
From A View to A Movie
From A View to A Picture
GLSL Basics Cocoa
MultiPhotoFrame
TargetGallery

Declared in
`NSView.h`

boundsRotation

Returns the angle, in degrees, of the receiver's bounds rectangle relative to its frame rectangle.

```
– (CGFloat)boundsRotation
```

Discussion
See the `setBoundsRotation:` (page 161) method description for more information on bounds rotation.

Availability
Available in OS X v10.0 and later.

See Also
– `rotateByAngle:` (page 151)

– setBoundsRotation: (page 161)

Declared in
NSView.h

cacheDisplayInRect:toBitmapImageRep:

Draws the specified area of the receiver, and its descendants, into a provided bitmap-representation object.

```
– (void)cacheDisplayInRect:(NSRect)rect toBitmapImageRep:(NSBitmapImageRep
*)bitmapImageRep
```

Parameters

`rect`

A rectangle defining the region to be drawn into `bimapImageRep`.

`bitmapImageRep`

An `NSBitmapImageRep` object. For pixel-format compatibility, `bitmapImageRep` should have been obtained from `bitmapImageRepForCachingDisplayInRect:` (page 59).

Discussion

You are responsible for initializing the bitmap to the desired configuration before calling this method. However, once initialized, you can reuse the same bitmap multiple times to refresh the cached copy of your view's contents.

The bitmap produced by this method is transparent (that is, has an alpha value of 0) wherever the receiver and its descendants do not draw any content.

Availability

Available in OS X v10.4 and later.

See Also

– bitmapImageRepForCachingDisplayInRect: (page 59)

Related Sample Code
AnimatedTableView
MultiPhotoFrame

Declared in
NSView.h

canBecomeKeyView

Returns whether the receiver can become key view.

```
- (BOOL)canBecomeKeyView
```

Return Value
Returns `YES` if the receiver can become key view, `NO` otherwise.

Availability
Available in OS X v10.3 and later.

Declared in
`NSView.h`

canDraw

Returns `YES` if drawing commands will produce any result, `NO` otherwise.

```
- (BOOL)canDraw
```

Discussion
Use this method when invoking a draw method directly along with `lockFocus` (page 122) and `unlockFocus` (page 196), bypassing the `display...` methods (which test drawing ability and perform locking for you). If this method returns `NO`, you shouldn't invoke `lockFocus` (page 122) or perform any drawing.

A view object can draw on-screen if it is not hidden, it is attached to a view hierarchy in a window (`NSWindow`), and the window has a corresponding window device. A view object can draw during printing if it is a descendant of the view being printed.

Availability
Available in OS X v10.0 and later.

See Also
– `setHidden:` (page 171)

Declared in
`NSView.h`

canDrawConcurrently

Returns whether the view's `drawRect:` method can be invoked on a background thread.

```
- (BOOL)canDrawConcurrently
```

Return Value
YES if drawRect: (page 96) can be invoked from a background thread, otherwise NO. The default is NO.

Availability
Available in OS X v10.6 and later.

See Also
– setCanDrawConcurrently: (page 163)

Declared in
NSView.h

canDrawSubviewsIntoLayer

Returns a Boolean value indicating whether the view incorporates content from its subviews into its own layer

```
- (BOOL)canDrawSubviewsIntoLayer
```

Return Value
YES if the view incorporates subview content into its layer or NO if each subview continues to use its own layer object.

Discussion
When this method returns YES, any subviews that have an implicitly created layer—that is, layers for which you did not explicitly call the setWantsLayer: (page 184) method with the value YES—draw their contents into the current view's layer. In other words, the subviews do not get a layer of their own and instead draw their content into the parent view's layer. All views involved in the operation draw their content using their drawRect: (page 96) method.

Availability
Available in OS X v10.9 and later.

Declared in
NSView.h

centerScanRect:

Converts the corners of a specified rectangle to lie on the center of device pixels, which is useful in compensating for rendering overscanning when the coordinate system has been scaled.

```
- (NSRect)centerScanRect:(NSRect)aRect
```

Parameters

`aRect`

The rectangle whose corners are to be converted.

Return Value

The adjusted rectangle.

Discussion

This method converts the given rectangle to device coordinates, adjusts the rectangle to lie in the center of the pixels, and converts the resulting rectangle back to the receiver's coordinate system. Note that this method does not take into account any transformations performed using the `NSAffineTransform` class or Quartz 2D routines.

Availability

Available in OS X v10.0 and later.

See Also

– `isRotatedOrScaledFromBase` (page 117)

Related Sample Code

QuickLookSketch

Sketch

Sketch+Accessibility

Declared in

`NSView.h`

compositingFilter

Returns the CoreImage filter that is used to composite the receiver's contents with the background

```
- (CIFilter *)compositingFilter
```

Return Value

The CoreImage filter.

Discussion

This method returns the value of the `filters` property of the receiver's layer.

Sending this message to a view that is not managing a Core Animation layer causes an exception.

Availability
Available in OS X v10.5 and later.

Declared in
`NSView.h`

constraints

Returns the constraints held by the view.

```
- (NSArray *)constraints
```

Return Value
The constraints held by the view.

Availability
Available in OS X v10.7 and later.

Declared in
`NSLayoutConstraint.h`

constraintsAffectingLayoutForOrientation:

Returns the constraints impacting the layout of the view for a given orientation.

```
- (NSArray
*)constraintsAffectingLayoutForOrientation:(NSLayoutConstraintOrientation)orientation
```

Parameters
`orientation`
The direction of the dimension for which the constraints should be found.

Return Value
The constraints impacting the layout of the view for the specified orientation.

Discussion
The returned set of constraints may not all include the view explicitly. Constraints that impact the location of the view implicitly may also be included. While this provides a good starting point for debugging, there is no guarantee that the returned set of constraints will include all of the constraints that have an impact on the view's layout in the given orientation.

This method should only be used for debugging constraint-based layout. No application should ship with calls to this method as part of its operation.

Availability
Available in OS X v10.7 and later.

Declared in
`NSLayoutConstraint.h`

contentCompressionResistancePriorityForOrientation:

Returns the priority with which a view resists being made smaller than its intrinsic size.

```
- (NSLayoutPriority)contentCompressionResistancePriorityForOrientation:(NSLayoutConstraintOrientation)orientation
```

Parameters
`orientation`
The orientation of the dimension of the view that might be reduced.

Return Value
The priority with which the view should resist being compressed from its intrinsic size in the specified orientation.

Discussion
The constraint-based layout system uses these priorities when determining the best layout for views that are encountering constraints that would require them to be smaller than their intrinsic size.

Subclasses should not override this method. Instead, custom views should set default values for their content on creation, typically to `NSLayoutPriorityDefaultLow` or `NSLayoutPriorityDefaultHigh`.

Availability
Available in OS X v10.7 and later.

See Also
`- setContentCompressionResistancePriority:forOrientation:` (page 165)

Declared in
`NSLayoutConstraint.h`

contentFilters

Returns the array of CoreImage filters that are applied to the contents of the receiver and its sublayers.

```
- (NSArray *)contentFilters
```

Return Value
An array of CoreImage filters

Discussion
This method returns the value of the `filters` property of the receiver's layer.

Sending this message to a view that is not managing a Core Animation layer causes an exception.

Availability
Available in OS X v10.5 and later.

Declared in
`NSView.h`

contentHuggingPriorityForOrientation:

Returns the priority with which a view resists being made larger than its intrinsic size.

```
- (NSLayoutPriority)contentHuggingPriorityForOrientation:(NSLayoutConstraintOrientation)orientation
```

Parameters
`orientation`
The orientation of the dimension of the view that might be enlarged.

Return Value
The priority with which the view should resist being enlarged from its intrinsic size in the specified orientation.

Discussion
The constraint-based layout system uses these priorities when determining the best layout for views that are encountering constraints that would require them to be larger than their intrinsic size.

Subclasses should not override this method. Instead, custom views should set default values for their content on creation, typically to `NSLayoutPriorityDefaultLow` or `NSLayoutPriorityDefaultHigh`.

Availability
Available in OS X v10.7 and later.

See Also
`- setContentHuggingPriority:forOrientation:` (page 166)

Declared in
NSLayoutConstraint.h

convertPoint:fromView:

Converts a point from the coordinate system of a given view to that of the receiver.

```
- (NSPoint)convertPoint:(NSPoint)aPoint fromView:(NSView *)aView
```

Parameters
aPoint

A point specifying a location in the coordinate system of aView.

aView

The view with aPoint in its coordinate system. Both aView and the receiver must belong to the same NSWindow object, and that window must not be nil. If aView is nil, this method converts from window coordinates instead.

Return Value
The point converted to the coordinate system of the receiver.

Availability
Available in OS X v10.0 and later.

See Also
– convertRect:fromView: (page 74)
– convertSize:fromView: (page 79)
– ancestorSharedWithView: (page 54)
contentView (NSWindow)

Related Sample Code
AnimatedTableView
CIAnnotation
CircleView
OverlayView
TreeView

Declared in
NSView.h

convertPoint:toView:

Converts a point from the receiver's coordinate system to that of a given view.

```
- (NSPoint)convertPoint:(NSPoint)aPoint toView:(NSView *)aView
```

Parameters

`aPoint`

A point specifying a location in the coordinate system of the receiver.

`aView`

The view into whose coordinate system `aPoint` is to be converted. Both `aView` and the receiver must belong to the same `NSWindow` object, and that window must not be `nil`. If `aView` is `nil`, this method converts to window coordinates instead.

Return Value

The point converted to the coordinate system of `aView`.

Availability

Available in OS X v10.0 and later.

See Also

`– convertRect:toView:` (page 75)
`– convertSize:toView:` (page 80)
`– ancestorSharedWithView:` (page 54)
`contentView` (`NSWindow`)

Related Sample Code

CustomMenus
TreeView

Declared in

`NSView.h`

convertPointFromBacking:

Converts a point from its pixel aligned backing store coordinate system to the view's interior coordinate system.

```
- (NSPoint)convertPointFromBacking:(NSPoint)aPoint
```

Parameters

`aPoint`

The point in the pixel backing store aligned coordinate system.

Return Value

A point in the view's interior coordinate system.

Availability
Available in OS X v10.7 and later.

See Also
– `convertPointToBacking:` (page 72)

Declared in
`NSView.h`

convertPointFromBase:

Converts the point from the base coordinate system to the receiver's coordinate system. (Deprecated. Use `convertPointFromBacking:` (page 70) instead.)

```
– (NSPoint)convertPointFromBase:(NSPoint)aPoint
```

Parameters
`aPoint`

A point specifying a location in the base coordinate system.

Return Value
The point converted to the receiver's base coordinate system.

Discussion
See "Transforming View Coordinates To and From Base Space" in *View Programming Guide* for a description of view coordinate to base coordinate conversion.

Availability
Available in OS X v10.5 and later.

Related Sample Code
Sketch+Accessibility

Declared in
`NSView.h`

convertPointFromLayer:

Convert the point from the layer's interior coordinate system to the view's interior coordinate system.

```
– (NSPoint)convertPointFromLayer:(NSPoint)aPoint
```

Parameters

`aPoint`

The point in the layer's interior coordinate system.

Return Value

The point in the view's interior coordinate system.

Discussion

The layer's space is virtual, and doesn't take into account the layer's `contentsScale` setting.

Availability

Available in OS X v10.7 and later.

See Also

– `convertPointToLayer:` (page 73)

Declared in

`NSView.h`

convertPointToBacking:

Converts a point from the view's interior coordinate system to its pixel aligned backing store coordinate system.

```
- (NSPoint)convertPointToBacking:(NSPoint)aPoint
```

Parameters

`aPoint`

The point in the view's interior coordinate system.

Return Value

A point in its pixel aligned backing store coordinate system.

Availability

Available in OS X v10.7 and later.

See Also

– `convertPointFromBacking:` (page 70)

Declared in

`NSView.h`

convertPointToBase:

Converts the point from the receiver's coordinate system to the base coordinate system. (Deprecated. Use `convertPointToBacking:` *(page 72) instead.)*

```
- (NSPoint)convertPointToBase:(NSPoint)aPoint
```

Parameters

`aPoint`

A point specifying a location in the coordinate system of the receiver.

Return Value

The point converted to the base coordinate system.

Discussion

See "Transforming View Coordinates To and From Base Space" in *View Programming Guide* for a description of view coordinate to base coordinate conversion.

Availability

Available in OS X v10.5 and later.

Related Sample Code

Sketch+Accessibility

ZipBrowser

Declared in

`NSView.h`

convertPointToLayer:

Convert the size from the view's interior coordinate system to the layer's interior coordinate system.

```
- (NSPoint)convertPointToLayer:(NSPoint)aPoint
```

Parameters

`aPoint`

A point in the view's interior coordinate system.

Return Value

A point in the view's layer interior coordinate system.

Discussion

The layer's space is virtual, and doesn't take into account the layer's `contentsScale` setting.

Availability
Available in OS X v10.7 and later.

See Also
– `convertPointFromLayer:` (page 71)

Declared in
`NSView.h`

convertRect:fromView:

Converts a rectangle from the coordinate system of another view to that of the receiver.

```
- (NSRect)convertRect:(NSRect)aRect fromView:(NSView *)aView
```

Parameters
`aRect`

The rectangle in `aView`'s coordinate system.

`aView`

The view with `aRect` in its coordinate system. Both `aView` and the receiver must belong to the same `NSWindow` object, and that window must not be `nil`. If `aView` is `nil`, this method converts from window coordinates instead.

Return Value
The converted rectangle.

Availability
Available in OS X v10.0 and later.

See Also
– `convertPoint:fromView:` (page 69)
– `convertSize:fromView:` (page 79)
– `ancestorSharedWithView:` (page 54)
`contentView` (`NSWindow`)

Related Sample Code
TreeView

Declared in
`NSView.h`

convertRect:toView:

Converts a rectangle from the receiver's coordinate system to that of another view.

```
- (NSRect)convertRect:(NSRect)aRect toView:(NSView *)aView
```

Parameters

`aRect`

A rectangle in the receiver's coordinate system.

`aView`

The view that is the target of the conversion operation. Both `aView` and the receiver must belong to the same `NSWindow` object, and that window must not be `nil`. If `aView` is `nil`, this method converts the rectangle to window coordinates instead.

Return Value

The converted rectangle.

Availability

Available in OS X v10.0 and later.

See Also

– `convertPoint:toView:` (page 69)
– `convertSize:toView:` (page 80)
– `ancestorSharedWithView:` (page 54)
`contentView` (`NSWindow`)

Related Sample Code

Rulers
SharingServices
TableViewPlayground

Declared in

`NSView.h`

convertRectFromBacking:

Converts a rectangle from its pixel aligned backing store coordinate system to the view's interior coordinate system.

```
- (NSRect)convertRectFromBacking:(NSRect)aRect
```

Parameters

`aRect`

The rectangle in the pixel backing store coordinate system.

Return Value

A rectangle in the view's interior coordinate system.

Availability

Available in OS X v10.7 and later.

See Also

– `convertRectToBacking:` (page 77)

Declared in

`NSView.h`

convertRectFromBase:

Converts the rectangle from the base coordinate system to the receiver's coordinate system. (Deprecated. Use `convertRectFromBacking:` (page 75) instead.)

```
– (NSRect)convertRectFromBase:(NSRect)aRect
```

Parameters

`aRect`

A rectangle in the base coordinate system

Return Value

A rectangle in the receiver's coordinate system

Discussion

See "Transforming View Coordinates To and From Base Space" in *View Programming Guide* for a description of view coordinate to base coordinate conversion.

Availability

Available in OS X v10.5 and later.

Related Sample Code

Cocoa Tips and Tricks

Declared in

`NSView.h`

convertRectFromLayer:

Convert the rectangle from the layer's interior coordinate system to the view's interior coordinate system.

```
- (NSRect)convertRectFromLayer:(NSRect)aRect
```

Parameters

`aRect`

A rectangle in the layer's interior coordinate system.

Return Value

A rectangle in the view's interior coordinate system.

Discussion

The layer's space is virtual, and doesn't take into account the layer's `contentsScale` setting.

Availability

Available in OS X v10.7 and later.

See Also

– `convertRectToLayer:` (page 78)

Declared in

`NSView.h`

convertRectToBacking:

Converts a rectangle from the view's interior coordinate system to its pixel aligned backing store coordinate system.

```
- (NSRect)convertRectToBacking:(NSRect)aRect
```

Parameters

`aRect`

A rectangle in the view's interior coordinate system.

Return Value

A rectangle in its pixel aligned backing store coordinate system.

Availability

Available in OS X v10.7 and later.

See Also

– `convertRectFromBacking:` (page 75)

Declared in
`NSView.h`

convertRectToBase:

Converts the rectangle from the receiver's coordinate system to the base coordinate system. (Deprecated. Use `convertRectToBacking:` (page 77) instead.)

```
– (NSRect)convertRectToBase:(NSRect)aRect
```

Parameters
`aRect`

A rectangle in the receiver's coordinate system

Return Value
A rectangle in the base coordinate system

Discussion
See "Transforming View Coordinates To and From Base Space" in *View Programming Guide* for a description of view coordinate to base coordinate conversion.

Availability
Available in OS X v10.5 and later.

Related Sample Code
AnimatedTableView
Cocoa Tips and Tricks

Declared in
`NSView.h`

convertRectToLayer:

Convert the size from the view's interior coordinate system to the layer's interior coordinate system.

```
– (NSRect)convertRectToLayer:(NSRect)aRect
```

Parameters
`aRect`

A rectangle in the view's interior coordinate system.

Return Value
A rectangle in the layer's interior coordinate system.

Discussion
The layer's space is virtual, and doesn't take into account the layer's `contentsScale` setting.

Availability
Available in OS X v10.7 and later.

See Also
`– convertRectFromLayer:` (page 77)

Declared in
`NSView.h`

convertSize:fromView:

Converts a size from another view's coordinate system to that of the receiver.

```
– (NSSize)convertSize:(NSSize)aSize fromView:(NSView *)aView
```

Parameters
`aSize`

The size (width and height) in `aView`'s coordinate system.

`aView`

The view with `aSize` in its coordinate system. Both `aView` and the receiver must belong to the same `NSWindow` object, and that window must not be `nil`. If `aView` is `nil`, this method converts from window coordinates instead.

Return Value
The converted size, as an `NSSize` structure.

Discussion
The returned `NSSize` values are always forced to have positive a width and height.

Availability
Available in OS X v10.0 and later.

See Also
`– convertPoint:fromView:` (page 69)
`– convertRect:fromView:` (page 74)
`– ancestorSharedWithView:` (page 54)
`contentView` (`NSWindow`)

Related Sample Code
TreeView

Declared in
`NSView.h`

convertSize:toView:

Converts a size from the receiver's coordinate system to that of another view.

```
- (NSSize)convertSize:(NSSize)aSize toView:(NSView *)aView
```

Parameters

`aSize`

The size (width and height) in the receiver's coordinate system.

`aView`

The view that is the target of the conversion operation. Both `aView` and the receiver must belong to the same `NSWindow` object, and that window must not be `nil`. If `aView` is `nil`, this method converts to window coordinates instead.

Return Value
The converted size, as an `NSSize` structure.

Discussion
The returned `NSSize` values are always forced to have positive a width and height.

Availability
Available in OS X v10.0 and later.

See Also
- `convertPoint:toView:` (page 69)
- `convertRect:toView:` (page 75)
- `ancestorSharedWithView:` (page 54)

`contentView` (NSWindow)

Related Sample Code
TreeView

Declared in
`NSView.h`

convertSizeFromBacking:

Converts a size from its pixel aligned backing store coordinate system to the view's interior coordinate system.

```
- (NSSize)convertSizeFromBacking:(NSSize)aSize
```

Parameters

`aSize`

The size in the pixel aligned coordinate coordinate system.

Return Value

The size in the view's interior coordinate system.

Availability

Available in OS X v10.7 and later.

See Also

– `convertSizeFromBacking:` (page 81)

Declared in

`NSView.h`

convertSizeFromBase:

Converts the size from the base coordinate system to the receiver's coordinate system. (Deprecated. Use `convertSizeFromBacking:` (page 81) instead.)

```
- (NSSize)convertSizeFromBase:(NSSize)aSize
```

Parameters

`aSize`

A size in the base coordinate system

Return Value

The size converted to the receiver's coordinate system.

Discussion

See "Transforming View Coordinates To and From Base Space" in *View Programming Guide* for a description of view coordinate to base coordinate conversion.

Availability

Available in OS X v10.5 and later.

Related Sample Code

Sketch+Accessibility

Declared in
`NSView.h`

convertSizeFromLayer:

Convert the size from the layer's interior coordinate system to the view's interior coordinate system.

```
- (NSSize)convertSizeFromLayer:(NSSize)aSize
```

Parameters
`aSize`
A size in the layer's interior coordinate system.

Return Value
A size in the view's interior coordinate system.

Discussion
The layer's space is virtual, and doesn't take into account the layer's `contentsScale` setting.

The returned `NSSize` values are always forced to have positive a width and height.

Availability
Available in OS X v10.7 and later.

See Also
`– convertSizeToLayer:` (page 83)

Declared in
`NSView.h`

convertSizeToBacking:

Converts a size from the view's interior coordinate system to its pixel aligned backing store coordinate system.

```
- (NSSize)convertSizeToBacking:(NSSize)aSize
```

Parameters
`aSize`
The size in the view's interior coordinate system.

Return Value
The size in the pixel aligned coordinate coordinate system.

Availability
Available in OS X v10.7 and later.

See Also
– convertSizeToBacking: (page 82)

Declared in
NSView.h

convertSizeToBase:

Converts the size from the receiver's coordinate system to the base coordinate system. (Deprecated. Use convertSizeToBacking: (page 82) instead.)

```
– (NSSize)convertSizeToBase:(NSSize)aSize
```

Parameters
aSize

A size in the receiver's coordinate system

Return Value
The size converted to the base coordinate system.

Discussion
See "Transforming View Coordinates To and From Base Space" in *View Programming Guide* for a description of view coordinate to base coordinate conversion.

Availability
Available in OS X v10.5 and later.

Related Sample Code
Sketch+Accessibility

ZipBrowser

Declared in
NSView.h

convertSizeToLayer:

Convert the size from the view's interior coordinate system to the layer's interior coordinate system.

```
– (NSSize)convertSizeToLayer:(NSSize)aSize
```

Parameters

`aSize`

A size in the view's interior coordinate system.

Return Value

A size in the layer's interior coordinate system.

Discussion

The layer's space is virtual, and doesn't take into account the layer's `contentsScale` setting.

The returned `NSSize` values are always forced to have positive a width and height.

Availability

Available in OS X v10.7 and later.

See Also

`– convertSizeFromLayer:` (page 82)

Declared in

`NSView.h`

dataWithEPSInsideRect:

Returns EPS data that draws the region of the receiver within a specified rectangle.

```
– (NSData *)dataWithEPSInsideRect:(NSRect)aRect
```

Parameters

`aRect`

A rectangle defining the region.

Discussion

This data can be placed on an `NSPasteboard` object, written to a file, or used to create an `NSImage` object.

Availability

Available in OS X v10.0 and later.

See Also

`– writeEPSInsideRect:toPasteboard:` (page 213)

Declared in

`NSView.h`

dataWithPDFInsideRect:

Returns PDF data that draws the region of the receiver within a specified rectangle.

```
– (NSData *)dataWithPDFInsideRect:(NSRect)aRect
```

Parameters

`aRect`

A rectangle defining the region.

Discussion

This data can be placed on an `NSPasteboard` object, written to a file, or used to create an `NSImage` object.

Availability

Available in OS X v10.0 and later.

See Also

– `writePDFInsideRect:toPasteboard:` (page 213)

Related Sample Code

QuickLookSketch

Sketch

Sketch+Accessibility

Declared in

`NSView.h`

didAddSubview:

Overridden by subclasses to perform additional actions when subviews are added to the receiver.

```
– (void)didAddSubview:(NSView *)subview
```

Parameters

`subview`

The view that was added as a subview.

Discussion

This method is invoked by `addSubview:` (page 44).

Availability

Available in OS X v10.0 and later.

Declared in

`NSView.h`

discardCursorRects

Invalidates all cursor rectangles set up using `addCursorRect:cursor:` (page 43).

```
– (void)discardCursorRects
```

Discussion

You need never invoke this method directly; neither is it typically invoked during the invalidation of cursor rectangles. `NSWindow` automatically invalidates cursor rectangles in response to `invalidateCursorRectsForView:` and before the receiver's cursor rectangles are reestablished using `resetCursorRects` (page 149). This method is invoked just before the receiver is removed from a window and when the receiver is deallocated.

Availability

Available in OS X v10.0 and later.

See Also

`discardCursorRects` (NSWindow)

Related Sample Code

DragItemAround

Declared in

`NSView.h`

display

Displays the receiver and all its subviews if possible, invoking each of the `NSView` methods `lockFocus` (page 122), `drawRect:` (page 96), and `unlockFocus` (page 196) as necessary.

```
– (void)display
```

Discussion

If the receiver isn't opaque, this method backs up the view hierarchy to the first opaque ancestor, calculates the portion of the opaque ancestor covered by the receiver, and begins displaying from there.

Availability

Available in OS X v10.0 and later.

See Also

– `canDraw` (page 63)
– `opaqueAncestor` (page 132)
– `visibleRect` (page 207)

– `displayIfNeededIgnoringOpacity` (page 87)

Related Sample Code
CocoaSpeechSynthesisExample

Declared in
`NSView.h`

displayIfNeeded

Displays the receiver and all its subviews if any part of the receiver has been marked as needing display.

```
– (void)displayIfNeeded
```

Discussion
This method invokes the `NSView` methods `lockFocus` (page 122), `drawRect:` (page 96), and `unlockFocus` (page 196) as necessary. If the receiver isn't opaque, this method backs up the view hierarchy to the first opaque ancestor, calculates the portion of the opaque ancestor covered by the receiver, and begins displaying from there.

Availability
Available in OS X v10.0 and later.

See Also
– `display` (page 86)
– `needsDisplay` (page 127)
– `displayIfNeededIgnoringOpacity` (page 87)

Related Sample Code
ButtonMadness
CompositeLab
Rulers

Declared in
`NSView.h`

displayIfNeededIgnoringOpacity

Acts as `displayIfNeeded` (page 87), except that this method doesn't back up to the first opaque ancestor–it simply causes the receiver and its descendants to execute their drawing code.

```
– (void)displayIfNeededIgnoringOpacity
```

Availability
Available in OS X v10.0 and later.

Related Sample Code
DispatchFractal

Declared in
`NSView.h`

displayIfNeededInRect:

Acts as `displayIfNeeded` (page 87), confining drawing to a specified region of the receiver..

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

Parameters
`aRect`

A rectangle defining the region to be redrawn. It should be specified in the coordinate system of the receiver.

Availability
Available in OS X v10.0 and later.

Declared in
`NSView.h`

displayIfNeededInRectIgnoringOpacity:

Acts as `displayIfNeeded` (page 87), but confining drawing to `aRect` and not backing up to the first opaque ancestor–it simply causes the receiver and its descendants to execute their drawing code.

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

Parameters
`aRect`

A rectangle defining the region to be redrawn. It should be specified in the coordinate system of the receiver.

Availability
Available in OS X v10.0 and later.

Declared in
`NSView.h`

displayRect:

Acts as `display` (page 86), but confining drawing to a rectangular region of the receiver.

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

Parameters

`aRect`

A rectangle defining the region of the receiver to be redrawn; should be specified in the coordinate system of the receiver.

Availability

Available in OS X v10.0 and later.

Declared in

`NSView.h`

displayRectIgnoringOpacity:

Displays the receiver but confines drawing to a specified region and does not back up to the first opaque ancestor–it simply causes the receiver and its descendants to execute their drawing code.

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

Parameters

`aRect`

A rectangle defining the region of the receiver to be redrawn; should be specified in the coordinate system of the receiver.

Availability

Available in OS X v10.0 and later.

Declared in

`NSView.h`

displayRectIgnoringOpacity:inContext:

Causes the receiver and its descendants to be redrawn to the specified graphics context.

```
- (void)displayRectIgnoringOpacity:(NSRect)aRect inContext:(NSGraphicsContext
*)context
```

Parameters

`aRect`

A rectangle defining the region of the receiver to be redrawn. It should be specified in the coordinate system of the receiver.

`context`

The graphics context in which drawing will occur. See the discussion below for more about this parameter.

Discussion

Acts as `display` (page 86), but confines drawing to `aRect`. This method initiates drawing with the receiver, even if the receiver is not opaque. Appropriate scaling factors for the view are obtained from `context`.

If the `context` parameter represents the context for the window containing the view, then all of the necessary transformations are applied. This includes the application of the receiver's bounds and frame transforms along with any transforms it inherited from its ancestors. In this situation, the view is also marked as no longer needing an update for the specified rectangle.

If `context` specifies any other graphics context, then only the receiver's bounds transform is applied. This means that drawing is not constrained to the view's visible rectangle. It also means that any dirty rectangles are not cleared, since they are not being redrawn to the window.

Availability

Available in OS X v10.4 and later.

Related Sample Code
QuickLookSketch

Declared in

`NSView.h`

dragFile:fromRect:slideBack:event:

Initiates a dragging operation from the receiver, allowing the user to drag a file icon to any application that has window or view objects that accept files.

```
- (BOOL)dragFile:(NSString *)fullPath fromRect:(NSRect)aRect slideBack:(BOOL)slideBack
 event:(NSEvent *)theEvent
```

Parameters

`fullPath`

A string that specifies the absolute path for the file that is dragged.

`aRect`

A rectangle that describes the position of the icon in the receiver's coordinate system.

`slideBack`

A Boolean that indicates whether the icon being dragged should slide back to its position in the receiver if the file isn't accepted. The icon slides back to `aRect` if `slideBack` is `YES`, the file is not accepted by the dragging destination, and the user has not disabled icon animation; otherwise it simply disappears.

`theEvent`

The mouse-down event object from which to initiate the drag operation. In particular, its mouse location is used for the offset of the icon being dragged.

Return Value

`YES` if the receiver successfully initiates the dragging operation (which doesn't necessarily mean the dragging operation concluded successfully). Otherwise returns `NO`.

Discussion

This method must be invoked only within an implementation of the `mouseDown:` method.

See the `NSDraggingSource`, `NSDraggingInfo`, and `NSDraggingDestination` protocol specifications for more information on dragging operations.

Availability

Available in OS X v10.0 and later.

See Also

– `dragImage:at:offset:event:pasteboard:source:slideBack:` (page 91)
– `shouldDelayWindowOrderingForEvent:` (page 187)

Declared in

`NSView.h`

dragImage:at:offset:event:pasteboard:source:slideBack:

Initiates a dragging operation from the receiver, allowing the user to drag arbitrary data with a specified icon into any application that has window or view objects that accept dragged data.

```
– (void)dragImage:(NSImage *)anImage at:(NSPoint)imageLoc offset:(NSSize)mouseOffset
 event:(NSEvent *)theEvent pasteboard:(NSPasteboard *)pboard source:(id)sourceObject
 slideBack:(BOOL)slideBack
```

Parameters

`anImage`

The `NSImage` object to be dragged.

`imageLoc`
: The location of the image's lower-left corner, in the receiver's coordinate system. It determines the placement of the dragged image under the cursor. When determining the image location you should use the mouse down coordinate, provided in `theEvent`, rather than the current mouse location.

`mouseOffset`
: This parameter is ignored.

`theEvent`
: The left mouse-down event that triggered the dragging operation (see discussion below).

`pboard`
: The pasteboard that holds the data to be transferred to the destination (see discussion below).

`sourceObject`
: An object that serves as the controller of the dragging operation. It must conform to the `NSDraggingSource` protocol and is typically the receiver itself or its `NSWindow` object.

`slideBack`
: A Boolean that determines whether the drag image should slide back if it's rejected. The image slides back to `imageLoc` if `slideBack` is `YES` and the image isn't accepted by the dragging destination. If `NO` the image doesn't slide back.

Discussion

This method must be invoked only within an implementation of the `mouseDown:` or `mouseDragged:` methods.

Before invoking this method, you must place the data to be transferred on `pboard`. To do this, get the drag pasteboard object (`NSDragPboard`), declare the types of the data, and then put the data on the pasteboard. This code fragment initiates a dragging operation on an image itself (that is, the image is the data to be transferred):

```
- (void)mouseDown:(NSEvent *)theEvent
{
    NSSize dragOffset = NSMakeSize(0.0, 0.0);
    NSPasteboard *pboard;

    pboard = [NSPasteboard pasteboardWithName:NSDragPboard];
    [pboard declareTypes:[NSArray arrayWithObject:NSTIFFPboardType]  owner:self];
    [pboard setData:[[self image] TIFFRepresentation] forType:NSTIFFPboardType];

    [self dragImage:[self image] at:[self imageLocation] offset:dragOffset
        event:theEvent pasteboard:pboard source:self slideBack:YES];
```

```
    return;
}
```

See the `NSDraggingSource`, `NSDraggingInfo`, and `NSDraggingDestination` protocol specifications for more information on dragging operations.

Availability
Available in OS X v10.0 and later.

See Also
– `dragFile:fromRect:slideBack:event:` (page 90)
– `shouldDelayWindowOrderingForEvent:` (page 187)

Declared in
`NSView.h`

dragPromisedFilesOfTypes:fromRect:source:slideBack:event:

Initiates a dragging operation from the receiver, allowing the user to drag one or more promised files (or directories) into any application that has window or view objects that accept promised file data.

```
– (BOOL)dragPromisedFilesOfTypes:(NSArray *)typeArray fromRect:(NSRect)aRect
source:(id)sourceObject slideBack:(BOOL)slideBack event:(NSEvent *)theEvent
```

Parameters

`typeArray`
An array of file types being promised. The array elements can consist of file extensions and HFS types encoded with the `NSFileTypeForHFSTypeCode` function. If promising a directory of files, only include the top directory in the array.

`aRect`
A rectangle that describes the position of the icon in the receiver's coordinate system.

`sourceObject`
An object that serves as the controller of the dragging operation. It must conform to the `NSDraggingSource` protocol, and is typically the receiver itself or its `NSWindow` object.

`slideBack`

A Boolean that indicates whether the icon being dragged should slide back to its position in the receiver if the file isn't accepted. The icon slides back to `aRect` if `slideBack` is `YES`, the promised files are not accepted by the dragging destination, and the user has not disabled icon animation; otherwise it simply disappears.

`theEvent`

The mouse-down event object from which to initiate the drag operation. In particular, its mouse location is used for the offset of the icon being dragged.

Return Value

`YES` if the drag operation is initiated successfully, `NO` otherwise.

Discussion

This method must be invoked only within an implementation of the `mouseDown:` method. As part of its implementation, this method invokes `dragImage:at:offset:event:pasteboard:source:slideBack:` (page 91).

Promised files are files that do not exist, yet, but that the drag source, `sourceObject`, promises to create at a file system location specified by the drag destination when the drag is successfully dropped.

See *Drag and Drop Programming Topics* for more information on dragging operations.

Availability

Available in OS X v10.2 and later.

See Also

- `registerForDraggedTypes:` (page 142)
- `unregisterDraggedTypes` (page 197)
- `dragImage:at:offset:event:pasteboard:source:slideBack:` (page 91)
- `shouldDelayWindowOrderingForEvent:` (page 187)
- `beginDraggingSessionWithItems:event:source:` (page 58)

Related Sample Code

Quartz Composer ImageFX

Quartz Composer ImageResizer

Declared in

`NSView.h`

drawFocusRingMask

Draws the focus ring mask for the view.

```
- (void)drawFocusRingMask
```

Discussion
This method provides the shape of the focus ring mask by drawing the focus ring mask. An implementation of this method should draw in the view's interior (bounds) coordinate space, that the focus ring style has been set (it will be set it to `NSFocusRingOnly` to capture the focus ring itself), and that the fill and stroke colors have been set to an arbitrary fully opaque color.

Subclasses that find the default behavior insufficient should only draw the focus ring shape.

The `NSView` implementation of this method simply fills `[self bounds]`.

Availability
Available in OS X v10.7 and later.

See Also
– `focusRingMaskBounds` (page 102)
– `noteFocusRingMaskChanged` (page 131)

Declared in
`NSView.h`

drawPageBorderWithSize:

Allows applications that use the Application Kit pagination facility to draw additional marks on each logical page.

```
- (void)drawPageBorderWithSize:(NSSize)borderSize
```

Parameters
`borderSize`
An `NSSize` structure that defines a logical page.

Discussion
The marks can be such things as alignment marks or a virtual sheet border of size `borderSize`. The default implementation doesn't draw anything.

Availability
Available in OS X v10.0 and later.

See Also
– `drawSheetBorderWithSize:` (page 97)

Declared in
`NSView.h`

drawRect:

Overridden by subclasses to draw the receiver's image within the specified rectangle.

```
- (void)drawRect:(NSRect)dirtyRect
```

Parameters
`dirtyRect`

A rectangle defining the portion of the view that requires redrawing. This rectangle usually represents the portion of the view that requires updating. When responsive scrolling is enabled, this rectangle can also represent a nonvisible portion of the view that AppKit wants to cache.

Discussion
Use this method to draw the specified portion of your view's content. Your implementation of this method should be as fast as possible and do as little work as possible. The `dirtyRect` parameter helps you achieve better performance by specifying the portion of the view that needs to be drawn. You should always limit drawing to the content inside this rectangle. For even better performance, you can call the `getRectsBeingDrawn:count:` (page 106) method and use the list of rectangles returned by that method to limit drawing even further. You can also use the `needsToDrawRect:` (page 128) method test whether objects in a particular rectangle need to be drawn.

The default implementation does nothing. Subclasses should override this method if they do custom drawing. Prior to calling this method, AppKit creates an appropriate drawing context and configures it for drawing to the view; you do not need to configure the drawing context yourself. If your app manages content using its layer object instead, use the `updateLayer` (page 198) method to update your layer instead of overriding this method.

If your custom view is a direct `NSView` subclass, you do not need to call `super`. For all other views, call `super` at some point in your implementation so that the parent class can perform any additional drawing.

> **Important:** If the view's `isOpaque` (page 116) method returns `YES`, the view must completely fill the `dirtyRect` rectangle with opaque content.

For information about how to draw in your app, see *Cocoa Drawing Guide*.

Availability
Available in OS X v10.0 and later.

See Also

- `display` (page 86)
- `getRectsBeingDrawn:count:` (page 106)
- `isFlipped` (page 114)
- `needsToDrawRect:` (page 128)
- `setNeedsDisplayInRect:` (page 176)
- `shouldDrawColor` (page 188)
- `updateLayer` (page 198)

Related Sample Code

WebKitDOMElementPlugIn

Declared in

`NSView.h`

drawSheetBorderWithSize:

Allows applications that use the Application Kit pagination facility to draw additional marks on each printed sheet.

```
- (void)drawSheetBorderWithSize:(NSSize)borderSize
```

Parameters

`borderSize`

An `NSSize` structure that defines a printed sheet.

Discussion

The marks can be such things as crop marks or fold lines of size `borderSize`. This method has been deprecated.

Availability

Available in OS X v10.0 and later.

See Also

- `drawPageBorderWithSize:` (page 95)

Declared in

`NSView.h`

enclosingMenuItem

Returns the menu item containing the receiver or any of its superviews in the view hierarchy.

```
- (NSMenuItem *)enclosingMenuItem
```

Return Value
Returns the menu item containing the receiver or any of its superviews in the view hierarchy, or `nil` if the receiver's view hierarchy is not in a menu item

Availability
Available in OS X v10.5 and later.

Declared in
`NSMenuItem.h`

enclosingScrollView

Returns the nearest ancestor `NSScrollView` object containing the receiver (not including the receiver itself); otherwise returns `nil`.

```
- (NSScrollView *)enclosingScrollView
```

Availability
Available in OS X v10.0 and later.

Related Sample Code
CIAnnotation
OverlayView
Rulers
Sketch
Sketch+Accessibility

Declared in
`NSView.h`

endDocument

This method is invoked at the end of the printing session.

```
- (void)endDocument
```

Discussion
If you override this method, call the superclass implementation.

Availability
Available in OS X v10.0 and later.

Declared in
NSView.h

endPage

Writes the end of a conforming page.

```
- (void)endPage
```

Discussion
This method is invoked after each page is printed. It invokes `unlockFocus` (page 196). This method also generates comments for the bounding box and page fonts, if they were specified as being at the end of the page.

Availability
Available in OS X v10.0 and later.

Declared in
NSView.h

enterFullScreenMode:withOptions:

Sets the receiver to full screen mode.

```
- (BOOL)enterFullScreenMode:(NSScreen *)screen withOptions:(NSDictionary *)options
```

Parameters
`screen`
The screen the receiver should cover.

`options`
A dictionary of options for the mode. For possible keys, see `"Full Screen Mode Options"` (page 217).

Return Value
`YES` if the receiver was able to enter full screen mode, otherwise `NO`.

Discussion
When the `NSFullScreenModeApplicationPresentationOptions` (page 218) is contained in the options dictionary, the presentation options that were in effect when this method is invoked are not altered, and no displays are captured.

If you do not wish to capture the screen when going to full screen mode, you can add `NSFullScreenModeApplicationPresentationOptions` (page 218) to the options dictionary with the value returned by the `presentationOptions`.

When the `NSFullScreenModeApplicationPresentationOptions` (page 218) options is specified, exiting full screen mode using `exitFullScreenModeWithOptions:` (page 101) will restore the previously active `presentationOptions`.

Special Considerations
On OS X v 10.5 invoking this method when the receiver was not in a window would cause an exception. On OS X v 10.6 and later, you can now send this message to a view not in a window. For applications that must also run on OS X v 10.5, a simple workaround is to place the view in an offscreen dummy window.

Availability
Available in OS X v10.5 and later.

See Also
– `exitFullScreenModeWithOptions:` (page 101)
– `isInFullScreenMode` (page 116)

Related Sample Code
CoreAnimationKioskStyleMenu
From A View to A Movie
From A View to A Picture
GL3 Text
Quartz Composer RepositoryBrowser

Declared in
`NSView.h`

exerciseAmbiguityInLayout

Randomly changes the frame of a view with an ambiguous layout between the different valid values.

```
– (void)exerciseAmbiguityInLayout
```

Discussion
This method randomly changes the frame of a view with an ambiguous layout between its different valid values, causing the view to move in the interface. This makes it easy to visually identify what the valid frames are and may enable the developer to discern what constraints need to be added to the layout to fully specify a location for the view.

This method should only be used for debugging constraint-based layout. No application should ship with calls to this method as part of its operation.

Availability
Available in OS X v10.7 and later.

See Also
– `hasAmbiguousLayout` (page 108)

Declared in
`NSLayoutConstraint.h`

exitFullScreenModeWithOptions:

Instructs the receiver to exit full screen mode.

```
– (void)exitFullScreenModeWithOptions:(NSDictionary *)options
```

Parameters
`options`

A dictionary of options for the mode. For possible keys, see "`Full Screen Mode Options`" (page 217).

Discussion
When the `NSFullScreenModeApplicationPresentationOptions` (page 218) options is specified when `enterFullScreenMode:withOptions:` (page 99) is invoked, exiting full screen mode will restore the previously active `presentationOptions`.

Availability
Available in OS X v10.5 and later.

See Also
– `enterFullScreenMode:withOptions:` (page 99)
– `isInFullScreenMode` (page 116)

Related Sample Code
From A View to A Movie
From A View to A Picture
GL3 Text
Quartz Composer RepositoryBrowser

Declared in
`NSView.h`

fittingSize

Returns the minimum size of the view that satisfies the constraints it holds.

```
- (NSSize)fittingSize
```

Return Value
The minimum size of the view that satisfies the constraints it holds.

Discussion
Determines the best size of the view considering all constraints it holds and those of its subviews, together with a preference for the view itself to be as small as possible.

Availability
Available in OS X v10.7 and later.

See Also
`NSLayoutPriorityFittingSizeCompression`

Declared in
`NSLayoutConstraint.h`

focusRingMaskBounds

Returns the focus ring mask bounds.

```
- (NSRect)focusRingMaskBounds
```

Return Value
A rectangle containing the mask in the view's interior (bounds) coordinate space.

Discussion
The mask bounds allows the focus ring's overall size and position to be determined before it is drawn.

Subclasses must override this method if they require the display of a focus ring.

The `NSView` implementation of this method simply returns `NSZeroRect`.

> **Note:** The information provided by `focusRingMaskBounds` will enable Accessibility to identify selected subelements for zoom tracking, so it is important that this method provide a reasonably tight bounding box and that `noteFocusRingMaskChanged` (page 131) is invoked as described.

Availability
Available in OS X v10.7 and later.

See Also
– `noteFocusRingMaskChanged` (page 131)
– `drawFocusRingMask` (page 94)

Declared in
`NSView.h`

focusRingType

Returns the type of focus ring drawn around the receiver.

```
– (NSFocusRingType)focusRingType
```

Return Value
An `enum` constant identifying a type of focus ring. Possible values are listed in `NSFocusRingType`.

Discussion
You can disable a view's drawing of its focus ring by overriding this method to return `NSFocusRingTypeNone`, or by invoking `setFocusRingType:` (page 166) with the argument `NSFocusRingTypeNone`.

You should only disable the default drawing of a view's focus ring if you want it to draw its own focus ring (for example, setting the background color of the view), or if the view does not have sufficient space to display a focus ring.

Availability
Available in OS X v10.3 and later.

See Also
– `setFocusRingType:` (page 166)

Declared in
`NSView.h`

frame

Returns the receiver's frame rectangle, which defines its position in its superview.

```
– (NSRect)frame
```

Discussion
The frame rectangle may be rotated; use the `frameRotation` (page 105) method to check this.

Availability
Available in OS X v10.0 and later.

See Also
– `bounds` (page 60)
– `setFrame:` (page 167)

Related Sample Code
CustomMenus

MatrixMixerTest

TextEdit

TextSizingExample

TreeView

Declared in
`NSView.h`

frameCenterRotation

Returns the receiver's rotation about the layer's position.

```
– (CGFloat)frameCenterRotation
```

Return Value
The angle of rotation of the frame around the center of the receiver.

Discussion
If the application has altered the layer's `anchorPoint` property, the behavior is undefined. Sending this message to a view that is not managing a Core Animation layer causes an exception.

Availability
Available in OS X v10.5 and later.

Declared in
`NSView.h`

frameForAlignmentRect:

Returns the view's frame for a given alignment rectangle.

```
– (NSRect)frameForAlignmentRect:(NSRect)alignmentRect
```

Parameters

`alignmentRect`

The alignment rectangle whose corresponding frame is desired.

Return Value

The frame for the specified alignment rectangle

Discussion

The constraint-based layout system uses alignment rectangles to align views, rather than their frame. This allows custom views to be aligned based on the location of their content while still having a frame that encompasses any ornamentation they need to draw around their content, such as shadows or reflections.

The default implementation returns `alignmentRect` modified by the view's `alignmentRectInsets` (page 52). Most custom views can override `alignmentRectInsets` to specify the location of their content within their frame. Custom views that require arbitrary transformations can override `alignmentRectForFrame:` (page 51) and `frameForAlignmentRect:` to describe the location of their content. These two methods must always be inverses of each other.

Availability

Available in OS X v10.7 and later.

Declared in

`NSLayoutConstraint.h`

frameRotation

Returns the angle, in degrees, of the receiver's frame relative to its superview's coordinate system.

```
- (CGFloat)frameRotation
```

Availability

Available in OS X v10.0 and later.

See Also

- `setFrameRotation:` (page 169)
- `boundsRotation` (page 61)

Declared in

`NSView.h`

getRectsBeingDrawn:count:

Returns by indirection a list of non-overlapping rectangles that define the area the receiver is being asked to draw in `drawRect:` (page 96).

```
- (void)getRectsBeingDrawn:(const NSRect **)rects count:(NSInteger *)count
```

Parameters

`rects`

On return, contains a list of non-overlapping rectangles defining areas to be drawn in. The rectangles returned in `rects` are in the coordinate space of the receiver.

`count`

On return, the number of rectangles in the `rects` list.

Discussion

An implementation of `drawRect:` can use this information to test whether objects or regions within the view intersect with the rectangles in the list, and thereby avoid unnecessary drawing that would be completely clipped away.

The `needsToDrawRect:` (page 128) method gives you a convenient way to test individual objects for intersection with the area being drawn in `drawRect:` (page 96). However, you may want to retrieve and directly inspect the rectangle list if this is a more efficient way to perform intersection testing.

You should send this message only from within a `drawRect:` (page 96) implementation. The `aRect` parameter of `drawRect:` is the rectangle enclosing the returned list of rectangles; you can use it in an initial pass to reject objects that are clearly outside the area to be drawn.

Availability

Available in OS X v10.3 and later.

See Also

– `wantsDefaultClipping` (page 209)

Declared in

`NSView.h`

getRectsExposedDuringLiveResize:count:

Returns a list of rectangles indicating the newly exposed areas of the receiver.

```
- (void)getRectsExposedDuringLiveResize:(NSRect)exposedRects count:(NSInteger *)count
```

Parameters

`exposedRects`

On return, contains the list of rectangles. The returned rectangles are in the coordinate space of the receiver.

`count`

Contains the number of rectangles in `exposedRects`; this value may be 0 and is guaranteed to be no more than 4.

Discussion

If your view does not support content preservation during live resizing, the entire area of your view is returned in the `exposedRects` parameter. To support content preservation, override `preservesContentDuringLiveResize` (page 136) in your view and have your implementation return `YES`.

> **Note:** The window containing your view must also support content preservation. To enable support for this feature in your window, use the `setPreservesContentDuringLiveResize:` method of `NSWindow`.

If the view decreased in both height and width, the list of returned rectangles will be empty. If the view increased in both height and width and its upper-left corner stayed anchored in the same position, the list of returned rectangles will contain a vertical and horizontal component indicating the exposed area.

Availability

Available in OS X v10.4 and later.

See Also

– `preservesContentDuringLiveResize` (page 136)
– `rectPreservedDuringLiveResize` (page 141)

Declared in

`NSView.h`

gState

Returns the identifier for the receiver's graphics state object, or 0 if the receiver doesn't have a graphics state object.

```
– (NSInteger)gState
```

Discussion

A view object's graphics state object is recreated from scratch whenever the view is focused, unless the `allocateGState` (page 52) method has been invoked. So if the receiver hasn't been focused or hasn't received the `allocateGState` (page 52) message, this method returns 0.

Although applications rarely need to use the value returned by `gState` (page 107), it can be passed to the few methods that take an object identifier as a parameter.

Availability

Available in OS X v10.0 and later.

See Also

- `allocateGState` (page 52)
- `setUpGState` (page 183)
- `renewGState` (page 148)
- `releaseGState` (page 143)
- `lockFocus` (page 122)

Declared in

`NSView.h`

hasAmbiguousLayout

Returns whether the constraints impacting the layout of the view incompletely specify the location of the view.

```
- (BOOL)hasAmbiguousLayout
```

Return Value

`YES` if the view's location is incompletely specified, `NO` otherwise.

Discussion

This method checks to see if there is any other frame the view could have that would also satisfy the constraints on the view. This is an expensive operation and is not run as part of the normal layout process, but can be useful when debugging whether a given interface has been specified with a sufficient number of constraints to ensure consistent layout. This method is automatically invoked when a window has been told to visualize constraints with the `visualizeConstraints:` method.

This method should only be used for debugging constraint-based layout. No application should ship with calls to this method as part of its operation.

Availability

Available in OS X v10.7 and later.

See Also
– `exerciseAmbiguityInLayout` (page 100)

Declared in
`NSLayoutConstraint.h`

heightAdjustLimit

Returns the fraction (from 0.0 to 1.0) of the page that can be pushed onto the next page during automatic pagination to prevent items such as lines of text from being divided across pages.

```
- (CGFloat)heightAdjustLimit
```

Discussion
This fraction is used to calculate the bottom edge limit for an `adjustPageHeightNew:top:bottom:limit:` (page 48) message.

Availability
Available in OS X v10.0 and later.

See Also
– `widthAdjustLimit` (page 211)

Declared in
`NSView.h`

hitTest:

Returns the farthest descendant of the receiver in the view hierarchy (including itself) that contains a specified point, or `nil` if that point lies completely outside the receiver.

```
- (NSView *)hitTest:(NSPoint)aPoint
```

Parameters
`aPoint`

A point that is in the coordinate system of the receiver's superview, not of the receiver itself.

Return Value
A view object that is the farthest descendent of `aPoint`.

Discussion

This method is used primarily by an `NSWindow` object to determine which view should receive a mouse-down event. You'd rarely need to invoke this method, but you might want to override it to have a view object hide mouse-down events from its subviews. This method ignores hidden views.

Availability

Available in OS X v10.0 and later.

See Also

– `mouse:inRect:` (page 125)
– `convertPoint:toView:` (page 69)
– `setHidden:` (page 171)

Related Sample Code

TableViewPlayground

TargetGallery

Declared in

`NSView.h`

initWithFrame:

Initializes and returns a newly allocated `NSView` object with a specified frame rectangle.

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

Parameters

`frameRect`

The frame rectangle for the created view object.

Return Value

An initialized `NSView` object or nil if the object couldn't be created.

Discussion

The new view object must be inserted into the view hierarchy of a window before it can be used. This method is the designated initializer for the `NSView` class. Returns an initialized object.

Availability

Available in OS X v10.0 and later.

See Also

– `addSubview:` (page 44)
– `addSubview:positioned:relativeTo:` (page 45)

– setFrame: (page 167)

Related Sample Code
CocoaSlides
DragItemAround
Movie Overlay
SonOfSillyBalls
TreeView

Declared in
NSView.h

inLiveResize

A convenience method, expected to be called from drawRect: (page 96), to assist in decisions about optimized drawing.

```
– (BOOL)inLiveResize
```

Return Value
YES if the receiver is in a live-resize operation, NO otherwise.

Availability
Available in OS X v10.1 and later.

See Also
– viewDidEndLiveResize (page 201)
– viewWillStartLiveResize (page 206)

Related Sample Code
HoverTableDemo
MatrixMixerTest

Declared in
NSView.h

inputContext

Returns the text input context object for the receiver.

```
– (NSTextInputContext *)inputContext
```

Return Value
The text input context object, or `nil` the receiver doesn't conform to `NSTextInputClient` protocol.

Availability
Available in OS X v10.6 and later.

Related Sample Code
TextInputView

Declared in
`NSView.h`

intrinsicContentSize

Returns the natural size for the receiving view, considering only properties of the view itself.

```
– (NSSize)intrinsicContentSize
```

Return Value
A size indicating the natural size for the receiving view based on its intrinsic properties.

Discussion
Custom views typically have content that they display of which the layout system is unaware. Overriding this method allows a custom view to communicate to the layout system what size it would like to be based on its content. This intrinsic size must be independent of the content frame, because there's no way to dynamically communicate a changed width to the layout system based on a changed height, for example.

If a custom view has no intrinsic size for a given dimension, it can return `NSViewNoInstrinsicMetric` (page 222) for that dimension.

Availability
Available in OS X v10.7 and later.

Declared in
`NSLayoutConstraint.h`

invalidateIntrinsicContentSize

Invalidates the view's intrinsic content size.

```
– (void)invalidateIntrinsicContentSize
```

Discussion

Call this when something changes in your custom view that invalidates its intrinsic content size. This allows the constraint-based layout system to take the new intrinsic content size into account in its next layout pass.

Availability

Available in OS X v10.7 and later.

Declared in

`NSLayoutConstraint.h`

isDescendantOf:

Returns `YES` if the receiver is a subview of a given view or if it's identical to that view; otherwise, it returns `NO`.

```
- (BOOL)isDescendantOf:(NSView *)aView
```

Parameters

`aView`

The view to test for subview relationship within the view hierarchy.

Discussion

The method returns `YES` if the receiver is either an immediate or distant subview of `aView`.

Availability

Available in OS X v10.0 and later.

See Also

- `superview` (page 192)
- `subviews` (page 191)
- `ancestorSharedWithView:` (page 54)

Declared in

`NSView.h`

isDrawingFindIndicator

Returns when the view or one of its ancestors is being drawn for a find indicator.

```
- (BOOL)isDrawingFindIndicator
```

Return Value

`YES` if the view is being drawn during find; otherwise `NO`.

Discussion

When this method returns YES the view contents are being drawn such that they are easily readable against the find indicator background.

Availability

Available in OS X v10.7 and later.

Declared in

NSView.h

isFlipped

Returns YES if the receiver uses flipped drawing coordinates or NO if it uses native coordinates.

```
- (BOOL)isFlipped
```

Discussion

The default implementation returns NO; subclasses that use flipped coordinates should override this method to return YES.

Availability

Available in OS X v10.0 and later.

Related Sample Code

ImageMap

ImageMapExample

Rulers

Sketch+Accessibility

ZipBrowser

Declared in

NSView.h

isHidden

Returns whether the receiver is marked as hidden.

```
- (BOOL)isHidden
```

Discussion

The return value reflects the state of the receiver only, as set in Interface Builder or through the most recent `setHidden:` (page 171) message, and does not account for the state of the receiver's ancestors in the view hierarchy, Thus this method returns `NO` when the receiver is effectively hidden because it has a hidden ancestor. See `setHidden:` for a discussion of the mechanics and implications of hidden views.

If you want to determine whether a view is effectively hidden, for whatever reason, send the `isHiddenOrHasHiddenAncestor` (page 115) to the view instead.

Availability

Available in OS X v10.3 and later.

Related Sample Code

CustomMenus

GLSLShowpiece

JavaScriptCoreHeadstart

SidebarDemo

Declared in

`NSView.h`

isHiddenOrHasHiddenAncestor

Returns `YES` if the receiver is marked as hidden or has an ancestor in the view hierarchy that is marked as hidden; returns `NO` otherwise.

```
- (BOOL)isHiddenOrHasHiddenAncestor
```

Discussion

The return value reflects state set through the `setHidden:` (page 171) method in the receiver of one of its ancestors in the view hierarchy. It does not account for other reasons why a view might be considered hidden, such as being positioned outside its superview's bounds, not having a window, or residing in a window that is offscreen or overlapped by another window.

Availability

Available in OS X v10.3 and later.

See Also

– `isHidden` (page 114)

Declared in

`NSView.h`

isInFullScreenMode

Returns whether the view is in full screen mode.

```
- (BOOL)isInFullScreenMode
```

Return Value

YES if the receiver is in full screen mode, otherwise NO.

Availability

Available in OS X v10.5 and later.

See Also

- enterFullScreenMode:withOptions: (page 99)
- exitFullScreenModeWithOptions: (page 101)

Related Sample Code

From A View to A Movie

From A View to A Picture

Declared in

NSView.h

isOpaque

Overridden by subclasses to return YES if the receiver is opaque, NO otherwise.

```
- (BOOL)isOpaque
```

Discussion

A view object is opaque if it completely covers its frame rectangle when drawing itself. The default implementation performs no drawing at all and so returns NO.

Availability

Available in OS X v10.0 and later.

See Also

- opaqueAncestor (page 132)
- displayRectIgnoringOpacity: (page 89)
- displayIfNeededIgnoringOpacity (page 87)
- displayIfNeededInRectIgnoringOpacity: (page 88)

Related Sample Code

GL3 Text

ImageApp
MultiGPUIOSurface
Sketch
Sketch+Accessibility

Declared in
NSView.h

isRotatedFromBase

Returns YES if the receiver or any of its ancestors has ever received a `setFrameRotation:` *(page 169) or* `setBoundsRotation:` *(page 161) message; otherwise returns NO.*

```
- (BOOL)isRotatedFromBase
```

Discussion
The intent of this information is to optimize drawing and coordinate calculation, not necessarily to reflect the exact state of the receiver's coordinate system, so it may not reflect the actual rotation. For example, if an NSView object is rotated to 45 degrees and later back to 0, this method still returns YES.

Availability
Available in OS X v10.0 and later.

See Also
– `frameRotation` (page 105)
– `boundsRotation` (page 61)

Declared in
NSView.h

isRotatedOrScaledFromBase

Returns YES if the receiver or any of its ancestors has ever had a nonzero frame or bounds rotation, or has been scaled from the window's base coordinate system; otherwise returns NO.

```
- (BOOL)isRotatedOrScaledFromBase
```

Discussion
The intent of this information is to optimize drawing and coordinate calculation, not necessarily to reflect the exact state of the receiver's coordinate system, so it may not reflect the actual rotation or scaling. For example, if an NSView object is rotated to 45 degrees and later back to 0, this method still returns YES.

Availability
Available in OS X v10.0 and later.

See Also
- `frameRotation` (page 105)
- `boundsRotation` (page 61)
- `centerScanRect:` (page 64)
- `setBounds:` (page 159)
- `setBoundsSize:` (page 162)
- `scaleUnitSquareToSize:` (page 152)

Declared in
`NSView.h`

knowsPageRange:

Returns `YES` if the receiver handles page boundaries, `NO` otherwise.

```
- (BOOL)knowsPageRange:(NSRangePointer)aRange
```

Parameters
`aRange`

On return, holds the page range if `YES` is returned directly. Page numbers are one-based—that is pages run from one to *N*.

Discussion
Returns `NO` if the receiver uses the default auto-pagination mechanism. The default implementation returns `NO`. Override this method if your class handles page boundaries.

Availability
Available in OS X v10.0 and later.

Declared in
`NSView.h`

layer

Returns the Core Animation layer that the receiver uses as its backing store.

```
- (CALayer *)layer
```

Return Value
The Core Animation layer the receiver uses as its backing store.

Availability
Available in OS X v10.5 and later.

Related Sample Code
AVSimpleEditorOSX
avvideowall
CALayerEssentials
CoreBluetooth: Heart Rate Monitor
Quartz2DBasics

Declared in
`NSView.h`

layerContentsPlacement

Returns the current layer contents placement policy.

```
- (NSViewLayerContentsPlacement)layerContentsPlacement
```

Return Value
The placement policy. See "NSViewLayerContentsPlacement" (page 219) for supported values.

Availability
Available in OS X v10.6 and later.

See Also
– `setLayerContentsPlacement:` (page 173)

Declared in
`NSView.h`

layerContentsRedrawPolicy

Returns the view's layer contents redraw policy.

```
- (NSViewLayerContentsRedrawPolicy)layerContentsRedrawPolicy
```

Return Value
The current redraw policy. See "NSViewLayerContentsRedrawPolicy" (page 218) for supported values.

Discussion

The `layerContentsRedrawPolicy` (page 119) and `layerContentsPlacement` (page 119) settings can have significant impacts on performance. See `setLayerContentsRedrawPolicy:` (page 174) and `setLayerContentsPlacement:` (page 173) for more information.

Availability

Available in OS X v10.6 and later.

See Also

– `setLayerContentsRedrawPolicy:` (page 174)

Declared in

`NSView.h`

layerUsesCoreImageFilters

Returns a Boolean value indicating whether the view's layer uses Core Image filters.

```
- (BOOL)layerUsesCoreImageFilters
```

Return Value

`YES` if the view's layer uses Core Image filters or `NO` if it does not.

Discussion

Use the `setLayerUsesCoreImageFilters:` method to set the value returned by this method. You must update the value returned by this method if your view's layer uses Core Image filters.

Availability

Available in OS X v10.9 and later.

See Also

– `setLayerUsesCoreImageFilters:` (page 175)

Declared in

`NSView.h`

layout

Perform layout in concert with the constraint-based layout system.

```
- (void)layout
```

Discussion

Override this method if your custom view needs to perform custom layout not expressible using the constraint-based layout system. In this case you are responsible for calling `setNeedsLayout:` (page 177) when something that impacts your custom layout changes.

You may not invalidate any constraints as part of your layout phase, nor invalidate the layout of your superview or views outside of your view hierarchy. You also may not invoke a drawing pass as part of layout.

You must call `[super layout]` as part of your implementation.

Availability

Available in OS X v10.7 and later.

Related Sample Code

CocoaSlides

Declared in

`NSLayoutConstraint.h`

layoutSubtreeIfNeeded

Updates the layout of the receiving view and its subviews based on the current views and constraints.

```
– (void)layoutSubtreeIfNeeded
```

Discussion

Before displaying a view that uses constraints-based layout the system invokes this method to ensure that the layout of the view and its subviews is up to date. This method updates the layout if needed, first invoking `updateConstraintsForSubtreeIfNeeded` (page 198) to ensure that all constraints are up to date. This method is called automatically by the system, but may be invoked manually if you need to examine the most up to date layout.

Subclasses should not override this method.

Availability

Available in OS X v10.7 and later.

Declared in

`NSLayoutConstraint.h`

locationOfPrintRect:

Invoked by `print:` (page 138) to determine the location of the region of the receiver being printed on the physical page.

```
- (NSPoint)locationOfPrintRect:(NSRect)aRect
```

Parameters

`aRect`

A rectangle defining a region of the receiver; it is expressed in the default coordinate system of the page.

Return Value

A point to be used for setting the origin for `aRect`, whose size the receiver can examine in order to properly place it. It is expressed in the default coordinate system of the page.

Discussion

The default implementation places `aRect` according to the status of the `NSPrintInfo` object for the print job. By default it places the image in the upper-left corner of the page, but if the `NSPrintInfo` methods `isHorizontallyCentered` or `isVerticallyCentered` return `YES`, it centers a single-page image along the appropriate axis. A multiple-page document, however, is always placed so the divided pieces can be assembled at their edges.

Availability

Available in OS X v10.0 and later.

Declared in

`NSView.h`

lockFocus

Locks the focus on the receiver, so subsequent commands take effect in the receiver's window and coordinate system.

```
- (void)lockFocus
```

Discussion

If you don't use a `display...` method to draw an `NSView` object, you must invoke `lockFocus` before invoking methods that send commands to the window server, and must balance it with an `unlockFocus` (page 196) message when finished.

Hiding or miniaturizing a one-shot window causes the backing store for that window to be released. If you don't use the standard display mechanism to draw, you should use `lockFocusIfCanDraw` (page 123) rather than `lockFocus` if there is a chance of drawing while the window is either miniaturized or hidden.

Availability
Available in OS X v10.0 and later.

See Also
`+ focusView` (page 39)
`– display` (page 86)
`– drawRect:` (page 96)
`– lockFocusIfCanDraw` (page 123)

Related Sample Code
BlastApp
GLFullScreen

Declared in
`NSView.h`

lockFocusIfCanDraw

Locks the focus to the receiver atomically if the `canDraw` method returns `YES` and returns the value of `canDraw`.

```
– (BOOL)lockFocusIfCanDraw
```

Discussion
Your thread will not be preempted by other threads between the `canDraw` method and the lock. This method fails to lock focus and returns `NO`, when the receiver is hidden and the current context is drawing to the screen (as opposed to a printing context).

Availability
Available in OS X v10.0 and later.

Declared in
`NSView.h`

lockFocusIfCanDrawInContext:

Locks the focus to the receiver atomically if drawing can occur in the specified graphics context.

```
– (BOOL)lockFocusIfCanDrawInContext:(NSGraphicsContext *)context
```

Parameters

`context`

The graphics context in which drawing might occur. See the discussion for the implications of the type of context.

Return Value

`YES` if successful; otherwise, returns `NO`.

Discussion

Your thread will not be preempted by other threads between the `canDraw` method and the lock.

If the `context` parameter represents the context for the window containing the view, then all of the necessary transformations are applied. This includes the application of the receiver's bounds and frame transforms along with any transforms it inherited from its ancestors. If `context` specifies any other graphics context, then only the receiver's bounds transform is applied.

Special Considerations

Important: This method was declared in OS X v10.4, but is not used in that release. It currently does nothing and returns `NO`. However, it might be implemented in a future release.

Availability

Available in OS X v10.4 and later.

See Also

– `lockFocus` (page 122)
– `lockFocusIfCanDraw` (page 123)

Declared in

`NSView.h`

makeBackingLayer

Creates the view's backing layer.

```
– (CALayer *)makeBackingLayer
```

Return Value

A layer to use as the view's backing layer.

Availability

Available in OS X v10.6 and later.

Related Sample Code
TreeView

Declared in
`NSView.h`

menuForEvent:

Overridden by subclasses to return a context-sensitive pop-up menu for a given mouse-down event.

```
- (NSMenu *)menuForEvent:(NSEvent *)theEvent
```

Parameters
`theEvent`
An object representing a mouse-down event.

Discussion
The receiver can use information in the mouse event, such as its location over a particular element of the receiver, to determine what kind of menu to return. For example, a text object might display a text-editing menu when the cursor lies over text and a menu for changing graphics attributes when the cursor lies over an embedded image.

The default implementation returns the receiver's normal menu.

Availability
Available in OS X v10.0 and later.

See Also
`+ defaultMenu` (page 39)
`menu` (`NSResponder`)

Declared in
`NSView.h`

mouse:inRect:

Returns whether a region of the receiver contains a specified point, accounting for whether the receiver is flipped or not.

```
- (BOOL)mouse:(NSPoint)aPoint inRect:(NSRect)aRect
```

Parameters

`aPoint`

A point that is expressed in the receiver's coordinate system. This point generally represents the hot spot of the mouse cursor.

`aRect`

A rectangle that is expressed in the receiver's coordinate system.

Return Value

`YES` if `aRect` contains `aPoint`, `NO` otherwise.

Discussion

Point-in-rectangle functions generally assume that the bottom edge of a rectangle is outside of the rectangle boundaries, while the upper edge is inside the boundaries. This method views `aRect` from the point of view of the user—that is, this method always treats the bottom edge of the rectangle as the one closest to the bottom edge of the user's screen. By making this adjustment, this function ensures consistent mouse-detection behavior from the user's perspective.

Never use the Foundation's `NSPointInRect` function as a substitute for this method. It doesn't account for flipped coordinate systems.

Availability

Available in OS X v10.0 and later.

See Also

`– hitTest:` (page 109)
`– isFlipped` (page 114)
`NSMouseInRect` (Foundation functions)
`– convertPoint:fromView:` (page 69)

Declared in

`NSView.h`

mouseDownCanMoveWindow

Returns `YES` if the receiver does not need to handle a mouse down and can pass it through to superviews; `NO` if it needs to handle the mouse down.

```
– (BOOL)mouseDownCanMoveWindow
```

Discussion
This allows iApp-type applications to determine the region by which a window can be moved. By default, this method returns `NO` if the view is opaque; otherwise, it returns `YES`. Subclasses can override this method to return a different value.

Availability
Available in OS X v10.2 and later.

Related Sample Code
SpeedometerView

Declared in
`NSView.h`

needsDisplay

Returns whether the view needs to be redrawn before being displayed.

```
- (BOOL)needsDisplay
```

Discussion
The `displayIfNeeded...` methods check this status to avoid unnecessary drawing, and all display methods clear this status to indicate that the view is up to date.

Availability
Available in OS X v10.0 and later.

Declared in
`NSView.h`

needsLayout

Returns whether the view needs a layout pass before it can be drawn.

```
- (BOOL)needsLayout
```

Return Value
`YES` if the view needs a layout pass, `NO` otherwise.

Availability
Available in OS X v10.7 and later.

See Also
`- setNeedsLayout:` (page 177)

Declared in
`NSLayoutConstraint.h`

needsPanelToBecomeKey

Overridden by subclasses to determine if the receiver requires its panel, which might otherwise avoid becoming key, to become the key window so that it can handle keyboard input and navigation.

```
- (BOOL)needsPanelToBecomeKey
```

Return Value
Returns `YES` if it should be come key, `NO` otherwise.

Discussion
Such a subclass should also override `acceptsFirstResponder` to return `YES`.

This method is also used in keyboard navigation. It determines if a mouse click should give focus to a view (make it first responder). Some views will want to get keyboard focus when you click in them, for example text fields. Other views should only get focus if you tab to them, for example, buttons. You wouldn't want focus to shift from a textfield that has editing in progress simply because you clicked on a check box.

The default implementation returns `NO`.

Availability
Available in OS X v10.0 and later.

See Also
`becomesKeyOnlyIfNeeded` (`NSPanel`)

Declared in
`NSView.h`

needsToDrawRect:

Returns whether the specified rectangle intersects any part of the area that the receiver is being asked to draw.

```
- (BOOL)needsToDrawRect:(NSRect)aRect
```

Parameters
`aRect`

A rectangle defining a region of the receiver.

Discussion

You typically send this message from within a `drawRect:` (page 96) implementation. It gives you a convenient way to determine whether any part of a given graphical entity might need to be drawn. It is optimized to efficiently reject any rectangle that lies outside the bounding box of the area the receiver is being asked to draw in `drawRect:`.

Availability

Available in OS X v10.3 and later.

Declared in

`NSView.h`

needsUpdateConstraints

Returns whether the view's constraints need updating.

```
- (BOOL)needsUpdateConstraints
```

Return Value

`YES` if the view's constraints need updating, `NO` otherwise.

Discussion

The constraint-based layout system uses the return value of this method to determine whether it needs to call `updateConstraints` (page 197) on your view as part of its normal layout pass.

Availability

Available in OS X v10.7 and later.

See Also

– `setNeedsUpdateConstraints:` (page 178)

Declared in

`NSLayoutConstraint.h`

nextKeyView

Returns the view object following the receiver in the key view loop.

```
- (NSView *)nextKeyView
```

Return Value

Returns the view object following the receiver in the key view loop, or `nil` if there is none.

Discussion
This view should, if possible, be made first responder when the user navigates forward from the receiver using keyboard interface control.

Availability
Available in OS X v10.0 and later.

See Also
- `nextValidKeyView` (page 130)
- `setNextKeyView:` (page 178)
- `previousKeyView` (page 137)
- `previousValidKeyView` (page 137)

`selectNextKeyView:` (`NSWindow`)
`selectKeyViewFollowingView:` (`NSWindow`)
`selectPreviousKeyView:` (`NSWindow`)
`selectKeyViewPrecedingView:` (`NSWindow`)

Related Sample Code
TrackBall

Declared in
`NSView.h`

nextValidKeyView

Returns the closest view object in the key view loop that follows the receiver and accepts first responder status.

```
- (NSView *)nextValidKeyView
```

Return Value
The closest view object in the key view loop that follows the receiver and accepts first responder status, or `nil` if there is none.

Discussion
This method ignores hidden views when it determines the next valid key view.

Availability
Available in OS X v10.0 and later.

See Also
- `nextKeyView` (page 129)
- `setNextKeyView:` (page 178)
- `previousKeyView` (page 137)

– previousValidKeyView (page 137)
selectNextKeyView: (NSWindow)
selectKeyViewFollowingView: (NSWindow)
selectPreviousKeyView: (NSWindow)
selectKeyViewPrecedingView: (NSWindow)
– setHidden: (page 171)

Related Sample Code
Dicey

Declared in
NSView.h

noteFocusRingMaskChanged

Invoked to notify the view that the focus ring mask requires updating.

```
– (void)noteFocusRingMaskChanged
```

Discussion
It is important to note that it is only necessary for developers to invoke this method when some internal state change of their application, that the Application Kit can't determine, affects the shape of the focus ring mask.

It is assumed that if the view is marked as needing display, or is resized, its focus ring shape is likely to have changed, and there is no need for clients to explicitly send this message in such cases, they are handled automatically.

If, however, a view is showing a focus ring around some part of its content (an NSImage, perhaps), and that content changes, the client must provide notification by invoking this method so that focusRingMaskBounds (page 102) and drawFocusRingMask (page 94) will be invoked to redraw the focus ring.

Availability
Available in OS X v10.7 and later.

See Also
– focusRingMaskBounds (page 102)
– drawFocusRingMask (page 94)

Declared in
NSView.h

opaqueAncestor

Returns the receiver's closest opaque ancestor (including the receiver itself).

```
- (NSView *)opaqueAncestor
```

Availability
Available in OS X v10.0 and later.

See Also
- `isOpaque` (page 116)
- `displayRectIgnoringOpacity:` (page 89)
- `displayIfNeededIgnoringOpacity` (page 87)
- `displayIfNeededInRectIgnoringOpacity:` (page 88)

Declared in
`NSView.h`

pageFooter

Returns a default footer string that includes the current page number and page count.

```
- (NSAttributedString *)pageFooter
```

Discussion
A printable view class can override this method to substitute its own content in place of the default value. You should not need to call this method directly. The printing system calls it once per page during printing.

Footers are generated only if the user defaults contain the key `NSPrintHeaderAndFooter` with the value `YES`.

Availability
Available in OS X v10.4 and later.

See Also
- `pageHeader` (page 132)

Declared in
`NSView.h`

pageHeader

Returns a default header string that includes the print job title and date.

```
- (NSAttributedString *)pageHeader
```

Discussion
Typically, the print job title is the same as the window title. A printable view class can override this method to substitute its own content in place of the default value. You should not need to call this method directly. The printing system calls it once per page during printing.

Headers are generated only if the user defaults contain the key `NSPrintHeaderAndFooter` with the value `YES`.

Availability
Available in OS X v10.4 and later.

See Also
– `pageFooter` (page 132)

Declared in
`NSView.h`

performKeyEquivalent:

Implemented by subclasses to respond to key equivalents (also known as keyboard shortcuts).

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

Parameters
`theEvent`
The key-down event object representing a key equivalent.

Return Value
`YES` if `theEvent` is a key equivalent that the receiver handled, `NO` if it is not a key equivalent that it should handle.

Discussion
If the receiver's key equivalent is the same as the characters of the key-down event `theEvent`, as returned by `charactersIgnoringModifiers`, the receiver should take the appropriate action and return `YES`. Otherwise, it should return the result of invoking `super`'s implementation. The default implementation of this method simply passes the message down the view hierarchy (from superviews to subviews) and returns `NO` if none of the receiver's subviews responds `YES`.

Availability
Available in OS X v10.0 and later.

See Also
– performMnemonic: (page 226)
keyDown: (NSWindow)

Declared in
NSView.h

postsBoundsChangedNotifications

Returns YES if the receiver posts notifications to the default notification center whenever its bounds rectangle changes; returns NO otherwise.

```
– (BOOL)postsBoundsChangedNotifications
```

Discussion
See setPostsBoundsChangedNotifications: (page 179) for a list of methods that result in notifications.

Availability
Available in OS X v10.0 and later.

Declared in
NSView.h

postsFrameChangedNotifications

Returns YES if the receiver posts notifications to the default notification center whenever its frame rectangle changes; returns NO otherwise.

```
– (BOOL)postsFrameChangedNotifications
```

Discussion
See setPostsBoundsChangedNotifications: (page 179) for a list of methods that result in notifications.

Availability
Available in OS X v10.0 and later.

Declared in
NSView.h

prepareContentInRect:

Prepares the overdraw region for drawing.

```
– (void)prepareContentInRect:(NSRect)rect
```

Parameters

`rect`

The current overdraw region, specified in the view's coordinate system. This rectangle includes the view's visible rectangle plus any space surrounding the visible rectangle that represents the overdraw region.

Discussion

During responsive scrolling, AppKit calls this method before asking your view to draw any content in the overdraw region. You can override this method in your own views and use it to prepare the content that is about to be drawn. For example, if your app defers the creation of subviews until they are scrolled into view, you would use this method to create them and add them to your view hierarchy.

Your implementation of this method must call `super` at some point. When calling `super`, you can extend the overdraw rectangle by passing a different rectangle for the `rect` parameter. For example, if you add a subview whose frame falls outside the current rectangle, you can grow the rectangle to include the entire frame of the subview.

AppKit may call this method multiple times to build up the current overdraw region slowly. Each time it calls the method, it extends the overdraw rectangle passed in the `rect` parameter. If you pass the same rectangle to `super` twice in succession, AppKit stops generating additional overdraw content. You can use this behavior to avoid generating more overdraw content than makes sense for your app. If the user scrolls the content, AppKit resets the current overdraw region and starts asking your app for content again. You can also reset the current overdraw region by assigning a value to the `preparedContentRect` (page 38) property.

Availability

Available in OS X v10.9 and later.

Declared in

`NSView.h`

prepareForReuse

Restores the view to an initial state so that it can be reused.

```
– (void)prepareForReuse
```

Discussion

The default implementation of this method sets the window's alpha to `1.0` and its hidden state to `NO`. Subclasses can override this method and use it to return the view to its initial state. Subclasses should call `super` at some point in their implementation.

This method offers a way to reset a view to some initial state so that it can be reused. For example, the `NSTableView` class uses it to prepare views for reuse and thereby avoid the expense of creating new views as they scroll into view. If you implement a view-reuse system in your own code, you can call this method from your own code prior to reusing them.

Availability

Available in OS X v10.7 and later.

Declared in

`NSView.h`

preservesContentDuringLiveResize

Returns `YES` if the view supports the optimization of live-resize operations by preserving content that has not moved; otherwise, returns `NO`.

```
- (BOOL)preservesContentDuringLiveResize
```

Discussion

The default is `NO`. If your view supports the content preservation feature, you should override this method and have your implementation return `YES`.

Content preservation lets your view decide what to redraw during a live resize operation. If your view supports this feature, you should also provide a custom implementation of `setFrameSize:` (page 170) that invalidates the portions of your view that actually need to be redrawn.

For information on how to implement this feature in your views, see *Cocoa Performance Guidelines*.

Availability

Available in OS X v10.4 and later.

See Also

– `setFrameSize:` (page 170)

Declared in

`NSView.h`

previousKeyView

Returns the view object preceding the receiver in the key view loop.

```
- (NSView *)previousKeyView
```

Return Value
The view object preceding the receiver in the key view loop, or `nil` if there is none.

Discussion
This view should, if possible, be made first responder when the user navigates backward from the receiver using keyboard interface control.

Availability
Available in OS X v10.0 and later.

See Also
- `previousValidKeyView` (page 137)
- `nextKeyView` (page 129)
- `nextValidKeyView` (page 130)
- `setNextKeyView:` (page 178)

`selectNextKeyView:` (`NSWindow`)
`selectKeyViewFollowingView:` (`NSWindow`)
`selectPreviousKeyView:` (`NSWindow`)
`selectKeyViewPrecedingView:` (`NSWindow`)

Declared in
`NSView.h`

previousValidKeyView

Returns the closest view object in the key view loop that precedes the receiver and accepts first responder status.

```
- (NSView *)previousValidKeyView
```

Return Value
The closest view object in the key view loop that precedes the receiver and accepts first responder status, or `nil` if there is none.

Discussion
This method ignores hidden views when it determines the previous valid key view.

Availability
Available in OS X v10.0 and later.

See Also

– previousKeyView (page 137)
– nextValidKeyView (page 130)
– nextKeyView (page 129)
– setNextKeyView: (page 178)
selectNextKeyView: (NSWindow)
selectKeyViewFollowingView: (NSWindow)
selectPreviousKeyView: (NSWindow)
selectKeyViewPrecedingView: (NSWindow)
– setHidden: (page 171)

Declared in

NSView.h

print:

This action method opens the Print panel, and if the user chooses an option other than canceling, prints the receiver and all its subviews to the device specified in the Print panel.

```
– (void)print:(id)sender
```

Parameters

sender

The object that sent the message.

Availability

Available in OS X v10.0 and later.

See Also

– dataWithEPSInsideRect: (page 84)
– writeEPSInsideRect:toPasteboard: (page 213)

Related Sample Code

BlastApp

Declared in

NSView.h

printJobTitle

Returns the receiver's print job title.

```
- (NSString *)printJobTitle
```

Discussion
The default implementation first tries the window's `NSDocument` display name (`displayName`), then the window's title.

Availability
Available in OS X v10.0 and later.

Related Sample Code
ImageApp
TextEdit

Declared in
`NSView.h`

rectForPage:

Implemented by subclasses to determine the portion of the receiver to be printed for the page number page.

```
- (NSRect)rectForPage:(NSInteger)pageNumber
```

Parameters
`pageNumber`
An integer indicating a page number. Page numbers are one-based—that is pages run from one to *N*.

Return Value
A rectangle defining the region of the receiver to be printed for `pageNumber`. This method returns `NSZeroRect` if `pageNumber` is outside the receiver's bounds.

Discussion
If the receiver responded `YES` to an earlier `knowsPageRange:` (page 118) message, this method is invoked for each page it specified in the out parameters of that message. The receiver is later made to display this rectangle in order to generate the image for this page.

If an `NSView` object responds `NO` to `knowsPageRange:` (page 118), this method isn't invoked by the printing mechanism.

Availability
Available in OS X v10.0 and later.

See Also
– `adjustPageHeightNew:top:bottom:limit:` (page 48)
– `adjustPageWidthNew:left:right:limit:` (page 49)

Declared in
`NSView.h`

rectForSmartMagnificationAtPoint:inRect:

Returns the appropriate rectangle to use when magnifying around the specified point.

```
– (NSRect)rectForSmartMagnificationAtPoint:(NSPoint)location
inRect:(NSRect)visibleRect
```

Parameters

`location`

The location in your view's coordinate system around which magnification is centered.

`visibleRect`

The visible portion of the view. Use this value to help determine the specific content group you want to target for magnification.

Return Value

The rectangle to use for magnification, specified in the view's coordinate system. To get the default magnification behavior, return `NSZeroRect`.

Discussion

AppKit calls this method when magnifying content in a scroll view. If you do not override this method, or if you return `NSZeroRect`, the scroll view magnifies the view's content around the specified point. If you override this method and return a custom rectangle, the scroll view adjusts the magnification behavior to accommodate the rectangle you provide.

Use this method to provide AppKit with rectangles for your view's custom content. If your view's content can be divided into logical groups of content, use the provided `location` and `visibleRect` parameters to determine which group is being targeted and then return the rectangle that fully encloses that group. For example, a view with multiple columns of content could return the rectangle for the targeted column. The returned rectangle should always fully enclose the content, regardless of whether that rectangle is larger than the visible rectangle.

Availability

Available in OS X v10.8 and later.

Declared in

`NSView.h`

rectPreservedDuringLiveResize

Returns the rectangle identifying the portion of your view that did not change during a live resize operation.

```
- (NSRect)rectPreservedDuringLiveResize
```

Discussion

The returned rectangle is in the coordinate system of your view and reflects the space your view previously occupied. This rectangle may be smaller or the same size as your view's current bounds, depending on whether the view grew or shrunk.

If your view does not support content preservation during live resizing, the returned rectangle will be empty. To support content preservation, override `preservesContentDuringLiveResize` (page 136) in your view and have your implementation return `YES`.

> **Note:** The window containing your view must also support content preservation. To enable support for this feature in your window, use the `setPreservesContentDuringLiveResize:` method of `NSWindow`.

Availability

Available in OS X v10.4 and later.

See Also

- `getRectsExposedDuringLiveResize:count:` (page 106)
- `preservesContentDuringLiveResize` (page 136)

Declared in

`NSView.h`

reflectScrolledClipView:

Notifies a clip view's superview that either the clip view's bounds rectangle or the document view's frame rectangle has changed, and that any indicators of the scroll position need to be adjusted.

```
- (void)reflectScrolledClipView:(NSClipView *)aClipView
```

Parameters

`aClipView`

The `NSClipView` object whose superview is to be notified.

Discussion

`NSScrollView` implements this method to update its `NSScroller` objects.

Availability
Available in OS X v10.0 and later.

Declared in
`NSClipView.h`

registeredDraggedTypes

Returns the array of pasteboard drag types that the view can accept.

```
- (NSArray *)registeredDraggedTypes
```

Discussion
This method returns the types registered by calling `registerForDraggedTypes:` (page 142). Each element of the array is a uniform type identifier. The returned elements are in no particular order, but the array is guaranteed not to contain duplicate entries.

Availability
Available in OS X v10.4 and later.

Declared in
`NSView.h`

registerForDraggedTypes:

Registers the pasteboard types that the receiver will accept as the destination of an image-dragging session.

```
- (void)registerForDraggedTypes:(NSArray *)newTypes
```

Parameters
`newTypes`

An array of uniform type identifiers. See Types for Standard Data (OS X v10.6 and later) for descriptions of the pasteboard type identifiers.

Discussion
Registering an `NSView` object for dragged types automatically makes it a candidate destination object for a dragging session. As such, it must properly implement some or all of the `NSDraggingDestination` protocol methods. As a convenience, `NSView` provides default implementations of these methods. See the `NSDraggingDestination` protocol specification for details.

Availability
Available in OS X v10.0 and later.

See Also

– `registeredDraggedTypes` (page 142)
– `unregisterDraggedTypes` (page 197)

Related Sample Code

CocoaDragAndDrop

CompositeLab

Declared in

`NSView.h`

releaseGState

Frees the receiver's graphics state object, if it has one.

```
– (void)releaseGState
```

Availability

Available in OS X v10.0 and later.

See Also

– `allocateGState` (page 52)

Declared in

`NSView.h`

removeAllToolTips

Removes all tool tips assigned to the receiver.

```
– (void)removeAllToolTips
```

Discussion

This method operates on tool tips created using either `addToolTipRect:owner:userData:` (page 46) or `setToolTip:` (page 182).

Availability

Available in OS X v10.0 and later.

Declared in

`NSView.h`

removeConstraint:

Removes the specified constraint from the view.

```
- (void)removeConstraint:(NSLayoutConstraint *)constraint
```

Parameters

`constraint`

The constraint to remove. Removing a constraint not held by the view has no effect.

Availability

Available in OS X v10.7 and later.

Declared in

`NSLayoutConstraint.h`

removeConstraints:

Removes the specified constraints from the view.

```
- (void)removeConstraints:(NSArray *)constraints
```

Parameters

`constraints`

The constraints to remove.

Availability

Available in OS X v10.7 and later.

Declared in

`NSLayoutConstraint.h`

removeCursorRect:cursor:

Completely removes a cursor rectangle from the receiver.

```
- (void)removeCursorRect:(NSRect)aRect cursor:(NSCursor *)aCursor
```

Parameters

`aRect`

A rectangle defining a region of the receiver. Must match a value previously specified using `addCursorRect:cursor:` (page 43).

`aCursor`

An object representing a cursor. Must match a value previously specified using `addCursorRect:cursor:` (page 43).

Discussion

You should rarely need to use this method. `resetCursorRects` (page 149), which is invoked any time cursor rectangles need to be rebuilt, should establish only the cursor rectangles needed. If you implement `resetCursorRects` (page 149) in this way, you can then simply modify the state that `resetCursorRects` (page 149) uses to build its cursor rectangles and then invoke the `NSWindow` method `invalidateCursorRectsForView:`.

Availability

Available in OS X v10.0 and later.

See Also

– `discardCursorRects` (page 86)

Declared in

`NSView.h`

removeFromSuperview

Unlinks the receiver from its superview and its window, removes it from the responder chain, and invalidates its cursor rectangles.

```
– (void)removeFromSuperview
```

Discussion

The receiver is also released; if you plan to reuse it, be sure to retain it before sending this message and to release it as appropriate when adding it as a subview of another `NSView`.

Calling this method removes any constraints that refer to the view you are removing, or that refer to any view in the subtree of the view you are removing.

Never invoke this method during display.

Availability

Available in OS X v10.0 and later.

See Also

– `addSubview:` (page 44)
– `addSubview:positioned:relativeTo:` (page 45)
– `removeFromSuperviewWithoutNeedingDisplay` (page 146)

Related Sample Code
AnimatingViews
CocoaSlides
FancyAbout
TextEdit
TreeView

Declared in
`NSView.h`

removeFromSuperviewWithoutNeedingDisplay

Unlinks the receiver from its superview and its window and removes it from the responder chain, but does not invalidate its cursor rectangles to cause redrawing.

```
- (void)removeFromSuperviewWithoutNeedingDisplay
```

Discussion
The receiver is also released; if you plan to reuse it, be sure to retain it before sending this message and to release it as appropriate when adding it as a subview of another view.

Unlike its counterpart, `removeFromSuperview` (page 145), this method can be safely invoked during display.

Availability
Available in OS X v10.0 and later.

See Also
– `addSubview:` (page 44)
– `addSubview:positioned:relativeTo:` (page 45)

Declared in
`NSView.h`

removeToolTip:

Removes the tool tip identified by specified tag.

```
- (void)removeToolTip:(NSToolTipTag)tag
```

Parameters

`tag`

An integer tag that is the value returned by a previous `addToolTipRect:owner:userData:` (page 46) message.

Availability

Available in OS X v10.0 and later.

Declared in

`NSView.h`

removeTrackingArea:

Removes a given tracking area from the receiver.

```
- (void)removeTrackingArea:(NSTrackingArea *)trackingArea
```

Parameters

`trackingArea`

The tracking area to remove from the receiver.

Availability

Available in OS X v10.5 and later.

Related Sample Code

CustomMenus

TrackIt

Declared in

`NSView.h`

removeTrackingRect:

Removes the tracking rectangle identified by a tag.

```
- (void)removeTrackingRect:(NSTrackingRectTag)aTag
```

Parameters

`aTag`

An integer value identifying a tracking rectangle. It was returned by a previously sent `addTrackingRect:owner:userData:assumeInside:` (page 47) message.

Availability
Available in OS X v10.0 and later.

See Also
– `addTrackingRect:owner:userData:assumeInside:` (page 47)
– `removeTrackingArea:` (page 147)

Related Sample Code
ImageMap

Declared in
`NSView.h`

renewGState

Invalidates the receiver's graphics state object, if it has one.

```
- (void)renewGState
```

Discussion
The receiver's graphics state object will be regenerated using `setUpGState` (page 183) the next time the receiver is focused for drawing

Availability
Available in OS X v10.0 and later.

See Also
– `lockFocus` (page 122)

Related Sample Code
GLEssentials
GLSLShowpiece
QTCoreVideo301

Declared in
`NSView.h`

replaceSubview:with:

Replaces one of the receiver's subviews with another view.

```
- (void)replaceSubview:(NSView *)oldView with:(NSView *)newView
```

Parameters

`oldView`

The view to be replaced by `newView`. May not be `nil`.

`newView`

The view to replace `oldView`. May not be `nil`.

Discussion

This method does nothing if `oldView` is not a subview of the receiver.

Neither `oldView` nor `newView` may be `nil`, and the behavior is undefined if either of these parameters is `nil`.

This method causes `oldView` to be released; if you plan to reuse it, be sure to retain it before sending this message and to release it as appropriate when adding it as a subview of another NSView.

Availability

Available in OS X v10.0 and later.

See Also

– `addSubview:` (page 44)
– `addSubview:positioned:relativeTo:` (page 45)

Related Sample Code

BasicCocoaAnimations
CocoaSlides
ImageTransition
MatrixMixerTest

Declared in

`NSView.h`

resetCursorRects

Overridden by subclasses to define their default cursor rectangles.

```
– (void)resetCursorRects
```

Discussion

A subclass's implementation must invoke `addCursorRect:cursor:` (page 43) for each cursor rectangle it wants to establish. The default implementation does nothing.

Application code should never invoke this method directly; it's invoked automatically as described in ""Responding to User Events and Actions" in *View Programming Guide*." Use the `invalidateCursorRectsForView:` method instead to explicitly rebuild cursor rectangles.

Availability
Available in OS X v10.0 and later.

See Also
– `visibleRect` (page 207)

Declared in
`NSView.h`

resizeSubviewsWithOldSize:

Informs the receiver's subviews that the receiver's bounds rectangle size has changed.

```
– (void)resizeSubviewsWithOldSize:(NSSize)oldBoundsSize
```

Parameters
`oldBoundsSize`

The previous size of the receiver's bounds rectangle.

Discussion
If the receiver is configured to autoresize its subviews, this method is automatically invoked by any method that changes the receiver's frame size.

The default implementation sends `resizeWithOldSuperviewSize:` (page 150) to the receiver's subviews with `oldBoundsSize` as the argument. You shouldn't invoke this method directly, but you can override it to define a specific retiling behavior.

Availability
Available in OS X v10.0 and later.

See Also
– `setAutoresizesSubviews:` (page 157)

Declared in
`NSView.h`

resizeWithOldSuperviewSize:

Informs the receiver that the bounds size of its superview has changed.

```
– (void)resizeWithOldSuperviewSize:(NSSize)oldBoundsSize
```

Parameters
`oldBoundsSize`
The previous size of the superview's bounds rectangle.

Discussion
This method is normally invoked automatically from `resizeSubviewsWithOldSize:` (page 150).

The default implementation resizes the receiver according to the autoresizing options listed under the `setAutoresizingMask:` (page 157) method description. You shouldn't invoke this method directly, but you can override it to define a specific resizing behavior.

If you override this method and call `super` as part of your implementation, you should be sure to call `super` before making changes to the receiving view's frame yourself.

Availability
Available in OS X v10.0 and later.

Declared in
`NSView.h`

rightMouseDown:

Informs the receiver that the user has pressed the right mouse button.

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

Parameters
`theEvent`
An object encapsulating information about the mouse-down event.

Discussion
The default implementation calls `menuForEvent:` (page 125) and, if non `nil`, presents the contextual menu. In OS X v10.7 and later, if the event is not handled, this method passes it up the responder chain.

See Also
`rightMouseDown:` (NSResponder)

rotateByAngle:

Rotates the receiver's bounds rectangle by a specified degree value around the origin of the coordinate system, (0.0, 0.0).

```
- (void)rotateByAngle:(CGFloat)angle
```

Parameters

`angle`

A `float` value specifying the angle of rotation, in degrees.

Discussion

See the `setBoundsRotation:` (page 161) method description for more information. This method neither redisplays the receiver nor marks it as needing display. You must do this yourself with `display` (page 86) or `setNeedsDisplay:` (page 175).

This method posts an `NSViewBoundsDidChangeNotification` (page 222) to the default notification center if the receiver is configured to do so.

Availability

Available in OS X v10.0 and later.

See Also

– `setFrameRotation:` (page 169)
– `setPostsBoundsChangedNotifications:` (page 179)

Declared in

`NSView.h`

scaleUnitSquareToSize:

Scales the receiver's coordinate system so that the unit square scales to the specified dimensions.

```
- (void)scaleUnitSquareToSize:(NSSize)newUnitSize
```

Parameters

`newUnitSize`

An `NSSize` structure specifying the new unit size.

Discussion

For example, a `newUnitSize` of (0.5, 1.0) causes the receiver's horizontal coordinates to be halved, in turn doubling the width of its bounds rectangle. Note that scaling is performed from the origin of the coordinate system, (0.0, 0.0), not the origin of the bounds rectangle; as a result, both the origin and size of the bounds rectangle are changed. The frame rectangle remains unchanged.

This method neither redisplays the receiver nor marks it as needing display. You must do this yourself with `display` (page 86) or `setNeedsDisplay:` (page 175).

This method posts an `NSViewBoundsDidChangeNotification` (page 222) to the default notification center if the receiver is configured to do so.

Availability
Available in OS X v10.0 and later.

See Also
– `setBoundsSize:` (page 162)
– `setPostsBoundsChangedNotifications:` (page 179)

Declared in
`NSView.h`

scrollClipView:toPoint:

Notifies the superview of a clip view that the clip view needs to reset the origin of its bounds rectangle.

```
- (void)scrollClipView:(NSClipView *)aClipView toPoint:(NSPoint)newOrigin
```

Parameters
`aClipView`
The `NSClipView` object whose superview is to be notified.

`newOrigin`
A point that specifies the new origin of the clip view's bounds rectangle.

Discussion
The superview of `aClipView` should then send a `scrollToPoint:` message to `aClipView` with `newOrigin` as the argument. This mechanism is provided so the `NSClipView` object's superview can coordinate scrolling of multiple tiled clip views.

Availability
Available in OS X v10.0 and later.

See Also
`scrollToPoint:` (NSClipView)

Declared in
`NSClipView.h`

scrollPoint:

Scrolls the receiver's closest ancestor `NSClipView` object so a point in the receiver lies at the origin of the clip view's bounds rectangle.

```
- (void)scrollPoint:(NSPoint)aPoint
```

Parameters

`aPoint`

The point in the receiver to scroll to.

Availability

Available in OS X v10.0 and later.

See Also

`- autoscroll:` (page 55)
`scrollToPoint:` (`NSClipView`)
`- isDescendantOf:` (page 113)

Declared in

`NSView.h`

scrollRect:by:

Copies the visible portion of the receiver's rendered image within a region and lays that portion down again at a specified offset .

```
- (void)scrollRect:(NSRect)aRect by:(NSSize)offset
```

Parameters

`aRect`

A rectangle defining a region of the receiver.

`offset`

A `NSSize` structure that specifies an offset from from `aRect`'s origin.

Discussion

This method is useful during scrolling or translation of the coordinate system to efficiently move as much of the receiver's rendered image as possible without requiring it to be redrawn, following these steps:

1. Invoke `scrollRect:by:` (page 154) to copy the rendered image.
2. Move the view object's origin or scroll it within its superview.

3. Calculate the newly exposed rectangles and invoke either `setNeedsDisplay:` (page 175) or `setNeedsDisplayInRect:` (page 176) to draw them.

You should rarely need to use this method, however. The `scrollPoint:` (page 154), `scrollRectToVisible:` (page 155), and `autoscroll:` (page 55) methods automatically perform optimized scrolling.

Availability
Available in OS X v10.0 and later.

See Also
– `setBoundsOrigin:` (page 160)
– `translateOriginToPoint:` (page 194)

Declared in
`NSView.h`

scrollRectToVisible:

Scrolls the receiver's closest ancestor `NSClipView` object the minimum distance needed so a specified region of the receiver becomes visible in the clip view.

```
– (BOOL)scrollRectToVisible:(NSRect)aRect
```

Parameters
`aRect`
The rectangle to be made visible in the clip view.

Discussion
`YES` if any scrolling is performed; otherwise returns `NO`.

Availability
Available in OS X v10.0 and later.

See Also
– `autoscroll:` (page 55)
`scrollToPoint:` (NSClipView)
– `isDescendantOf:` (page 113)

Related Sample Code
TextEdit
TreeView

Declared in
NSView.h

setAcceptsTouchEvents:

Sets whether the view should accept touch events.

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

Parameters
flag
YES if the view should accept touch events, otherwise NO.

Discussion
By default views do not accept touch events.

Availability
Available in OS X v10.6 and later.

See Also
– acceptsTouchEvents (page 41)

Declared in
NSView.h

setAlphaValue:

Sets the opacity of the receiver.

```
- (void)setAlphaValue:(CGFloat)viewAlpha
```

Parameters
viewAlpha
The desired opacity of the receiver. Possible values are between 0.0 (transparent) and 1.0 (opaque). The default is 1.0.

Special Considerations
Prior to OS X v10.6, views not managing a Core Animation layer did not support this method.

Availability
Available in OS X v10.5 and later.

Related Sample Code
TableViewPlayground

Declared in
`NSView.h`

setAutoresizesSubviews:

Determines whether the receiver automatically resizes its subviews when its frame size changes.

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

Parameters
`flag`

If `YES`, the receiver invokes `resizeSubviewsWithOldSize:` (page 150) whenever its frame size changes; if `NO`, it doesn't.

Discussion
View objects do autoresize their subviews by default.

Availability
Available in OS X v10.0 and later.

See Also
– `autoresizesSubviews` (page 54)

Related Sample Code
CustomMenus
TextSizingExample
TreeView

Declared in
`NSView.h`

setAutoresizingMask:

Determines how the receiver's `resizeWithOldSuperviewSize:` (page 150) method changes its frame rectangle.

```
- (void)setAutoresizingMask:(NSUInteger)mask
```

Parameters

`mask`

An integer bit mask. `mask` can be specified by combining using the C bitwise OR operator any of the options described in "Resizing masks" (page 215).

Discussion

Where more than one option along an axis is set, `resizeWithOldSuperviewSize:` (page 150) by default distributes the size difference as evenly as possible among the flexible portions. For example, if `NSViewWidthSizable` and `NSViewMaxXMargin` are set and the superview's width has increased by 10.0 units, the receiver's frame and right margin are each widened by 5.0 units.

Availability

Available in OS X v10.0 and later.

See Also

– `autoresizingMask` (page 54)
– `resizeSubviewsWithOldSize:` (page 150)
– `setAutoresizesSubviews:` (page 157)

Related Sample Code

CocoaSlides
TextEdit
TextLayoutDemo
TextSizingExample
Worm

Declared in

`NSView.h`

setBackgroundFilters:

An array of CoreImage filters that are applied to the receiver's background.

```
- (void)setBackgroundFilters:(NSArray *)filters
```

Parameters

`filters`

An array of CoreImage filters.

Discussion

This method sets the value of the `backgroundFilters` property of the receiver's layer.

Sending this message to a view that is not managing a Core Animation layer causes an exception.

Availability
Available in OS X v10.5 and later.

Related Sample Code
LayerBackedOpenGLView

Declared in
`NSView.h`

setBounds:

Sets the receiver's bounds rectangle.

```
- (void)setBounds:(NSRect)boundsRect
```

Parameters
`boundsRect`
A rectangle defining the new bounds of the receiver.

Discussion
The bounds rectangle determines the origin and scale of the receiver's coordinate system within its frame rectangle. This method neither redisplays the receiver nor marks it as needing display. You must do this yourself with `display` (page 86) or `setNeedsDisplay:` (page 175).

This method posts an `NSViewBoundsDidChangeNotification` (page 222) to the default notification center if the receiver is configured to do so.

After calling this method, `NSView` creates an internal transform (or appends these changes to an existing internal transform) to convert from frame coordinates to bounds coordinates in your view. As long as the width-to-height ratio of the two coordinate systems remains the same, your content appears normal. If the ratios differ, your content may appear skewed.

Availability
Available in OS X v10.0 and later.

See Also
- `bounds` (page 60)
- `setBoundsRotation:` (page 161)
- `setBoundsOrigin:` (page 160)
- `setBoundsSize:` (page 162)
- `setFrame:` (page 167)
- `setPostsBoundsChangedNotifications:` (page 179)

Related Sample Code
CIAnnotation

Declared in
`NSView.h`

setBoundsOrigin:

Sets the origin of the receiver's bounds rectangle to a specified point,

```
- (void)setBoundsOrigin:(NSPoint)newOrigin
```

Parameters
`newOrigin`
A point specifying the new bounds origin of the receiver.

Discussion
In setting the new bounds origin, this method effectively shifts the receiver's coordinate system so `newOrigin` lies at the origin of the receiver's frame rectangle. It neither redisplays the receiver nor marks it as needing display. You must do this yourself with `display` or `setNeedsDisplay:` (page 175).

This method posts an `NSViewBoundsDidChangeNotification` (page 222) to the default notification center if the receiver is configured to do so.

After calling this method, `NSView` creates an internal transform (or appends these changes to an existing internal transform) to convert from frame coordinates to bounds coordinates in your view. As long as the width-to-height ratio of the two coordinate systems remains the same, your content appears normal. If the ratios differ, your content may appear skewed.

Availability
Available in OS X v10.0 and later.

See Also
- `translateOriginToPoint:` (page 194)
- `bounds` (page 60)
- `setBoundsRotation:` (page 161)
- `setBounds:` (page 159)
- `setBoundsSize:` (page 162)
- `setPostsBoundsChangedNotifications:` (page 179)

Related Sample Code
Rulers

Declared in
`NSView.h`

setBoundsRotation:

Sets the rotation of the receiver's bounds rectangle to a specific degree value.

```
- (void)setBoundsRotation:(CGFloat)angle
```

Parameters
`angle`
A `float` value specifying the angle of rotation, in degrees.

Discussion
Positive values indicate counterclockwise rotation, negative clockwise. Rotation is performed around the coordinate system origin, (0.0, 0.0), which need not coincide with that of the frame rectangle or the bounds rectangle. This method neither redisplays the receiver nor marks it as needing display. You must do this yourself with `display` (page 86) or `setNeedsDisplay:` (page 175).

This method posts an `NSViewBoundsDidChangeNotification` (page 222) to the default notification center if the receiver is configured to do so.

Bounds rotation affects the orientation of the drawing within the view object's frame rectangle, but not the orientation of the frame rectangle itself. Also, for a rotated bounds rectangle to enclose all the visible areas of its view object—that is, to guarantee coverage over the frame rectangle—it must also contain some areas that aren't visible. This can cause unnecessary drawing to be requested, which may affect performance. It may be better in many cases to rotate the coordinate system in the `drawRect:` (page 96) method rather than use this method.

After calling this method, `NSView` creates an internal transform (or appends these changes to an existing internal transform) to convert from frame coordinates to bounds coordinates in your view. As long as the width-to-height ratio of the two coordinate systems remains the same, your content appears normal. If the ratios differ, your content may appear skewed.

Availability
Available in OS X v10.0 and later.

See Also
– `rotateByAngle:` (page 151)
– `boundsRotation` (page 61)
– `setFrameRotation:` (page 169)
– `setPostsBoundsChangedNotifications:` (page 179)

Related Sample Code
TrackIt

Declared in
`NSView.h`

setBoundsSize:

Sets the size of the receiver's bounds rectangle to specified dimensions, inversely scaling its coordinate system relative to its frame rectangle.

```
- (void)setBoundsSize:(NSSize)newSize
```

Parameters
`newSize`

An `NSSize` structure specifying the new width and height of the receiver's bounds rectangle.

Discussion
For example, a view object with a frame size of (100.0, 100.0) and a bounds size of (200.0, 100.0) draws half as wide along the x axis. This method neither redisplays the receiver nor marks it as needing display. You must do this yourself with `display` (page 86) or `setNeedsDisplay:` (page 175).

This method posts an `NSViewBoundsDidChangeNotification` (page 222) to the default notification center if the receiver is configured to do so.

After calling this method, `NSView` creates an internal transform (or appends these changes to an existing internal transform) to convert from frame coordinates to bounds coordinates in your view. As long as the width-to-height ratio of the two coordinate systems remains the same, your content appears normal. If the ratios differ, your content may appear skewed.

Availability
Available in OS X v10.0 and later.

See Also
- `bounds` (page 60)
- `setBoundsRotation:` (page 161)
- `setBounds:` (page 159)
- `setBoundsOrigin:` (page 160)
- `setPostsBoundsChangedNotifications:` (page 179)

Related Sample Code
BezierPathLab

Rulers

Sketch
Sketch+Accessibility

Declared in
NSView.h

setCanDrawConcurrently:

Sets whether the view's `drawRect:` method can be invoked on a background thread.

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

Parameters
`flag`

`YES` if `drawRect:` (page 96) can be invoked from a background thread, otherwise `NO`. The default is `NO` for most types of views..

Discussion
The view's window must also have its `allowsConcurrentViewDrawing` property set to `YES` (the default) for threading of view drawing to actually take place.

Availability
Available in OS X v10.6 and later.

See Also
`– setCanDrawConcurrently:` (page 163)
`allowsConcurrentViewDrawing` (NSWindow)

Declared in
NSView.h

setCanDrawSubviewsIntoLayer:

Sets whether the view incorporates content from its subviews into its own layer.

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

Parameters
`flag`

`YES` if the layer should incorporate content from its subviews or `NO` if it should not.

Discussion

Use this method to flatten the layer hierarchy for a layer-backed view and its subviews. Flattening a layer hierarchy reduces the number of layers (and may reduce the amount of memory) used by your view hierarchy. Reducing the number of layers can be more efficient in situations where there is significant overlap among the subviews or where the content of the view and subviews does not change significantly. For example, flattening a hierarchy reduces the amount of time spent compositing your views together. Do not flatten a view hierarchy if you plan to animate one or more subviews in that hierarchy.

When calling this method, the current view must have a layer object. When you specify `YES` for the flag parameter, subviews that have an implicit layer—that is, subviews for which you did not explicitly call the `setWantsLayer:` method—draw their content into the layer of the current view. If you do request a layer explicitly for a subview, that subview continues to draw its content into its own layer.

In a flattened layer hierarchy, the current view and its subviews draw their content explicitly using the `drawRect:` (page 96) method. They do not use the `updateLayer` method to update their layer contents, even if the `wantsUpdateLayer` method returns `YES`.

Availability

Available in OS X v10.9 and later.

Declared in

`NSView.h`

setCompositingFilter:

Sets a CoreImage filter that is used to composite the receiver's contents with the background.

```
- (void)setCompositingFilter:(CIFilter *)filter
```

Parameters

`filter`

A CoreImage filter.

Discussion

This method sets the value of the `compositingFilter` (page 65) property of the receiver's layer.

Sending this message to a view that is not managing a Core Animation layer causes an exception.

Availability

Available in OS X v10.5 and later.

Declared in

`NSView.h`

setContentCompressionResistancePriority:forOrientation:

Sets the priority with which a view resists being made smaller than its intrinsic size.

```
– (void)setContentCompressionResistancePriority:(NSLayoutPriority)priority
forOrientation:(NSLayoutConstraintOrientation)orientation
```

Parameters

`priority`

The new priority.

`orientation`

The orientation for which the compression resistance priority should be set.

Discussion

Custom views should set default values for both orientations on creation, based on their content, typically to `NSLayoutPriorityDefaultLow` or `NSLayoutPriorityDefaultHigh`. When creating user interfaces, the layout designer can modify these priorities for specific views when the overall layout design requires different tradeoffs than the natural priorities of the views being used in the interface.

Subclasses should not override this method.

Availability

Available in OS X v10.7 and later.

See Also

– `contentCompressionResistancePriorityForOrientation:` (page 67)

Declared in

`NSLayoutConstraint.h`

setContentFilters:

Sets the array of CoreImage filters that are applied to the contents of the receiver and its sublayers.

```
– (void)setContentFilters:(NSArray *)filters
```

Parameters

`filters`

An array of CoreImage filters.

Discussion

This method sets the value of the `filters` property of the receiver's layer.

Sending this message to a view that is not managing a Core Animation layer causes an exception.

Availability
Available in OS X v10.5 and later.

Declared in
`NSView.h`

setContentHuggingPriority:forOrientation:

Sets the priority with which a view resists being made larger than its intrinsic size.

```
- (void)setContentHuggingPriority:(NSLayoutPriority)priority
forOrientation:(NSLayoutConstraintOrientation)orientation
```

Parameters
`priority`

The new priority.

`orientation`

The orientation for which the content hugging priority should be set.

Discussion
Custom views should set default values for both orientations on creation, based on their content, typically to `NSLayoutPriorityDefaultLow` or `NSLayoutPriorityDefaultHigh`. When creating user interfaces, the layout designer can modify these priorities for specific views when the overall layout design requires different tradeoffs than the natural priorities of the views being used in the interface.

Subclasses should not override this method.

Availability
Available in OS X v10.7 and later.

See Also
– `contentHuggingPriorityForOrientation:` (page 68)

Declared in
`NSLayoutConstraint.h`

setFocusRingType:

Sets the type of focus ring to be drawn around the receiver.

```
- (void)setFocusRingType:(NSFocusRingType)focusRingType
```

Parameters

`focusRingType`

An `enum` constant identifying a type of focus ring. Possible values are listed in `NSFocusRingType`. You can specify `NSFocusRingTypeNone` to indicate you do not want your view to have a focus ring.

Discussion

This method only sets the desired focus ring type and does not cause the view to draw the actual focus ring. You are responsible for drawing the focus ring in your view's drawRect: (page 96) method whenever your view is made the first responder.

Availability

Available in OS X v10.3 and later.

See Also

– `focusRingType` (page 103)

Related Sample Code

GridMenu

Declared in

`NSView.h`

setFrame:

Sets the receiver's frame rectangle to the specified rectangle.

```
- (void)setFrame:(NSRect)frameRect
```

Parameters

`frameRect`

The new frame rectangle for the view.

Discussion

This method, in setting the frame rectangle, repositions and resizes the receiver within the coordinate system of its superview. It neither redisplays the receiver nor marks it as needing display. You must do this yourself with `display` (page 86) or `setNeedsDisplay:` (page 175).

This method posts an `NSViewFrameDidChangeNotification` (page 224) to the default notification center if the receiver is configured to do so.

If your view does not use a custom bounds rectangle, this method also sets your view bounds to match the size of the new frame. You specify a custom bounds rectangle by calling `setBounds:` (page 159), `setBoundsOrigin:` (page 160), `setBoundsRotation:` (page 161), or `setBoundsSize:` (page 162)explicitly. Once

set, NSView creates an internal transform to convert from frame coordinates to bounds coordinates. As long as the width-to-height ratio of the two coordinate systems remains the same, your content appears normal. If the ratios differ, your content may appear skewed.

Availability
Available in OS X v10.0 and later.

See Also
- `frame` (page 103)
- `setFrameRotation:` (page 169)
- `setFrameOrigin:` (page 169)
- `setFrameSize:` (page 170)
- `setPostsFrameChangedNotifications:` (page 180)

Related Sample Code
Cocoa Tips and Tricks
DatePicker
Rulers
Sketch
Sketch+Accessibility

Declared in
`NSView.h`

setFrameCenterRotation:

Rotates the frame of the receiver about the layer's position.

```
- (void)setFrameCenterRotation:(CGFloat)angle
```

Parameters
`angle`
The angle to rotate the frame around the center of the receiver.

Discussion
If the application has altered the layer's `anchorPoint` property, the behavior is undefined. Sending this message to a view that is not managing a Core Animation layer causes an exception.

Availability
Available in OS X v10.5 and later.

Declared in
`NSView.h`

setFrameOrigin:

Sets the origin of the receiver's frame rectangle to the specified point, effectively repositioning it within its superview.

```
- (void)setFrameOrigin:(NSPoint)newOrigin
```

Parameters

`newOrigin`

The point that is the new origin of the receiver's frame.

Discussion

This method neither redisplays the receiver nor marks it as needing display. You must do this yourself with `display` (page 86) or `setNeedsDisplay:` (page 175).

This method posts an `NSViewFrameDidChangeNotification` (page 224) to the default notification center if the receiver is configured to do so.

Availability

Available in OS X v10.0 and later.

See Also

- `frame` (page 103)
- `setFrameSize:` (page 170)
- `setFrame:` (page 167)
- `setFrameRotation:` (page 169)
- `setPostsFrameChangedNotifications:` (page 180)

Related Sample Code

CIAnnotation

TextSizingExample

TreeView

Declared in

`NSView.h`

setFrameRotation:

Sets the rotation of the receiver's frame rectangle to a specified degree value, rotating it within its superview without affecting its coordinate system.

```
- (void)setFrameRotation:(CGFloat)angle
```

Parameters

`angle`

A `float` value indicating the degree of rotation.

Discussion

Positive values indicate counterclockwise rotation, negative clockwise. Rotation is performed around the origin of the frame rectangle.

This method neither redisplays the receiver nor marks it as needing display. You must do this yourself with `display` (page 86) or `setNeedsDisplay:` (page 175).

This method posts an `NSViewFrameDidChangeNotification` (page 224) to the default notification center if the receiver is configured to do so.

Availability

Available in OS X v10.0 and later.

See Also

– `frameRotation` (page 105)
– `setBoundsRotation:` (page 161)

Declared in

`NSView.h`

setFrameSize:

Sets the size of the receiver's frame rectangle to the specified dimensions, resizing it within its superview without affecting its coordinate system.

```
- (void)setFrameSize:(NSSize)newSize
```

Parameters

`newSize`

An `NSSize` structure specifying the new height and width of the frame rectangle.

Discussion

This method neither redisplays the receiver nor marks it as needing display. You must do this yourself with `display` (page 86) or `setNeedsDisplay:` (page 175).

This method posts an `NSViewFrameDidChangeNotification` (page 224) to the default notification center if the receiver is configured to do so.

In OS X version 10.4 and later, you can override this method to support content preservation during live resizing. In your overridden implementation, include some conditional code to be executed only during a live resize operation. Your code must invalidate any portions of your view that need to be redrawn.

Availability
Available in OS X v10.0 and later.

See Also
- `frame` (page 103)
- `setFrameOrigin:` (page 169)
- `setFrame:` (page 167)
- `setFrameRotation:` (page 169)
- `setPostsFrameChangedNotifications:` (page 180)

Related Sample Code
BezierPathLab
CIAnnotation
GLSLShowpiece
ScreenSnapshot
TreeView

Declared in
`NSView.h`

setHidden:

Sets whether the view is hidden.

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

Parameters
`flag`
`YES` if the receiver is to be hidden, `NO` otherwise.

Discussion
A hidden view disappears from its window and does not receive input events. It remains in its superview's list of subviews, however, and participates in autoresizing as usual. The Application Kit also disables any cursor rectangle, tool-tip rectangle, or tracking rectangle associated with a hidden view. Hiding a view with subviews has the effect of hiding those subviews and any view descendants they might have. This effect is implicit and does not alter the hidden state of the receiver's descendants as reported by `isHidden` (page 114).

Hiding the view that is the window's current first responder causes the view's next valid key view (`nextValidKeyView` (page 130)) to become the new first responder. A hidden view remains in the `nextKeyView` (page 129) chain of views it was previously part of, but is ignored during keyboard navigation.

Availability
Available in OS X v10.3 and later.

See Also
– `isHidden` (page 114)
– `isHiddenOrHasHiddenAncestor` (page 115)

Related Sample Code
CoreWLANWirelessManager
DatePicker
IdentitySample
Quartz 2D Shadings
Quartz 2D Transformer

Declared in
`NSView.h`

setKeyboardFocusRingNeedsDisplayInRect:

Invalidates the area around the focus ring.

```
- (void)setKeyboardFocusRingNeedsDisplayInRect:(NSRect)rect
```

Parameters
`rect`
The rectangle of the control or cell defining the area around the focus ring. `rect` will be expanded to include the focus ring for invalidation.

Availability
Available in OS X v10.1 and later.

Declared in
`NSView.h`

setLayer:

Sets the Core Animation layer that the receiver uses for layer-backing to the specified layer.

```
- (void)setLayer:(CALayer *)newLayer
```

Parameters

`newLayer`

A Core Animation layer to use as the receiver's backing store.

Availability

Available in OS X v10.5 and later.

Related Sample Code

AVReaderWriter for OSX

CocoaSlides

CoreAnimationKioskStyleMenu

Quartz Composer AnimatedCompostionLayer

Quartz Composer LiveEdit

Declared in

`NSView.h`

setLayerContentsPlacement:

Sets the view's layer contents placement policy.

```
- (void)setLayerContentsPlacement:(NSViewLayerContentsPlacement)newPlacement
```

Parameters

`newPlacement`

The placement policy. See "NSViewLayerContentsPlacement" (page 219) for supported values.

Discussion

The content placement determines how the backing layer's existing cached content image will be mapped into the layer as the layer is resized. It is analogous to, and underpinned by, the CALayer class `contentsGravity` property. See `setLayerContentsRedrawPolicy:` (page 174) for more information.

Availability

Available in OS X v10.6 and later.

See Also

– `layerContentsPlacement` (page 119)

Declared in

`NSView.h`

setLayerContentsRedrawPolicy:

Sets the receiver layer contents redraw policy.

```
– (void)setLayerContentsRedrawPolicy:(NSViewLayerContentsRedrawPolicy)newPolicy
```

Parameters

`newPolicy`

The redraw policy. See "NSViewLayerContentsRedrawPolicy" (page 218) for supported values.

Discussion

If you know that redrawing at each animation frame is not necessary to produce correctly rendered results for a particular view, or are willing to accept an approximation of the view's intermediate appearance during potentially brief animations in exchange for an animation performance and smoothness benefit, you can change the view's `layerContentsRedrawPolicy` (page 119) to one of the modes that does not require constant redrawing.

When doing this, you must also specify the desired layer content placement for the view. The content placement determines how the backing layer's existing cached content image will be mapped into the layer as the layer is resized. It is analogous to, and underpinned by, the CALayer class `contentsGravity` property.

For a view that has no associated layer, or that has been assigned a developer-provided layer (a layer-hosting view) using the NSView `setLayer:` (page 172) method, the default contents redraw policy is is `NSViewLayerContentsRedrawNever` (page 218), with an accompanying `layerContentsPlacement` (page 119) of `NSViewLayerContentsPlacementScaleAxesIndependently` (page 220). This instructs AppKit that it is not allowed to replace the layer's content, and provides the same content placement as CALayer's default `contentsGravity` setting of `kCAGravityResize`. For a view that has acquired an AppKit-generated backing layer (a layer-backed view), AppKit sets the contents redraw policy to a default of `NSViewLayerContentsRedrawDuringViewResize` (page 219), forcing the view's content to be continually redrawn into the view's backing layer during animated resizing of the view, which produces strictly correct but not optimally performance results.

Availability

Available in OS X v10.6 and later.

Related Sample Code

TreeView

Declared in

`NSView.h`

setLayerUsesCoreImageFilters:

Sets whether the view's layer uses Core Image filters and should therefore be rendered in process.

```
- (void)setLayerUsesCoreImageFilters:(BOOL)usesFilters
```

Parameters

`usesFilters`

`YES` if the view's layer contains Core Image filters or `NO` if it does not contain any filters.

Discussion

If you apply Core Image filters directly to your view's layer object, you must call this method to let AppKit know of that fact. In OS X v10.9 and later, AppKit prefers to render layer trees out-of-process but cannot do so if any layers have Core Image filters attached to them. Specifying `YES` for the `usesFilters` parameter lets AppKit know that this view has Core Image filters and that AppKit should move rendering of the layer hierarchy back into your app's process. If you do not call this method with a value of `YES`, adding a filter to the view's layer triggers an exception.

If you assign Core Image filters to your view using the `setBackgroundFilters:` (page 158), `setCompositingFilter:` (page 164), or `setContentFilters:` (page 165) methods, you do not need to call this method explicitly. Those methods automatically let AppKit know that it needs to render the layer hierarchy in-process.

Availability

Available in OS X v10.9 and later.

See Also

– `layerUsesCoreImageFilters` (page 120)

Declared in

`NSView.h`

setNeedsDisplay:

Controls whether the receiver's entire bounds is marked as needing display.

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

Parameters

`flag`

If `YES`, marks the receiver's entire bounds as needing display; if `NO`, marks it as not needing display.

Discussion

Whenever the data or state used for drawing a view object changes, the view should be sent a setNeedsDisplay: message. `NSView` objects marked as needing display are automatically redisplayed on each pass through the application's event loop. (View objects that need to redisplay before the event loop comes around can of course immediately be sent the appropriate `display...` method.)

Availability

Available in OS X v10.0 and later.

See Also

– `setNeedsDisplayInRect:` (page 176)
– `needsDisplay` (page 127)

Related Sample Code

BezierPathLab
CircleView
OpenALExample
Quartz 2D Shadings
Quartz 2D Transformer

Declared in

`NSView.h`

setNeedsDisplayInRect:

Marks the region of the receiver within the specified rectangle as needing display, increasing the receiver's existing invalid region to include it.

```
– (void)setNeedsDisplayInRect:(NSRect)invalidRect
```

Parameters

`invalidRect`

The rectangular region of the receiver to mark as invalid; it should be specified in the coordinate system of the receiver.

Discussion

A later `displayIfNeeded...` method will then perform drawing only within the invalid region. View objects marked as needing display are automatically redisplayed on each pass through the application's event loop. (View objects that need to redisplay before the event loop comes around can of course immediately be sent the appropriate `display...` method.)

Availability
Available in OS X v10.0 and later.

See Also
– `setNeedsDisplay:` (page 175)
– `needsDisplay` (page 127)

Related Sample Code
CIAnnotation
DragItemAround
PDF Annotation Editor
Rulers
TextSizingExample

Declared in
`NSView.h`

setNeedsLayout:

Controls whether the view's subtree needs layout.

```
– (void)setNeedsLayout:(BOOL)flag
```

Parameters
`flag`
`YES` to indicate that the view needs layout, `NO` otherwise.

Discussion
You only ever need to invoke this method if your view implements custom layout not expressible in the constraint-based layout system by overriding the `layout` (page 120) method. The system invokes this method automatically for all views using constraints for layout.

Availability
Available in OS X v10.7 and later.

See Also
– `needsLayout` (page 127)

Declared in
`NSLayoutConstraint.h`

setNeedsUpdateConstraints:

Controls whether the view's constraints need updating.

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

Parameters

`flag`

`YES` to indicate the view's constraints need updating, `NO` otherwise.

Discussion

When a property of your custom view changes in a way that would impact constraints, you can call this method to indicate that the constraints need to be updated at some point in the future. The system will then call `updateConstraints` (page 197) as part of its normal layout pass. Updating constraints all at once just before they are needed ensures that you don't needlessly recalculate constraints when multiple changes are made to your view in between layout passes.

Availability

Available in OS X v10.7 and later.

See Also

– `needsUpdateConstraints` (page 129)

Declared in

`NSLayoutConstraint.h`

setNextKeyView:

Inserts a specified view object after the receiver in the key view loop of the receiver's window.

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

Parameters

`aView`

The `NSView` object to insert.

Availability

Available in OS X v10.0 and later.

See Also

– `nextKeyView` (page 129)
– `nextValidKeyView` (page 130)
– `previousKeyView` (page 137)

– previousValidKeyView (page 137)

Declared in
NSView.h

setPostsBoundsChangedNotifications:

Controls whether the receiver informs observers when its bounds rectangle changes.

```
– (void)setPostsBoundsChangedNotifications:(BOOL)flag
```

Parameters
flag

If YES, the receiver will post notifications to the default notification center whenever its bounds rectangle changes; if flag is NO it won't.

Discussion
Note that if flag is YES and bounds notifications are suppressed, when the bounds change notification is reenabled the view will immediately post a single such notification if its bounds changed during this time. This will happen even if there has been no net change in the view's bounds.

The following methods can result in notification posting:

setBounds: (page 159)

setBoundsOrigin: (page 160)

setBoundsRotation: (page 161)

setBoundsSize: (page 162)

translateOriginToPoint: (page 194)

scaleUnitSquareToSize: (page 152)

rotateByAngle: (page 151)

Availability
Available in OS X v10.0 and later.

See Also
– postsBoundsChangedNotifications (page 134)

Declared in
NSView.h

setPostsFrameChangedNotifications:

Controls whether the receiver informs observers when its frame rectangle changes.

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

Parameters

`flag`

If `YES`, the receiver will post notifications to the default notification center whenever its frame rectangle changes; if `flag` is `NO` it won't.

Discussion

Note that if `flag` is `YES` and frame notifications are suppressed, when the frame change notification is reenabled the view will immediately post a single such notification if its frame changed during this time. This will happen even if there has been no net change in the view's frame.

The following methods can result in notification posting:

`setFrame:` (page 167)

`setFrameOrigin:` (page 169)

`setFrameRotation:` (page 169)

`setFrameSize:` (page 170)

Availability

Available in OS X v10.0 and later.

See Also

– `postsFrameChangedNotifications` (page 134)

Declared in

`NSView.h`

setShadow:

Sets the shadow drawn by the receiver.

```
- (void)setShadow:(NSShadow *)shadow
```

Parameters

`shadow`

An instance of `NSShadow`.

Discussion

This method sets the `shadowColor`,`shadowOffset`, `shadowOpacity` and `shadowRadius` properties of the receiver's layer.

Sending this message to a view that is not managing a Core Animation layer causes an exception.

Availability

Available in OS X v10.5 and later.

Declared in

`NSView.h`

setSubviews:

Sets the receiver's subviews to the specified subviews.

```
– (void)setSubviews:(NSArray *)newSubviews
```

Parameters

`newSubviews`

An array of subviews. The `newSubviews` array can consist of existing subviews of the receiver or other views that have `nil` as their superview. If `newSubviews` is `nil`, or contains duplicated views, or if any of its members have a superview other than `nil` or the receiver, an invalid argument exception is thrown.

Discussion

Using this method you can: reorder the receiver's existing subviews, add or remove subviews en masse, replace all of the receiver's subviews with a new set of subviews, or remove all the receiver's subviews.

Given a valid array of views in `newSubviews`, `setSubviews:` (page 181) performs any required sorting of the subviews array, as well as sending any `addSubview:` (page 44) and `removeFromSuperview` (page 145) messages as necessary to leave the receiver with the requested new array of subviews. Any member of `newSubviews` that isn't already a subview of the receiver is added. Any member of the view's existing `subviews` array that isn't in `newSubviews` is removed. And any views that are in both `subviews` (page 191) and `newSubviews` are moved in the subviews array as needed, without being removed and re-added.

This method marks the affected view and window areas as needing display.

Availability

Available in OS X v10.5 and later.

Related Sample Code

CocoaSlides

Declared in
NSView.h

setToolTip:

Sets the tool tip text for the view to `string`.

```
- (void)setToolTip:(NSString *)string
```

Parameters
`string`
A string that contains the text to use for the tool tip. If `nil`, it cancels tool tip display for the view.

Availability
Available in OS X v10.0 and later.

See Also
– `toolTip` (page 193)

Related Sample Code
TextEdit

Declared in
NSView.h

setTranslatesAutoresizingMaskIntoConstraints:

Sets whether the view's autoresizing mask should be translated into constraints for the constraint-based layout system.

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

Parameters
`flag`
`YES` if the view's autoresizing mask should be translated into constraints for the constraint-based layout system, `NO` otherwise.

Discussion
Because the autoresizing mask naturally gives rise to constraints that fully specify a view's position, any view that you wish to apply more flexible constraints to must be set to ignore its autoresizing mask using this method. You should call this method yourself for programmatically created views. Views created using a tool that allows setting constraints should have this set already.

Availability
Available in OS X v10.7 and later.

See Also
– `translatesAutoresizingMaskIntoConstraints` (page 196)

Related Sample Code
InfoBarStackView

Declared in
`NSLayoutConstraint.h`

setUpGState

Overridden by subclasses to (re)initialize the receiver's graphics state object.

```
– (void)setUpGState
```

Discussion
This method is automatically invoked when the graphics state object created using `allocateGState` (page 52) needs to be initialized. The default implementation does nothing. Your subclass can override it to set the current font, line width, or any other graphics state parameter except coordinate transformations and the clipping path—these are established by the frame and bounds rectangles and by methods such as `scaleUnitSquareToSize:` (page 152) and `translateOriginToPoint:` (page 194). Note that `drawRect:` can further transform the coordinate system and clipping path for whatever temporary effects it needs.

Availability
Available in OS X v10.0 and later.

See Also
– `allocateGState` (page 52)
– `renewGState` (page 148)

Declared in
`NSView.h`

setUserInterfaceLayoutDirection:

Sets the preferred layout direction for the view's content.

```
– (void)setUserInterfaceLayoutDirection:(NSUserInterfaceLayoutDirection)value
```

Parameters

`value`

The preferred layout direction for the view's content.

Discussion

Use this method to change the preferred layout direction for content from the default value.

Availability

Available in OS X v10.8 and later.

See Also

`– userInterfaceLayoutDirection` (page 200)

Declared in

`NSView.h`

setWantsBestResolutionOpenGLSurface:

Sets whether the view supports a high resolution OpenGL backing surface.

```
– (void)setWantsBestResolutionOpenGLSurface:(BOOL)flag
```

Parameters

`flag`

Specify `YES` if the view supports a backing surface with a resolution greater than 1 pixel per point. Specify `NO` if you want the view's backing store to always have a resolution of 1 pixel per point.

Discussion

Specifying `YES` for the `flag` parameter tells AppKit that it has permission to allocate a higher resolution frame buffer for the view. AppKit uses the backing scale factor and the targeted display to determine whether a higher resolution buffer is appropriate, and it may change the surface resolution when the display mode changes or the view moves to a different display.

Availability

Available in OS X v10.7 and later.

Declared in

`NSOpenGLView.h`

setWantsLayer:

Specifies whether the receiver and its subviews use a Core Animation layer as a backing store.

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

Parameters

`flag`

`YES` if the receiver and its subviews should use a Core Animation layer as its backing store, otherwise `NO`.

Discussion

Calling this method with a value of `YES` turns the view into a layer-backed view—that is, the view uses a `CALayer` object to manage its rendered content. Creating a layer-backed view implicitly causes the entire view hierarchy under that view to become layer-backed. Thus, the view and all of its subviews (including subviews of subviews) become layer-backed.

In a layer-backed view, any drawing done by the view is cached to the underlying layer object. This cached content can then be manipulated in ways that are more performant than redrawing the view contents explicitly. AppKit automatically creates the underlying layer object (using the `makeBackingLayer` (page 124) method) and handles the caching of the view's content. If the `wantsUpdateLayer` (page 211) method returns `NO`, you should not interact with the underlying layer object directly. Instead, use the methods of this class to make any changes to the view and its layer. If `wantsUpdateLayer` returns `YES`, it is acceptable (and appropriate) to modify the layer in the view's `updateLayer` (page 198) method.

In addition to creating a layer-backed view, you can also use the `setLayer:` (page 172) method to create a layer-hosting view. In a layer-hosting view, you create the layer object yourself and are responsible for managing it. To create a layer-hosting view, you must call `setLayer:` and supply your layer object before you call the `setWantsLayer:` method; the order of these method calls is crucial.

In a layer-hosting view, do not rely on the view for drawing. Similarly, do not add subviews to a layer-hosting view. The root layer—that is, the layer you set using the `setLayer:` method—becomes the root layer of the layer tree. Any manipulations of that layer tree must be done using the Core Animation interfaces. You still use the view for handling mouse and keyboard events, but any resulting drawing must be handled by Core Animation.

For any layer-backed view, you can flatten the layer hierarchy by calling the `setCanDrawSubviewsIntoLayer:` (page 163) method. To exclude one or more subviews from the resulting flattened layer hierarchy, call the `setWantsLayer:` method on them to give them a layer explicitly.

Availability

Available in OS X v10.5 and later.

See Also

– `wantsLayer` (page 209)

Related Sample Code

AnimatedTableView

AVReaderWriter for OSX

avvideowall
CocoaSlides
Quartz Composer LiveEdit

Declared in
`NSView.h`

setWantsRestingTouches:

Sets whether the view wants to receive resting touch events.

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

Parameters
`flag`
`YES` if the view wants resting touches, otherwise `NO`. The default is `NO`.

Discussion
A resting touch occurs when a user rests their thumb on a device (for example, the glass trackpad of a MacBook).

By default, these touches are not delivered and are not included in the event's set of touches. Touches may transition in and out of resting at any time. Unless the view wants restingTouches, began / ended events are simulated as touches transition from resting to active and vice versa.

In general resting touches should be ignored.

Availability
Available in OS X v10.6 and later.

See Also
– `wantsRestingTouches` (page 210)

Declared in
`NSView.h`

shadow

Returns the shadow drawn by the receiver

```
- (NSShadow *)shadow
```

Return Value
An instance of `NSShadow` that is created using the `shadowColor`,`shadowOffset`, `shadowOpacity` and `shadowRadius` properties of the receiver's layer.

Discussion
Sending this message to a view that is not managing a Core Animation layer causes an exception.

Availability
Available in OS X v10.5 and later.

Related Sample Code
CocoaSlides

Declared in
`NSView.h`

shouldDelayWindowOrderingForEvent:

Overridden by subclasses to allow the user to drag images from the receiver without its window moving forward and possibly obscuring the destination and without activating the application.

```
– (BOOL)shouldDelayWindowOrderingForEvent:(NSEvent *)theEvent
```

Parameters
`theEvent`

An object representing an initial mouse-down event.

Return Value
If this method returns `YES`, the normal window-ordering and activation mechanism is delayed (not necessarily prevented) until the next mouse-up event. If it returns `NO`, then normal ordering and activation occur.

Discussion
Never invoke this method directly; it's invoked automatically for each mouse-down event directed at the NSView.

An `NSView` subclass that allows dragging should implement this method to return `YES` if `theEvent` is potentially the beginning of a dragging session or of some other context where window ordering isn't appropriate. This method is invoked before a `mouseDown:` message for `theEvent` is sent. The default implementation returns `NO`.

If, after delaying window ordering, the receiver actually initiates a dragging session or similar operation, it should also send a `preventWindowOrdering` message to `NSApp`, which completely prevents the window from ordering forward and the activation from becoming active. `preventWindowOrdering` is sent automatically by the `NSViewdragImage:...` and `dragFile:...` methods.

Availability
Available in OS X v10.0 and later.

See Also
– `registerForDraggedTypes:` (page 142)
– `unregisterDraggedTypes` (page 197)
– `dragImage:at:offset:event:pasteboard:source:slideBack:` (page 91)
– `beginDraggingSessionWithItems:event:source:` (page 58)

Declared in
`NSView.h`

shouldDrawColor

Returns `NO` if the receiver is being drawn in an `NSWindow` object (as opposed, for example, to being printed) and the window object can't store color; otherwise returns `YES`.

```
– (BOOL)shouldDrawColor
```

Discussion
A view object can base its drawing behavior on the return value of this method to improve its appearance in grayscale windows.

Availability
Available in OS X v10.0 and later.

See Also
– `drawRect:` (page 96)
`canStoreColor` (NSWindow)

Declared in
`NSView.h`

showDefinitionForAttributedString:atPoint:

Shows a window displaying the definition of the of the attributed string at the specified point.

```
- (void)showDefinitionForAttributedString:(NSAttributedString *)attrString
atPoint:(NSPoint)textBaselineOrigin
```

Parameters

`attrString`

The attributed string for which to show the definition. If the receiver is an instance of NSTextView, the `attrString` can be `nil`, in which case the text view will automatically supply values suitable for displaying definitions for the specified range within its text content.

`textBaselineOrigin`

Specifies the baseline origin of `attrString` in the receiver's view coordinate system.

Discussion

Shows a window that displays the definition (or other subject depending on available dictionaries) of the specified attributed string.

This method can be used for implementing the same functionality as the `NSTextView` "Look Up in Dictionary" contextual menu on a custom view.

Availability

Available in OS X v10.6 and later.

See Also

– `showDefinitionForAttributedString:range:options:baselineOriginProvider:` (page 189)

Declared in

`NSView.h`

showDefinitionForAttributedString:range:options:baselineOriginProvider:

Shows a window displaying the definition of the specified range of the attributed string.

```
- (void)showDefinitionForAttributedString:(NSAttributedString *)attrString
range:(NSRange)targetRange options:(NSDictionary *)options
baselineOriginProvider:(NSPoint (^)(NSRange adjustedRange))originProvider
```

Parameters

`attrString`

The attributed string for which to show the definition. If the receiver is an instance of NSTextView, the `attrString` can be `nil`, in which case the text view will automatically supply values suitable for displaying definitions for the specified range within its text content.

`range`

The range of the attributed string to define. You can pass a zero-length range and the appropriate range will be auto-detected around the range's offset. That's the recommended approach when there is no selection.

`options`

An optional dictionary that specifies how the definition is displayed. See "NSDefinition Presentation Constants" (page 221) for the key and it's possible values.

`originProvider`

The originProvider block object should return the baseline origin for the first character at the adjusted range.

If the receiver is an instance of NSTextView, the originProvider can be `NULL`, in which case the text view will automatically supply values suitable for displaying definitions for the specified range within its text content.

The block object takes a single argument:

adjustedRange

The adjusted range.

The block object returns an `NSPoint` to be used as the baseline origin of the first character, in the receiver view coordinate system.

Discussion

This method does not cause scrolling, so clients should perform any necessary scrolling before calling this method.

Availability

Available in OS X v10.6 and later.

See Also

`– showDefinitionForAttributedString:atPoint:` (page 188)

Declared in

`NSView.h`

sortSubviewsUsingFunction:context:

Orders the receiver's immediate subviews using the specified comparator function.

```
– (void)sortSubviewsUsingFunction:(NSComparisonResult (*)(id, id, void *))compare
context:(void *)context
```

Parameters

`compare`

A pointer to the comparator function. This function must take as arguments two subviews to be ordered and contextual data (supplied in `context` which may be arbitrary data used to help in the comparison. The comparator function should return `NSOrderedAscending` if the first subview should be ordered lower, `NSOrderedDescending` if the second subview should be ordered lower, and `NSOrderedSame` if their ordering isn't important.

`context`

Arbitrary data that might help the comparator function `compare` in its decisions.

Availability

Available in OS X v10.0 and later.

See Also

`sortedArrayUsingFunction:context:` (`NSArray`)

Declared in

`NSView.h`

subviews

Return the receiver's immediate subviews.

```
- (NSArray *)subviews
```

Return Value

Returns an array containing the receiver's subviews.

Discussion

The order of the subviews may be considered as being back-to-front, but this does not imply invalidation and drawing behavior. The order is based on the order of the receiver's subviews as specified in the nib file from which they were unarchived or the programmatic interface for modifying the receiver's subview list. This ordering is also the reverse of the order in which hit-testing is done.

> **Note:** The contents of this array may change at any time. If you intend to manipulate this array you should copy it rather than simply retain.

Availability

Available in OS X v10.0 and later.

See Also

– `superview` (page 192)
– `addSubview:` (page 44)
– `addSubview:positioned:relativeTo:` (page 45)
– `removeFromSuperview` (page 145)

Related Sample Code

AnimatingViews
ClipboardViewer
GridMenu
MenuItemView
TreeView

Declared in

`NSView.h`

superview

Returns the receiver's superview, or `nil` if it has none.

```
- (NSView *)superview
```

Discussion

When applying this method iteratively or recursively, be sure to compare the returned view object to the content view of the window to avoid proceeding out of the view hierarchy.

Availability

Available in OS X v10.0 and later.

See Also

– `window` (page 212)
– `subviews` (page 191)
– `removeFromSuperview` (page 145)

Related Sample Code

MatrixMixerTest
Rulers
TextEdit
TextSizingExample
TreeView

Declared in

`NSView.h`

tag

Returns the receiver's tag, an integer that you can use to identify view objects in your application.

```
- (NSInteger)tag
```

Discussion

The default implementation returns –1. Subclasses can override this method to provide individual tags, possibly adding storage and a `setTag:` method (which `NSView` doesn't define).

Availability

Available in OS X v10.0 and later.

See Also

`– viewWithTag:` (page 207)

Related Sample Code

GridMenu

PDF Annotation Editor

Quartz2DBasics

Sketch+Accessibility

Declared in

`NSView.h`

toolTip

Returns the text for the view's tool tip.

```
- (NSString *)toolTip
```

Return Value

The tool tip text or `nil` if the view doesn't currently display tool tip text

Availability

Available in OS X v10.0 and later.

See Also

`– setToolTip:` (page 182)

Declared in

`NSView.h`

trackingAreas

Returns an array of the receiver's tracking areas.

```
- (NSArray *)trackingAreas
```

Return Value

An array of the receiver's tracking areas (instances of `NSTrackingArea`). If the receiver has no tracking areas, returns an empty array.

Availability

Available in OS X v10.5 and later.

Related Sample Code

GridMenu

HoverTableDemo

MenuItemView

Declared in

`NSView.h`

translateOriginToPoint:

Translates the receiver's coordinate system so that its origin moves to a new location.

```
- (void)translateOriginToPoint:(NSPoint)newOrigin
```

Parameters

`newOrigin`

A point that specifies the new origin.

Discussion

In the process, the origin of the receiver's bounds rectangle is shifted by (–`newOrigin.x`, –`newOrigin.y`). This method neither redisplays the receiver nor marks it as needing display. You must do this yourself with `display` (page 86) or `setNeedsDisplay:` (page 175).

Note the difference between this method and setting the bounds origin. Translation effectively moves the image inside the bounds rectangle, while setting the bounds origin effectively moves the rectangle over the image. The two are in a sense inverse, although translation is cumulative, and setting the bounds origin is absolute.

This method posts an `NSViewBoundsDidChangeNotification` (page 222) to the default notification center if the receiver is configured to do so.

Availability
Available in OS X v10.0 and later.

See Also
- `setBoundsOrigin:` (page 160)
- `setBounds:` (page 159)
- `setPostsBoundsChangedNotifications:` (page 179)

Declared in
`NSView.h`

translateRectsNeedingDisplayInRect:by:

Translates the display rectangles by the specified delta.

```
- (void)translateRectsNeedingDisplayInRect:(NSRect)clipRect by:(NSSize)delta
```

Parameters
`clipRect`

A rectangle defining the region of the receiver, typically the receiver's bounds.

`delta`

A NSSize structure that specifies an offset from from aRect's origin.

Discussion
This method performs the shifting of dirty rectangles that an equivalent `scrollRect:by:` (page 154) operation would cause, without performing the actual scroll operation. It is only useful in very rare cases where a view implements its own low-level scrolling mechanics.

This method:

1. Collects the receiving view's dirty rectangles.
2. Clears all dirty rectangles in the intersection of `clipRect` and the view's bounds.
3. Shifts the retrieved rectangles by the `delta` offset.
4. Clips the result to the intersection of `clipRect` and the view's bounds
5. Marks the resultant rectangles as needing display.

The developer must ensure that `clipRect` and `delta` are pixel-aligned in order to guarantee correct drawing. See "Transforming View Coordinates To and From Base Space" in *View Programming Guide* for a description of how to pixel-align view coordinates.

Availability
Available in OS X v10.5 and later.

Declared in
`NSView.h`

translatesAutoresizingMaskIntoConstraints

Returns a Boolean value that indicates whether the view's autoresizing mask is translated into constraints for the constraint-based layout system.

```
- (BOOL)translatesAutoresizingMaskIntoConstraints
```

Return Value
`YES` if the view's autoresizing mask is translated into constraints for the constraint-based layout system, `NO` otherwise.

Discussion
If this is value is `YES`, the view's superview looks at the view's autoresizing mask, produces constraints that implement it, and adds those constraints to itself (the superview).

Availability
Available in OS X v10.7 and later.

See Also
`– setTranslatesAutoresizingMaskIntoConstraints:` (page 182)

Related Sample Code
InfoBarStackView

TextEdit

Declared in
`NSLayoutConstraint.h`

unlockFocus

Unlocks focus from the current view.

```
- (void)unlockFocus
```

Discussion

Call this method after a previous call to the `lockFocus` (page 122), `lockFocusIfCanDraw` (page 123), or `lockFocusIfCanDrawInContext:` (page 123) method of this view object. Doing so releases focus from the current view and returns it back to the previously focused view, if any. This method raises an `NSInvalidArgumentException` if the current view does not have the focus.

Availability

Available in OS X v10.0 and later.

See Also

– `allocateGState` (page 52)

Related Sample Code

BlastApp

Declared in

`NSView.h`

unregisterDraggedTypes

Unregisters the receiver as a possible destination in a dragging session.

```
- (void)unregisterDraggedTypes
```

Availability

Available in OS X v10.0 and later.

See Also

– `registerForDraggedTypes:` (page 142)
– `unregisterDraggedTypes` (page 197)
– `dragImage:at:offset:event:pasteboard:source:slideBack:` (page 91)
– `shouldDelayWindowOrderingForEvent:` (page 187)
– `beginDraggingSessionWithItems:event:source:` (page 58)

Declared in

`NSView.h`

updateConstraints

Update constraints for the view.

```
- (void)updateConstraints
```

Discussion
Custom views that set up constraints themselves should do so by overriding this method. When your custom view notes that a change has been made to the view that invalidates one of its constraints, it should immediately remove that constraint, and then call `setNeedsUpdateConstraints:` (page 178) to note that constraints need to be updated. Before layout is performed, your implementation of `updateConstraints` will be invoked, allowing you to verify that all necessary constraints for your content are in place at a time when your custom view's properties are not changing.

You may not invalidate any constraints as part of your constraint update phase. You also may not invoke a layout or drawing phase as part of constraint updating.

You must call `[super updateConstraints]` at the end of your implementation.

Availability
Available in OS X v10.7 and later.

Declared in
`NSLayoutConstraint.h`

updateConstraintsForSubtreeIfNeeded

Updates the constraints for the receiving view and its subviews.

```
- (void)updateConstraintsForSubtreeIfNeeded
```

Discussion
Whenever a new layout pass is triggered for a view, the system invokes this method to ensure that any constraints for the view and its subviews are updated with information from the current view hierarchy and its constraints. This method is called automatically by the system, but may be invoked manually if you need to examine the most up to date constraints.

Subclasses should not override this method.

Availability
Available in OS X v10.7 and later.

Declared in
`NSLayoutConstraint.h`

updateLayer

Updates the view's content by modifying its underlying layer.

```
- (void)updateLayer
```

Discussion
You use this method to optimize the rendering of your view in situations where you can represent your views contents entirely using a layer object. If your view's `wantsUpdateLayer` (page 211) method returns `YES`, the view calls this method instead of `drawRect:` (page 96) during the view update cycle. Custom views can override this method and use it to modify the properties of the underlying layer object. Modifying layer properties is a much more efficient way to update your view than is redrawing its content each time something changes.

When you want to update the contents of your layer, mark the view as dirty by calling its `setNeedsDisplay:` (page 175) method with a value of `YES`. Doing so adds the view to the list of views that need to be refreshed during the next update cycle. During that update cycle, this method is called if `wantsUpdateLayer` (page 211) still returns `YES`.

Your implementation of this method should not call `super`.

Availability
Available in OS X v10.8 and later.

See Also
– `wantsUpdateLayer` (page 211)

Declared in
`NSView.h`

updateTrackingAreas

Invoked automatically when the view's geometry changes such that its tracking areas need to be recalculated.

```
- (void)updateTrackingAreas
```

Discussion
You should override this method to remove out of date tracking areas and add recomputed tracking areas; your implementation should call `super`.

Availability
Available in OS X v10.5 and later.

Related Sample Code
CustomMenus
HoverTableDemo

Declared in
`NSView.h`

userInterfaceLayoutDirection

Returns the layout direction for content in the view.

```
– (NSUserInterfaceLayoutDirection)userInterfaceLayoutDirection
```

Discussion
Different languages support different directions for laying out content. While many languages support left-to-right layout, some support right-to-left layout. This method returns the preferred layout direction employed by the view. It is the responsibility of the view to respect this value and lay out its content appropriately.

In OS X v10.9 and later, if no layout direction is set explicitly, this method returns the value reported by the app's `userInterfaceLayoutDirection` method. In prior versions of OS X, it returns the value `NSUserInterfaceLayoutDirectionLeftToRight` by default. Certain AppKit subclasses, such as `NSOutlineView`, respect the value returned by this method and adjust their layout accordingly.

Availability
Available in OS X v10.8 and later.

See Also
– `setUserInterfaceLayoutDirection:` (page 183)

Declared in
`NSView.h`

viewDidChangeBackingProperties

Invoked when the receiver's backing store properties change.

```
– (void)viewDidChangeBackingProperties
```

Discussion
The receiver gets this message when the backing store scale or color space changes. Your app should provide an implementation if it needs to swap assets or make other adjustments when a view's backing store properties change.

Availability
Available in OS X v10.7 and later.

Declared in
`NSView.h`

viewDidEndLiveResize

Informs the receiver of the end of a live resize.

```
– (void)viewDidEndLiveResize
```

Discussion

In the simple case, a view is sent `viewWillStartLiveResize` (page 206) before the first resize operation on the containing window and `viewDidEndLiveResize` after the last resize operation. A view that is repeatedly added and removed from a window during live resize will receive only one `viewWillStartLiveResize` (on the first time it is added to the window) and one `viewDidEndLiveResize` (when the window has completed the live resize operation). This allows a superview such as `NSBrowser` object to add and remove its `NSMatrix` subviews during live resize without the `NSMatrix` receiving multiple calls to these methods.

A view might allocate data structures to cache-drawing information in `viewWillStartLiveResize` (page 206) and should clean up these data structures in `viewDidEndLiveResize`. In addition, a view that does optimized drawing during live resize might want to do full drawing after `viewDidEndLiveResize`, although a view should not assume that it has a drawing context in `viewDidEndLiveResize` (since it may have been removed from the window during live resize). A view that wants to redraw itself after live resize should call `[self setNeedsDisplay:YES]` in `viewDidEndLiveResize`.

A view subclass should call `super` from these methods.

Availability

Available in OS X v10.1 and later.

See Also

– `viewWillStartLiveResize` (page 206)
– `inLiveResize` (page 111)

Related Sample Code

ImageApp

Declared in

`NSView.h`

viewDidHide

Invoked when the receiver is hidden, either directly, or in response to an ancestor being hidden.

```
– (void)viewDidHide
```

Discussion

The receiver receives this message when its `isHiddenOrHasHiddenAncestor` (page 115) state goes from `NO` to `YES`. This will happen when the view or an ancestor is marked as hidden, or when the view or an ancestor is inserted into a new view hierarchy.

Availability

Available in OS X v10.5 and later.

Declared in

`NSView.h`

viewDidMoveToSuperview

Informs the receiver that its superview has changed (possibly to `nil`).

```
- (void)viewDidMoveToSuperview
```

Discussion

The default implementation does nothing; subclasses can override this method to perform whatever actions are necessary.

Availability

Available in OS X v10.0 and later.

See Also

- `viewDidMoveToWindow` (page 202)
- `viewWillMoveToSuperview:` (page 205)
- `viewWillMoveToWindow:` (page 205)

Declared in

`NSView.h`

viewDidMoveToWindow

Informs the receiver that it has been added to a new view hierarchy.

```
- (void)viewDidMoveToWindow
```

Discussion

The default implementation does nothing; subclasses can override this method to perform whatever actions are necessary.

If you call the `window` (page 212) method and it returns `nil`, that result signifies that the view was removed from its window and does not currently reside in any window.

Availability
Available in OS X v10.0 and later.

See Also
- `viewDidMoveToSuperview` (page 202)
- `viewWillMoveToSuperview:` (page 205)
- `viewWillMoveToWindow:` (page 205)

Related Sample Code
CustomMenus
Dicey
DispatchFractal
GridMenu

Declared in
`NSView.h`

viewDidUnhide

Invoked when the receiver is unhidden, either directly, or in response to an ancestor being unhidden

```
- (void)viewDidUnhide
```

Discussion
The receiver receives this message when its isHiddenOrHasHiddenAncestor state goes from `YES` to `NO`. This can happen when the view or an ancestor is marked as not hidden, or when the view or an ancestor is removed from its containing view hierarchy.

Availability
Available in OS X v10.5 and later.

Declared in
`NSView.h`

viewWillDraw

Informs the receiver that it will be required to draw content.

```
- (void)viewWillDraw
```

Discussion

In response to receiving one of the display... messages the receiver will recurse down the view hierarchy, sending this message to each of the views that may be involved in the display operation.

Subclasses can override this method to move or resize views, mark additional areas as requiring display, or other actions that can best be deferred until they are required for drawing. During the recursion, sending of `setNeedsDisplay:` (page 175) and `setNeedsDisplayInRect:` (page 176) messages to views in the hierarchy that's about to be drawn is valid and supported, and will affect the assessment of the total area to be rendered in that drawing pass.

A subclass's implementation of `viewWillDraw` can use the existing NSView `getRectsBeingDrawn:count:` (page 106) method to obtain a list of rectangles that bound the affected area, enabling it to restrict its efforts to that area.

The following is an example of a generic subclass implementation:

```
- (void)viewWillDraw {
    // Perform some operations before recursing for descendants.

    // Now recurse to handle all our descendants.
    // Overrides must call up to super like this.
    [super viewWillDraw];

    // Perform some operations that might depend on descendants
    //  already having had a chance to update.
}
```

Availability

Available in OS X v10.5 and later.

Related Sample Code

AnimatedTableView

CustomMenus

SidebarDemo

TreeView

Declared in

`NSView.h`

viewWillMoveToSuperview:

Informs the receiver that its superview is about to change to the specified superview (which may be `nil`).

```
- (void)viewWillMoveToSuperview:(NSView *)newSuperview
```

Parameters

`newSuperview`

A view object that will be the new superview of the receiver.

Discussion

Subclasses can override this method to perform whatever actions are necessary.

Availability

Available in OS X v10.0 and later.

See Also

- `viewDidMoveToSuperview` (page 202)
- `viewDidMoveToWindow` (page 202)
- `viewWillMoveToWindow:` (page 205)

Declared in

`NSView.h`

viewWillMoveToWindow:

Informs the receiver that it's being added to the view hierarchy of the specified window object (which may be `nil`).

```
- (void)viewWillMoveToWindow:(NSWindow *)newWindow
```

Parameters

`newWindow`

The window object that will be at the root of the receiver's new view hierarchy. If the view is being removed from a window and there is no new window, this parameter is `nil`.

Discussion

AppKit calls this method when the window of a view changes. It also calls it in cases where a view stays in the same window but its position in its view hierarchy changes. The view that moved also calls this method on all of its subviews, giving each of them a chance to respond to the change.

Subclasses can override this method to perform whatever actions are necessary. For example, when a window is deallocated, you can use this method to remove notification observers and bindings associated with the view.

When a window is deallocated, AppKit calls this method for each view in the window, passing `nil` for the `newWindow` parameter. AppKit does not necessarily call this method when closing a window, though. Closing a window usually just hides the window. Closed windows are deallocated only if their `isReleasedWhenClosed` method returns `YES`.

Availability
Available in OS X v10.0 and later.

See Also
- `viewDidMoveToSuperview` (page 202)
- `viewDidMoveToWindow` (page 202)
- `viewWillMoveToSuperview:` (page 205)

Declared in
`NSView.h`

viewWillStartLiveResize

Informs the receiver of the start of a live resize.

```
- (void)viewWillStartLiveResize
```

Discussion
In the simple case, a view is sent `viewWillStartLiveResize` before the first resize operation on the containing window and `viewDidEndLiveResize` (page 201) after the last resize operation. A view that is repeatedly added and removed from a window during live resize will receive only one `viewWillStartLiveResize` (on the first time it is added to the window) and one `viewDidEndLiveResize` (when the window has completed the live resize operation). This allows a superview such as `NSBrowser` object to add and remove its `NSMatrix` subviews during live resize without the `NSMatrix` object receiving multiple calls to these methods.

A view might allocate data structures to cache-drawing information in `viewWillStartLiveResize` and should clean up these data structures in `viewDidEndLiveResize` (page 201). In addition, a view that does optimized drawing during live resize might want to do full drawing after `viewDidEndLiveResize`, although a view should not assume that it has a drawing context in `viewDidEndLiveResize` (since it may have been removed from the window during live resize). A view that wants to redraw itself after live resize should call `[self setNeedsDisplay:YES]` in `viewDidEndLiveResize`.

A view subclass should call `super` from these methods.

Availability
Available in OS X v10.1 and later.

See Also

– `viewDidEndLiveResize` (page 201)
– `inLiveResize` (page 111)

Declared in

`NSView.h`

viewWithTag:

Returns the receiver's nearest descendant (including itself) with a specific tag, or `nil` if no subview has that tag.

```
– (id)viewWithTag:(NSInteger)aTag
```

Parameters

`aTag`

An integer identifier associated with a view object.

Availability

Available in OS X v10.0 and later.

See Also

– `tag` (page 193)

Related Sample Code

AnimatedTableView

MatrixMixerTest

Declared in

`NSView.h`

visibleRect

Returns the portion of the receiver not clipped by its superviews.

```
– (NSRect)visibleRect
```

Return Value

A rectangle defining the unclipped portion of the receiver.

Discussion

Visibility for this method is defined quite simply and doesn't account for whether other `NSView` objects (or windows) overlap the receiver or whether the receiver has a window at all. This method returns `NSZeroRect` if the receiver is effectively hidden.

During a printing operation the visible rectangle is further clipped to the page being imaged.

Availability
Available in OS X v10.0 and later.

See Also
`– setHidden:` (page 171)
`isVisible` (NSWindow)
`documentVisibleRect` (NSScrollView)
`documentVisibleRect` (NSClipView)

Related Sample Code
CIAnnotation
CITransitionSelectorSample2
DictionaryController
Rulers
TextEdit

Declared in
`NSView.h`

wantsBestResolutionOpenGLSurface

Returns a Boolean value indicating whether the view wants an OpenGL backing surface with resolution greater than 1 pixel per point.

```
– (BOOL)wantsBestResolutionOpenGLSurface
```

Return Value
`YES` if the view wants a high resolution backing surface or `NO` if it does not.

Discussion
AppKit calls this method for views that are bound to an `NSOpenGLContext` object. Layer-backed views ignore the value returned by this method and configure their own backing surface at an appropriate resolution.

The default implementation of this method returns `NO`, which always results in a backing surface with a resolution of 1 pixel per point. You can change the default value by calling the `setWantsBestResolutionOpenGLSurface:` (page 184) method.

If this method returns YES, the view must be able to convert between view units and pixel units when needed. For example, passing the view's bounds unmodified to the glViewport function yields incorrect results at backing scale factors other than 1.0. You can use the convertPointToBacking: (page 72), convertRectToBacking: (page 77), and convertSizeToBacking: (page 82) methods to convert coordinate values to the resolution of the backing store.

Availability
Available in OS X v10.7 and later.

Declared in
NSOpenGLView.h

wantsDefaultClipping

Returns whether the Application Kit's default clipping provided to drawRect: (page 96) implementations is in effect.

```
- (BOOL)wantsDefaultClipping
```

Return Value
YES if the default clipping is in effect, NO otherwise. By default, this method returns YES.

Discussion
Subclasses may override this method to return NO if they want to suppress the default clipping. They may want to do this in situations where drawing performance is critical to avoid the cost of setting up, enforcing, and cleaning up the clip path

A view that overrides this method to refuse the default clipping must either set up whatever clipping it requires or constrain its drawing exactly to the list of rectangles returned by getRectsBeingDrawn:count: (page 106). Failing to do so could result in corruption of other drawing in the view's window.

Availability
Available in OS X v10.3 and later.

Declared in
NSView.h

wantsLayer

Returns a Boolean value that indicates whether the receiver is using a layer as its backing store.

```
- (BOOL)wantsLayer
```

Return Value
YES if the receiver is using a Core Animation layer as its backing store, otherwise NO.

Availability
Available in OS X v10.5 and later.

See Also
– setWantsLayer: (page 184)

Related Sample Code
InfoBarStackView

Declared in
NSView.h

wantsRestingTouches

Returns whether the view wants resting touches.

```
– (BOOL)wantsRestingTouches
```

Return Value
YES if the view wants resting touches, otherwise NO. The default is NO.

Discussion
A resting touch occurs when a user rests their thumb on a device (for example, the glass trackpad of a MacBook).

By default, these touches are not delivered and are not included in the event's set of touches. Touches may transition in and out of resting at any time. Unless the view wants restingTouches, began / ended events are simulated as touches transition from resting to active and vice versa.

In general resting touches should be ignored.

Availability
Available in OS X v10.6 and later.

See Also
– setWantsRestingTouches: (page 186)

Declared in
NSView.h

wantsUpdateLayer

Returns a Boolean value indicating which drawing path the view takes when updating its contents.

```
- (BOOL)wantsUpdateLayer
```

Discussion

A view can update its contents using one of two techniques. It can draw those contents using its `drawRect:` (page 96) method or it can modify its underlying layer object directly. During the view update cycle, each dirty view calls this method on itself to determine which technique to use. The default implementation of this method returns `NO`, which causes the view to use its `drawRect:` method.

If your view is layer-backed and updates itself by modifying its layer, override this method and return `YES`. Modifying the layer is significantly faster than redrawing the layer contents using `drawRect:`. If you return `YES` from this method, you must also override the `updateLayer` method of your view and use it to make the changes to your layer. Do not use this method to modify your layer. Your implementation of this method should return `YES` or `NO` quickly and not perform other tasks.

If the `canDrawSubviewsIntoLayer` (page 64) method of this view returns `YES`, the view ignores the value returned by this method. Instead, the view always uses its `drawRect:` method to draw its content.

> **Note:** When using the `updateLayer` method to update your view, it is recommended that you set the view's redraw policy to `NSViewLayerContentsRedrawOnSetNeedsDisplay` (page 218). This policy lets you decide when you want to update the layer's contents.

Availability

Available in OS X v10.8 and later.

See Also

– `updateLayer` (page 198)

Declared in

`NSView.h`

widthAdjustLimit

Returns the fraction (from 0.0 to 1.0) of the page that can be pushed onto the next page during automatic pagination to prevent items such as small images or text columns from being divided across pages.

```
- (CGFloat)widthAdjustLimit
```

Discussion

This fraction is used to calculate the right edge limit for a `adjustPageWidthNew:left:right:limit:` (page 49) message.

Availability

Available in OS X v10.0 and later.

See Also

– `heightAdjustLimit` (page 109)

Declared in

`NSView.h`

willRemoveSubview:

Overridden by subclasses to perform additional actions before subviews are removed from the receiver.

```
– (void)willRemoveSubview:(NSView *)subview
```

Parameters

`subview`

The subview that will be removed.

Discussion

This method is invoked when `subview` receives a `removeFromSuperview` (page 145) message or `subview` is removed from the receiver due to it being added to another view with `addSubview:` (page 44).

Availability

Available in OS X v10.0 and later.

Declared in

`NSView.h`

window

Returns the receiver's window object, or `nil` if it has none.

```
– (NSWindow *)window
```

Availability

Available in OS X v10.0 and later.

See Also

– `superview` (page 192)

Related Sample Code

BlastApp

CIAnnotation

DragItemAround

Sketch+Accessibility

TextEdit

Declared in

`NSView.h`

writeEPSInsideRect:toPasteboard:

Writes EPS data that draws the region of the receiver within a specified rectangle onto a pasteboard.

```
- (void)writeEPSInsideRect:(NSRect)aRect toPasteboard:(NSPasteboard *)pboard
```

Parameters

`aRect`

A rectangle defining the region.

`pboard`

An object representing a pasteboard.

Availability

Available in OS X v10.0 and later.

See Also

– `dataWithEPSInsideRect:` (page 84)

Declared in

`NSView.h`

writePDFInsideRect:toPasteboard:

Writes PDF data that draws the region of the receiver within a specified rectangle onto a pasteboard.

```
- (void)writePDFInsideRect:(NSRect)aRect toPasteboard:(NSPasteboard *)pboard
```

Parameters

`aRect`

A rectangle defining the region.

`pboard`

An object representing a pasteboard.

Availability

Available in OS X v10.0 and later.

See Also

`– dataWithPDFInsideRect:` (page 85)

Declared in

`NSView.h`

Constants

NSBorderType

These constants specify the type of a view's border.

```
enum {
   NSNoBorder     = 0,
   NSLineBorder   = 1,
   NSBezelBorder  = 2,
   NSGrooveBorder = 3
};
typedef NSUInteger NSBorderType;
```

Constants

`NSBezelBorder`

A concave border that makes the view look sunken.

Available in OS X v10.0 and later.

Declared in `NSView.h`.

`NSGrooveBorder`

A thin border that looks etched around the image.

Available in OS X v10.0 and later.

Declared in `NSView.h`.

`NSLineBorder`

A black line border around the view.

Available in OS X v10.0 and later.

Declared in `NSView.h`.

`NSNoBorder`

No border.

Available in OS X v10.0 and later.

Declared in `NSView.h`.

Resizing masks

These constants are used by `setAutoresizingMask:` (page 157).

```
enum {
   NSViewNotSizable     = 0,
   NSViewMinXMargin     = 1,
   NSViewWidthSizable   = 2,
   NSViewMaxXMargin     = 4,
   NSViewMinYMargin     = 8,
   NSViewHeightSizable  = 16,
   NSViewMaxYMargin     = 32
};
```

Constants

`NSViewNotSizable`

The receiver cannot be resized.

Available in OS X v10.0 and later.

Declared in `NSView.h`.

`NSViewMinXMargin`

The left margin between the receiver and its superview is flexible.

Available in OS X v10.0 and later.

Declared in `NSView.h`.

`NSViewWidthSizable`

The receiver's width is flexible.

Available in OS X v10.0 and later.

Declared in `NSView.h`.

`NSViewMaxXMargin`

The right margin between the receiver and its superview is flexible.

Available in OS X v10.0 and later.

Declared in `NSView.h`.

`NSViewMinYMargin`

The bottom margin between the receiver and its superview is flexible.

Available in OS X v10.0 and later.

Declared in `NSView.h`.

`NSViewHeightSizable`

The receiver's height is flexible.

Available in OS X v10.0 and later.

Declared in `NSView.h`.

`NSViewMaxYMargin`

The top margin between the receiver and its superview is flexible.

Available in OS X v10.0 and later.

Declared in `NSView.h`.

NSToolTipTag

This type describes the rectangle used to identify a tool tip rectangle.

```
typedef NSInteger NSToolTipTag;
```

Discussion

If the value of this type is 0, it is invalid. See the methods `addToolTipRect:owner:userData:` (page 46) and `removeToolTip:` (page 146).

Availability

Available in OS X v10.0 and later.

Declared in

`NSView.h`

NSTrackingRectTag

This type describes the rectangle used to track the mouse.

```
typedef NSInteger NSTrackingRectTag;
```

Discussion

If the value of this type is 0, it is invalid. See the methods `addTrackingRect:owner:userData:assumeInside:` (page 47) and `removeTrackingRect:` (page 147).

Availability
Available in OS X v10.0 and later.

Declared in
`NSView.h`

Full Screen Mode Options

These constants are keys that you can use in the options dictionary in `enterFullScreenMode:withOptions:` (page 99) and `exitFullScreenModeWithOptions:` (page 101).

```
NSString * const NSFullScreenModeAllScreens;
NSString * const NSFullScreenModeSetting;
NSString * const NSFullScreenModeWindowLevel;
NSString * const NSFullScreenModeApplicationPresentationOptions;
```

Constants
`NSFullScreenModeAllScreens`

Key whose corresponding value specifies whether the view should take over all screens.

The corresponding value is an instance of `NSNumber` containing a Boolean value.

Available in OS X v10.5 and later.

Declared in `NSView.h`.

`NSFullScreenModeSetting`

Key whose corresponding value specifies the the full screen mode setting.

The corresponding value is an instance of `NSDictionary` that contains keys specified in `Display Mode Standard Properties` and `Display Mode Optional Properties` in *Quartz Display Services Reference*.

When the `NSFullScreenModeApplicationPresentationOptions` (page 218) is specified in the options dictionary specifying this option as well will cause an exception.

Available in OS X v10.5 and later.

Declared in `NSView.h`.

`NSFullScreenModeWindowLevel`

Key whose corresponding value specifies the screen mode window level.

The corresponding value is an instance of `NSNumber` containing an integer value.

Available in OS X v10.5 and later.

Declared in `NSView.h`.

`NSFullScreenModeApplicationPresentationOptions`

Key whose corresponding value specifies the application presentation options..

The corresponding value is an instance of `NSNumber` containing an unsigned integer value of `NSApplicationPresentationOptions`. Those options can be combined using the C bit-wise `OR` operator before created the `NSNumber` instance. See *NSApplication Class Reference* constants section `NSApplicationPresentationOptions` for more information on these options.

Available in OS X v10.5 and later.

Declared in `NSView.h`.

NSViewLayerContentsRedrawPolicy

These constants specify how layer resizing is handled when a view is layer-backed or layer-hosting. See `layerContentsRedrawPolicy` (page 119) and `setLayerContentsRedrawPolicy:` (page 174) for more information.

```
enum {
   NSViewLayerContentsRedrawNever                = 0,
   NSViewLayerContentsRedrawOnSetNeedsDisplay    = 1,
   NSViewLayerContentsRedrawDuringViewResize     = 2,
   NSViewLayerContentsRedrawBeforeViewResize     = 3,
   NSViewLayerContentsRedrawCrossfade            = 4
};
typedef NSInteger NSViewLayerContentsRedrawPolicy;
```

Constants

`NSViewLayerContentsRedrawNever`

Leave the layer's contents alone. Never mark the layer as needing display, or draw the view's contents to the layer. This is how developer created layers (layer-hosting views) are treated.

Available in OS X v10.6 and later.

Declared in `NSView.h`.

`NSViewLayerContentsRedrawOnSetNeedsDisplay`

Any of the `setNeedsDisplay...` methods sent to the view will cause the view redraw the affected layer parts by invoking the view's `drawRect:` (page 96), but neither the layer or the view are marked as needing display when the view's size changes.

Available in OS X v10.6 and later.

Declared in `NSView.h`.

`NSViewLayerContentsRedrawDuringViewResize`

Resize the view's backing-layer and redraw the view to the layer when the view's size changes. If the resize is animated, AppKit will drive the resize animation itself and will do this resize and redraw at each step of the animation. Affected parts of the layer will also be redrawn when the view is marked as needing display. This mode is a superset of `NSViewLayerContentsRedrawOnSetNeedsDisplay` (page 218). This is the way that layer-backed views are currently treated.

Available in OS X v10.6 and later.

Declared in `NSView.h`.

`NSViewLayerContentsRedrawBeforeViewResize`

Resize the layer and redraw the view to the layer when the view's size changes. This will be done just once at the beginning of a resize animation, not at each frame of the animation. Affected parts of the layer will also be redrawn when the view is marked as needing display. This mode is a superset of `NSViewLayerContentsRedrawOnSetNeedsDisplay` (page 218).

Available in OS X v10.6 and later.

Declared in `NSView.h`.

`NSViewLayerContentsRedrawCrossfade`

Redraw the layer contents at the new size and crossfade from the old contents to the new contents. Use this in conjunction with the `"NSViewLayerContentsPlacement"` (page 219) constants to get a nice crossfade animation for complex layer-backed views that cannot update correctly at each step of the animation.

Available in OS X v10.9 and later.

Declared in `NSView.h`.

NSViewLayerContentsPlacement

These constants specify the location of the layer content when the content is not re-rendered in response to view resizing. See `setLayerContentsPlacement:` (page 173) for more information.

```
enum {
   NSViewLayerContentsPlacementScaleAxesIndependently      =  0,
   NSViewLayerContentsPlacementScaleProportionallyToFit    =  1,
   NSViewLayerContentsPlacementScaleProportionallyToFill   =  2,
   NSViewLayerContentsPlacementCenter                      =  3,
   NSViewLayerContentsPlacementTop                         =  4,
   NSViewLayerContentsPlacementTopRight                    =  5,
   NSViewLayerContentsPlacementRight                       =  6,
   NSViewLayerContentsPlacementBottomRight                 =  7,
   NSViewLayerContentsPlacementBottom                      =  8,
   NSViewLayerContentsPlacementBottomLeft                  =  9,
   NSViewLayerContentsPlacementLeft                        = 10,
```

```
   NSViewLayerContentsPlacementTopLeft                    = 11
};
typedef NSInteger NSViewLayerContentsPlacement;
```

Constants

`NSViewLayerContentsPlacementScaleAxesIndependently`

The content is resized to fit the entire bounds rectangle.

Available in OS X v10.6 and later.

Declared in `NSView.h`.

`NSViewLayerContentsPlacementScaleProportionallyToFit`

The content is resized to fit the bounds rectangle, preserving the aspect of the content. If the content does not completely fill the bounds rectangle, the content is centered in the partial axis.

Available in OS X v10.6 and later.

Declared in `NSView.h`.

`NSViewLayerContentsPlacementScaleProportionallyToFill`

The content is resized to completely fill the bounds rectangle, while still preserving the aspect of the content. The content is centered in the axis it exceeds.

Available in OS X v10.6 and later.

Declared in `NSView.h`.

`NSViewLayerContentsPlacementCenter`

The content is horizontally and vertically centered in the bounds rectangle.

Available in OS X v10.6 and later.

Declared in `NSView.h`.

`NSViewLayerContentsPlacementTop`

The content is horizontally centered at the top-edge of the bounds rectangle.

Available in OS X v10.6 and later.

Declared in `NSView.h`.

`NSViewLayerContentsPlacementTopRight`

The content is positioned in the top-right corner of the bounds rectangle.

Available in OS X v10.6 and later.

Declared in `NSView.h`.

`NSViewLayerContentsPlacementRight`

The content is vertically centered at the right-edge of the bounds rectangle.

Available in OS X v10.6 and later.

Declared in `NSView.h`.

`NSViewLayerContentsPlacementBottomRight`

The content is positioned in the bottom-right corner of the bounds rectangle.

Available in OS X v10.6 and later.

Declared in `NSView.h`.

`NSViewLayerContentsPlacementBottom`

The content is horizontally centered at the bottom-edge of the bounds rectangle.

Available in OS X v10.6 and later.

Declared in `NSView.h`.

`NSViewLayerContentsPlacementBottomLeft`

The content is positioned in the bottom-left corner of the bounds rectangle.

Available in OS X v10.6 and later.

Declared in `NSView.h`.

`NSViewLayerContentsPlacementLeft`

The content is vertically centered at the left-edge of the bounds rectangle.

Available in OS X v10.6 and later.

Declared in `NSView.h`.

`NSViewLayerContentsPlacementTopLeft`

The content is positioned in the top-left corner of the bounds rectangle.

Available in OS X v10.6 and later.

Declared in `NSView.h`.

NSDefinition Presentation Constants

These constants are used to control how definition windows are displayed by `showDefinitionForAttributedString:range:options:baselineOriginProvider:` *(page189).Ifthisoption is unspecified, the definition will be shown in either of those presentation forms depending on the 'Contextual Menu:' setting in Dictionary application preferences.*

```
NSString * const NSDefinitionPresentationTypeKey;
NSString * const NSDefinitionPresentationTypeOverlay;
NSString * const NSDefinitionPresentationTypeDictionaryApplication;
```

Constants

`NSDefinitionPresentationTypeKey`

An optional key in the options dictionary that specifies the presentation type of the definition display. It can have a value of `NSDefinitionPresentationTypeOverlay` (page 222) or `NSDefinitionPresentationTypeDictionaryApplication` (page 222).

Available in OS X v10.6 and later.

Declared in `NSView.h`.

`NSDefinitionPresentationTypeOverlay`

A possible value of the `NSDefinitionPresentationTypeKey` (page 222) dictionary key that produces a small overlay window at the string location,

Available in OS X v10.6 and later.

Declared in `NSView.h`.

`NSDefinitionPresentationTypeDictionaryApplication`

A possible value of the `NSDefinitionPresentationTypeKey` (page 222) dictionary key that invokes Dictionary application to display the definition.

Available in OS X v10.6 and later.

Declared in `NSView.h`.

NSView Intrinsic Metric Constant

Used to indicate that a view has no intrinsic metric for a given numeric property.

```
const CGFloat NSViewNoInstrinsicMetric; // -1
```

Constants

`NSViewNoInstrinsicMetric`

Used to indicate that a view has no intrinsic metric for a given numeric property.

Available in OS X v10.7 and later.

Declared in `NSLayoutConstraint.h`.

Notifications

NSViewBoundsDidChangeNotification

Posted whenever the `NSView`'s bounds rectangle changes independently of the frame rectangle, if the `NSView` is configured using `setPostsBoundsChangedNotifications:` (page 179) to post such notifications.

The notification object is the `NSView` object whose bounds rectangle has changed. This notification does not contain a `userInfo` dictionary.

The following methods can result in notification posting:

- `setBounds:` (page 159)
- `setBoundsOrigin:` (page 160)
- `setBoundsRotation:` (page 161)
- `setBoundsSize:` (page 162)
- `translateOriginToPoint:` (page 194)
- `scaleUnitSquareToSize:` (page 152)
- `rotateByAngle:` (page 151)

Note that the bounds rectangle resizes automatically to track the frame rectangle. Because the primary change is that of the frame rectangle, however, `setFrame:` (page 167) and `setFrameSize:` (page 170) don't result in a bounds-changed notification.

Availability
Available in OS X v10.0 and later.

Declared in
`NSView.h`

NSViewFocusDidChangeNotification

Deprecated in OS X v10.4 and later. Posted for an `NSView` object and each of its descendants (recursively) whenever the frame or bounds geometry of the view changed.

Instead use `NSViewBoundsDidChangeNotification` and `NSViewFrameDidChangeNotification` to get the same information provided by this notification.

The notification object is the view whose geometry changed. This notification does not contain a `userInfo` dictionary.

Availability
Deprecated in OS X v10.4 and later.

See Also
`NSViewBoundsDidChangeNotification` (page 222)
`NSViewFrameDidChangeNotification` (page 224)

Declared in
NSView.h

NSViewFrameDidChangeNotification

Posted whenever the view's frame rectangle changes, if the view is configured using `setPostsFrameChangedNotifications:` *(page 180) to post such notifications.*

The notification object is the `NSView` object whose frame rectangle has changed. This notification does not contain a `userInfo` dictionary.

The following methods can result in notification posting:

- `setFrame:` (page 167)
- `setFrameOrigin:` (page 169)
- `setFrameRotation:` (page 169)
- `setFrameSize:` (page 170)

Availability
Available in OS X v10.0 and later.

Declared in
NSView.h

NSViewDidUpdateTrackingAreasNotification

Posted whenever an `NSView` *object recalculates its tracking areas.* It is sent after the view receives `updateTrackingAreas` (page 199).

Availability
Available in OS X v10.5 and later.

Declared in
NSView.h

NSViewGlobalFrameDidChangeNotification

Posted whenever an `NSView` *object that has attached surfaces (that is,* `NSOpenGLContext` *objects) moves to a different screen, or other cases where the* `NSOpenGLContext` *object needs to be updated.* The notification object is the surface's view. This notification does not contain a `userInfo` dictionary.

Availability
Available in OS X v10.1 and later.

Declared in
`NSView.h`

Deprecated NSView Methods

A method identified as deprecated has been superseded and may become unsupported in the future.

Deprecated in OS X v10.8

performMnemonic:

Implemented by subclasses to respond to mnemonics. (Deprecated in OS X v10.8.)

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

Parameters

`aString`

A string representing the mnemonic to handle.

Discussion

If the receiver's mnemonic is the same as the characters of the string `aString`, the receiver should take the appropriate action and return `YES`. Otherwise, it should return the result of invoking `super`'s implementation. The default implementation of this method simply passes the message down the view hierarchy (from superviews to subviews) and returns `NO` if none of the receiver's subviews responds `YES`. Mnemonics are not supported in OS X.

Availability

Available in OS X v10.0 and later.

Deprecated in OS X v10.8.

See Also

`- performKeyEquivalent:` (page 133)
`keyDown:` (`NSWindow`)

Declared in

`NSView.h`

Document Revision History

This table describes the changes to *NSView Class Reference*.

Date	Notes
2014-07-15	Bug fixes to resizeWithOldSuperviewSize:, NSToolTipTag, and NSTrackingRectTag. Noted that if `NSToolTipTag` (page 216) or `NSTrackingRectTag` (page 216) has a value of 0, it is uninitialized. Added requirement that overrides of `resizeWithOldSuperviewSize:` (page 150) that call `super` must call `super` before making changes to their own view's frame.
2014-03-10	Made minor correction to Overview.
2013-10-22	Documented missing methods and methods added in OS X v10.9.
2013-08-08	Noted that setAlphaValue: is supported on non-layer-backed views. See `setAlphaValue:` (page 156).
2012-07-23	Updated for high resolution on OS X. Added the method `viewDidChangeBackingProperties` (page 200).
2011-07-06	Updated for OS X v10.7. New focus ring functionality, text finder bar support, and drag initiation methods. Added new API for constraint-based layout system.
2010-03-24	Added details that rightMouseDown: does not send messages up the responder chain.
2009-12-04	Updated translateRectsNeedingDisplayInRect: description. Corrected typos.

Date	Notes
2009-11-17	Describes using UTIs to declare promise drag types.
2009-10-19	Expanded descriptions of enterFullScreenMode:withOptions: and exitFullScreenModeWithOptions: methods.
2009-05-28	Updated for OS X v10.6. Added touch handling, concurrent drawing, and new layer API.
2009-02-04	Corrected typos.
2009-01-06	Updated description of the bounds method to reflect its behavior when drawing the view in an NSOpenGLContext.
2008-10-15	TBD
2008-03-11	Corrected information about the isOpaque method that's found in the discussion of the drawRect: method.
2007-10-31	Corrected typos.
2007-07-24	Added new API introduced in OS X v10.5. Clarified discardCursorRects. Improved description of final parameter of addTrackingRect:owner:userData:assumeInside:
2006-05-23	First publication of this content as a separate document.

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