NSCell Class Reference

Cocoa > User Experience

2009-02-04

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

Simultaneously published in the United States and Canada.

Contents

NSCell Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/AppKit.framework
Availability	Available in Mac OS X v10.0 and later.
Companion guide	Control and Cell Programming Topics for Cocoa
Declared in	NSCell.h
Related sample code	QTKitMovieShuffler Quartz Composer WWDC 2005 TextEdit TextEditPlus TrackBall VertexPerformanceTest

Overview

The `NSCell` class provides a mechanism for displaying text or images in an `NSView` object without the overhead of a full `NSView` subclass. It's used heavily by most of the `NSControl` classes to implement their internal workings.

Adopted Protocols

NSCoding
- `encodeWithCoder:`
- `initWithCoder:`

NSCopying
- `copyWithZone:`

Tasks

Initializing a Cell

- `initImageCell:` (page 38)
  Returns an `NSCell` object initialized with the specified image and set to have the cell's default menu.
- `initTextCell:` (page 38)
  Returns an NSCell object initialized with the specified string and set to have the cell's default menu.

Managing Cell Values

- `setObjectValue:` (page 66)
  Sets the receiver's object value.
- `objectValue` (page 47)
  Returns the receiver's value as an Objective-C object
- `hasValidObjectValue` (page 34)
  Returns a Boolean value that indicates whether the receiver has a valid object value.
- `setIntValue:` (page 64)
  Sets the value of the receiver using an integer.
- `intValue` (page 39)
  Returns the receiver's value as an integer.
- `setIntegerValue:` (page 63)
  Sets the value of the receiver using an `NSInteger`.
- `integerValue` (page 38)
  Returns the receiver's value as an `NSInteger`.
- `setStringValue:` (page 70)
  Sets the value of the receiver's cell using an `NSString` object.
- `stringValue` (page 77)
  Returns the value of the receiver's cell as an `NSString` object.
- `setDoubleValue:` (page 58)
  Sets the value of the receiver's cell using a double-precision floating-point number.
- `doubleValue` (page 28)
  Returns the value of the receiver's cell as a double-precision floating-point number.
- `setFloatValue:` (page 60)
  Sets the value of the receiver's cell using a single-precision floating-point number.
- `floatValue` (page 32)
  Returns the value of the receiver's cell as a single-precision floating-point number.

Managing Cell Attributes

- `setCellAttribute:to:` (page 56)
  Sets the value for the specified cell attribute.

- `cellAttribute:` (page 24)
  Returns the value for the specified cell attribute.
- `setType:` (page 73)
  Sets the type of the cell, changing it to a text cell, image cell, or null cell.
- `type` (page 83)
  Returns the type of the receiver
- `setEnabled:` (page 59)
  Sets whether the receiver is enabled or disabled.
- `isEnabled` (page 41)
  Returns a Boolean value that indicates whether the receiver is enabled or disabled.
- `allowsUndo` (page 22)
  Returns a Boolean value that indicates whether the receiver assumes responsibility for undo operations.
- `setAllowsUndo:` (page 53)
  Sets whether the receiver assumes responsibility for undo operations within the cell.

Managing Display Attributes

- `setBezeled:` (page 55)
  Sets whether the receiver draws itself with a bezeled border.
- `isBezeled` (page 40)
  Returns a Boolean value that indicates whether the receiver has a bezeled border.
- `setBordered:` (page 55)
  Sets whether the receiver draws itself outlined with a plain border.
- `isBordered` (page 40)
  Returns a Boolean value that indicates whether the receiver has a plain border.
- `isOpaque` (page 42)
  Returns a Boolean value that indicates whether the receiver is opaque (nontransparent).
- `setControlTint:` (page 57)
  Sets the receiver's control tint.
- `controlTint` (page 27)
  Returns the receiver's control tint.
- `setBackgroundStyle:` (page 54)
  Sets the background style for the receiver.
- `backgroundStyle` (page 23)
  Returns the background style for the receiver.
- `interiorBackgroundStyle` (page 39)
  Returns the interior background style for the receiver.

Managing Cell State

- `allowsMixedState` (page 21)
  Returns a Boolean value that indicates whether the receiver supports three states.

- `nextState` (page 46)
  Returns the receiver's next state.
- `setAllowsMixedState:` (page 52)
  Sets whether the receiver supports three states or just two.
- `setNextState` (page 66)
  Changes the state of the receiver to its next state.
- `setState:` (page 69)
  Sets the receiver's state to the specified value.
- `state` (page 76)
  Returns the receiver's state.

Modifying Textual Attributes

- `setEditable:` (page 58)
  Sets whether the user can edit the receiver's text.
- `isEditable` (page 41)
  Returns a Boolean value that indicates whether the receiver is editable.
- `setSelectable:` (page 68)
  Sets whether text in the receiver can be selected.
- `isSelectable` (page 43)
  Returns a Boolean value that indicates whether the text of the receiver can be selected.
- `setScrollable:` (page 67)
  Sets whether excess text in the receiver is scrolled past the cell's bounds.
- `isScrollable` (page 43)
  Returns a Boolean value that indicates whether the receiver scrolls excess text past the cell's bounds.
- `setAlignment:` (page 52)
  Sets the alignment of text in the receiver.
- `alignment` (page 21)
  Returns the alignment of text in the receiver.
- `setFont:` (page 61)
  Sets the font to use when the receiver displays text.
- `font` (page 33)
  Returns the font used to display text in the receiver
- `lineBreakMode` (page 44)
  Returns the line break mode currently used when drawing text.
- `setLineBreakMode:` (page 64)
  Sets the line break mode to use when drawing text
- `truncatesLastVisibleLine` (page 83)
  Returns a Boolean value indicating whether the receiver truncates and adds the ellipsis character to the last visible line if the text doesn't fit into the cell bounds.
- `setTruncatesLastVisibleLine:` (page 72)
  Sets whether the receiver truncates and adds the ellipsis character to the last visible line if the text doesn't fit into the cell bounds.

- `setWraps:` (page 74)
  Sets whether text in the receiver wraps when its length exceeds the frame of the cell.
- `wraps` (page 84)
  Returns a Boolean value that indicates whether the receiver wraps its text when the text exceeds the borders of the cell.
- `baseWritingDirection` (page 23)
  Returns the initial writing direction used to determine the actual writing direction for text.
- `setBaseWritingDirection:` (page 54)
  Sets the initial writing direction used to determine the actual writing direction for text .
- `setAttributedStringValue:` (page 53)
  Sets the value of the receiver's cell using an attributed string.
- `attributedStringValue` (page 22)
  Returns the value of the receiver's cell as an attributed string using the receiver's formatter object (if one exists).
- `setAllowsEditingTextAttributes:` (page 52)
  Sets whether the receiver allows the user to edit textual attributes of its contents.
- `allowsEditingTextAttributes` (page 21)
  Returns a Boolean value that indicates whether the receiver allows user editing of textual attributes.
- `setImportsGraphics:` (page 63)
  Sets whether the receiver can import images into its text.
- `importsGraphics` (page 37)
  Returns a Boolean value that indicates whether the text of the receiver can contain imported graphics.
- `setUpFieldEditorAttributes:` (page 73)
  Configures the textual and background attributes of the receiver's field editor.
- `title` (page 81)
  Returns the receiver's title.
- `setTitle:` (page 71)
  Sets the title of the receiver.

Managing the Target and Action

- `setAction:` (page 51)
  Sets the cell's action method to the specified selector.
- `action` (page 20)
  Returns the default action-message selector associated with the cell.
- `setTarget:` (page 71)
  Sets the target object to receive action messages.
- `target` (page 81)
  Returns the target object of the receiver.
- `setContinuous:` (page 56)
  Sets whether the receiver's cell sends its action message continuously to its target during mouse tracking.

- `isContinuous` (page 40)
  Returns a Boolean value that indicates whether the receiver's cell sends its action message continuously to its target during mouse tracking.
- `sendActionOn:` (page 50)
  Sets the conditions on which the receiver sends action messages to its target.

Managing the Image

- `setImage:` (page 62)
  Sets the image to be displayed by the receiver.
- `image` (page 36)
  Returns the image displayed by the receiver (if any).
- `preparedImage` (page 47)
  Returns the prepared image for the receiver.

Managing the Tag

- `setTag:` (page 70)
  Sets the tag of the receiver.
- `tag` (page 78)
  Returns the tag identifying the receiver.

Formatting and Validating Data

- `setFormatter:` (page 62)
  Sets the receiver's formatter object.
- `formatter` (page 33)
  Returns the receiver's formatter object.
- `isEntryAcceptable:` (page 41)
  Returns whether a string representing a numeric or date value is formatted in a suitable way for the cell's entry type.
- `setFloatingPointFormat:left:right:` (page 59)
  Sets the autoranging and floating point number format of the receiver's cell.
- `entryType` (page 99) Deprecated in Mac OS X v10.0 and later
  Returns the type of data the user can type into the receiver. (Deprecated. Use a formatter instead—see `setFormatter:` (page 62).)
- `setEntryType:` (page 99) Deprecated in Mac OS X v10.0 and later
  Sets how numeric data is formatted in the receiver and places restrictions on acceptable input. (Deprecated. Use a formatter instead—see `setFormatter:` (page 62).)

Managing Menus

+ `defaultMenu` (page 19)
  Returns the default menu for instances of the receiver.

- `setMenu:` (page 65)
  Sets the contextual menu for the cell.

- `menu` (page 44)
  Returns the receiver's contextual menu.

- `menuForEvent:inRect:ofView:` (page 44)
  Returns the menu associated with the receiver and related to the specified event and frame.

Comparing Cells

- `compare:` (page 26)
  Compares the string values of the receiver another cell, disregarding case.

Respond to Keyboard Events

- `acceptsFirstResponder` (page 20)
  Returns a Boolean value that indicates whether the receiver accepts first responder status.

- `setShowsFirstResponder:` (page 69)
  Sets whether the receiver draws some indication of its first responder status.

- `showsFirstResponder` (page 75)
  Returns a Boolean value that indicates whether the receiver should draw some indication of its first responder status.

- `setTitleWithMnemonic:` (page 72)
  Sets the title of the receiver with one character in the string denoted as an access key.

- `mnemonic` (page 45)
  Returns the character in the receiver's title that appears underlined for use as a mnemonic.

- `refusesFirstResponder` (page 48)
  Returns a Boolean value that indicates whether the receiver should not become the first responder.

- `setMnemonicLocation:` (page 65)
  Sets the character of the receiver's title to be used as a mnemonic character.

- `setRefusesFirstResponder:` (page 66)
  Sets whether the receiver should not become the first responder.

- `mnemonicLocation` (page 45)
  Returns the position of the underlined mnemonic character in the receiver's title.

- `performClick:` (page 47)
  Simulates a single mouse click on the receiver.

Deriving Values

- `takeObjectValueFrom:` (page 80)
  Sets the value of the receiver's cell to the object value obtained from the specified object.
- `takeIntegerValueFrom:` (page 79)
  Sets the value of the receiver's cell to an integer value obtained from the specified object.
- `takeIntValueFrom:` (page 79)
  Sets the value of the receiver's cell to an integer value obtained from the specified object.
- `takeStringValueFrom:` (page 80)
  Sets the value of the receiver's cell to the string value obtained from the specified object.
- `takeDoubleValueFrom:` (page 78)
  Sets the value of the receiver's cell to a double-precision floating-point value obtained from the specified object.
- `takeFloatValueFrom:` (page 79)
  Sets the value of the receiver's cell to a single-precision floating-point value obtained from the specified object.

Representing an Object

- `setRepresentedObject:` (page 67)
  Sets the object represented by the receiver.
- `representedObject` (page 48)
  Returns the object the receiver represents.

Tracking the Mouse

- `trackMouse:inRect:ofView:untilMouseUp:` (page 82)
  Initiates the mouse tracking behavior in a cell.
- `startTrackingAt:inView:` (page 75)
  Begins tracking mouse events within the receiver.
- `continueTracking:at:inView:` (page 26)
  Returns a Boolean value that indicates whether mouse tracking should continue in the receiving cell.
- `stopTracking:at:inView:mouseIsUp:` (page 76)
  Stops tracking mouse events within the receiver.
- `mouseDownFlags` (page 46)
  Returns the modifier flags for the last (left) mouse-down event.
+ `prefersTrackingUntilMouseUp` (page 19)
  Returns a Boolean value that indicates whether tracking stops when the cursor leaves the cell.
- `getPeriodicDelay:interval:` (page 33)
  Returns the initial delay and repeat values for continuous sending of action messages to target objects.

Hit Testing

- `hitTestForEvent:inRect:ofView:` (page 36)
  Returns hit testing information for the receiver.

Managing the Cursor

- `resetCursorRect:inView:` (page 49)
  Sets the receiver to show the I-beam cursor while it tracks the mouse.

Handling Keyboard Alternatives

- `keyEquivalent` (page 43)
  Returns the key equivalent to clicking the cell.

Managing Focus Rings

+ `defaultFocusRingType` (page 19)
  Returns the default type of focus ring for the receiver.
- `setFocusRingType:` (page 61)
  Sets the type of focus ring to be used.
- `focusRingType` (page 32)
  Returns the type of focus ring currently set for the receiver.

Determining Cell Size

- `calcDrawInfo:` (page 24)
  Recalculates the cell geometry.
- `cellSize` (page 25)
  Returns the minimum size needed to display the receiver.
- `cellSizeForBounds:` (page 25)
  Returns the minimum size needed to display the receiver, constraining it to the specified rectangle.
- `drawingRectForBounds:` (page 28)
  Returns the rectangle within which the receiver draws itself
- `imageRectForBounds:` (page 37)
  Returns the rectangle in which the receiver draws its image.
- `titleRectForBounds:` (page 82)
  Returns the rectangle in which the receiver draws its title text.
- `controlSize` (page 27)
  Returns the size of the receiver.
- `setControlSize:` (page 56)
  Sets the size of the receiver.

Drawing and Highlighting

- `drawWithFrame:inView:` (page 30)
  Draws the receiver's border and then draws the interior of the cell.
- `highlightColorWithFrame:inView:` (page 35)
  Returns the color the receiver uses when drawing the selection highlight.
- `drawInteriorWithFrame:inView:` (page 28)
  Draws the interior portion of the receiver, which includes the image or text portion but does not include the border.
- `controlView` (page 27)
  Returns the receiver's control.
- `setControlView:` (page 57)
  Sets the receiver's control view.
- `highlight:withFrame:inView:` (page 34)
  Redraws the receiver with the specified highlight setting.
- `setHighlighted:` (page 62)
  Sets whether the receiver has a highlighted appearance.
- `isHighlighted` (page 42)
  Returns a Boolean value that indicates whether the receiver is highlighted.

Editing and Selecting Text

- `editWithFrame:inView:editor:delegate:event:` (page 30)
  Begins editing of the receiver's text using the specified field editor.
- `selectWithFrame:inView:editor:delegate:start:length:` (page 49)
  Selects the specified text range in the cell's field editor.
- `sendsActionOnEndEditing` (page 50)
  Returns a Boolean value that indicates whether the receiver's `NSControl` object sends its action message whenever the user finishes editing the cell's text.
- `setSendsActionOnEndEditing:` (page 68)
  Sets whether the receiver's NSControl object sends its action message whenever the user finishes editing the cell's text.
- `endEditing:` (page 31)
  Ends the editing of text in the receiver using the specified field editor.
- `wantsNotificationForMarkedText` (page 84)
  Returns a Boolean value that indicates whether the field editor initiated by the receiver should post text change notifications.

Managing Expansion Frames

- `expansionFrameWithFrame:inView:` (page 31)
  Returns the expansion cell frame for the receiver.
- `drawWithExpansionFrame:inView:` (page 29)
  Instructs the receiver to draw in an expansion frame.

Class Methods

defaultFocusRingType

Returns the default type of focus ring for the receiver.

```
+ (NSFocusRingType)defaultFocusRingType
```

Return Value
The default type of focus ring for the receiver (one of the values listed in `NSFocusRingType`).

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

Declared In
`NSCell.h`

defaultMenu

Returns the default menu for instances of the receiver.

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

Return Value
The default menu. The `NSCell` implementation of this method returns `nil`.

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

See Also
`- menu` (page 44)
`- setMenu:` (page 65)

Declared In
`NSCell.h`

prefersTrackingUntilMouseUp

Returns a Boolean value that indicates whether tracking stops when the cursor leaves the cell.

```
+ (BOOL)prefersTrackingUntilMouseUp
```

Return Value
`YES` if tracking stops when the cursor leaves the cell, otherwise `NO`.

Discussion
The default implementation returns `NO`. Subclasses may override this method to return a different value.

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

See Also
- `trackMouse:inRect:ofView:untilMouseUp:` (page 82)

Declared In
`NSCell.h`

Instance Methods

acceptsFirstResponder

Returns a Boolean value that indicates whether the receiver accepts first responder status.

```
- (BOOL)acceptsFirstResponder
```

Return Value
`YES` if the receiver can become the first responder, otherwise `NO`.

Discussion
The default value is `YES` if the receiver is enabled. Subclasses may override this method to return a different value.

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

See Also
- `performClick:` (page 47)
- `setShowsFirstResponder:` (page 69)
- `setTitleWithMnemonic:` (page 72)

Declared In
`NSCell.h`

action

Returns the default action-message selector associated with the cell.

```
- (SEL)action
```

Return Value
The selector associated with the cell. The `NSCell` implementation of this method returns `NULL` by default.

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

See Also
- `setAction:` (page 51)
- `setTarget:` (page 71)
- `target` (page 81)

Declared In
NSCell.h

alignment

Returns the alignment of text in the receiver.

```
- (NSTextAlignment)alignment
```

Return Value
The alignment of text in the receiver (one of the following constants: NSLeftTextAlignment, NSRightTextAlignment,NSCenterTextAlignment, NSJustifiedTextAlignment, NSNaturalTextAlignment).

Discussion
The default value is NSNaturalTextAlignment.

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

See Also
- setAlignment: (page 52)

Declared In
NSCell.h

allowsEditingTextAttributes

Returns a Boolean value that indicates whether the receiver allows user editing of textual attributes.

```
- (BOOL)allowsEditingTextAttributes
```

Return Value
YES if the receiver allows the user to edit textual attributes of the cell's text, otherwise NO.

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

See Also
- setAllowsEditingTextAttributes: (page 52)

Declared In
NSCell.h

allowsMixedState

Returns a Boolean value that indicates whether the receiver supports three states.

```
- (BOOL)allowsMixedState
```

Return Value
`YES` if the receiver supports all three states (on, off, and mixed), otherwise `NO` (the receiver supports only the on and off states).

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

See Also
- `nextState` (page 46)
- `setAllowsMixedState:` (page 52)
- `setNextState` (page 66)

Declared In
`NSCell.h`

allowsUndo

Returns a Boolean value that indicates whether the receiver assumes responsibility for undo operations.

```
- (BOOL)allowsUndo
```

Return Value
`YES` if the receiver handles undo operations, otherwise `NO`.

Discussion
By default, the `NSTextFieldCell` class uses this feature to handle undo operations for edited text. Other controls set a value that is appropriate for their implementation.

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

See Also
- `setAllowsUndo:` (page 53)

Declared In
`NSCell.h`

attributedStringValue

Returns the value of the receiver's cell as an attributed string using the receiver's formatter object (if one exists).

```
- (NSAttributedString *)attributedStringValue
```

Return Value
The value of the cell interpreted as an attributed string.

Discussion
The textual attributes are the default paragraph style, the receiver's font and alignment, and whether the receiver is enabled and scrollable.

For Mac OS X v10.3 and later: If you use a class that responds to the selector `attributedStringValue` for the object value of a cell, then the cell will use that method to fetch the string to draw rather than using `stringValue` (page 77).

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

See Also
`- setAttributedStringValue:` (page 53)

Declared In
`NSCell.h`

backgroundStyle

Returns the background style for the receiver.

```
- (NSBackgroundStyle)backgroundStyle
```

Return Value
The background style for the receiver.

Discussion
The background describes the surface the cell is drawn onto in `drawWithFrame:inView:` (page 30). A control typically sets this before it asks the cell to draw. A cell may draw differently based on background characteristics. For example, a tableview drawing a cell in a selected row might call `[cell setBackgroundStyle:NSBackgroundStyleDark]`. A text cell might decide to render its text white as a result. A rating-style level indicator might draw its stars white instead of gray.

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

Declared In
`NSCell.h`

baseWritingDirection

Returns the initial writing direction used to determine the actual writing direction for text.

```
- (NSWritingDirection)baseWritingDirection
```

Return Value
The initial writing direction the receiver uses to determine the actual writing direction for text (one of the following values: `NSWritingDirectionNatural`, `NSWritingDirectionLeftToRight`, `NSWritingDirectionRightToLeft`). If no writing direction is set, returns `NSWritingDirectionNatural`.

Discussion
The default value is `NSWritingDirectionNatural`.

The Text system uses this value as a hint for calculating the actual direction for displaying Unicode characters. You should not need to call this method directly.

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

See Also
- `setBaseWritingDirection:` (page 54)

Declared In
`NSCell.h`

calcDrawInfo:

Recalculates the cell geometry.

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

Parameters

aRect

The reference rectangle to use when calculating the cell information.

Discussion
Objects (such as controls) that manage `NSCell` objects generally maintain a flag that informs them if any of their cells have been modified in such a way that the location or size of the cell should be recomputed. If so, `calcSize` method of `NSControl` is automatically invoked prior to the display of the cell, and that method invokes the `calcDrawInfo:` method of the cell.

The default implementation of this method does nothing.

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

See Also
- `cellSize` (page 25)
- `drawingRectForBounds:` (page 28)

Declared In
`NSCell.h`

cellAttribute:

Returns the value for the specified cell attribute.

```
- (NSInteger)cellAttribute:(NSCellAttribute)aParameter
```

Parameters

aParameter

The cell attribute whose value you want to get. Attributes include the receiver's current state and whether it is disabled, editable, or highlighted.

Return Value
The value for the cell attribute specified by *aParameter*.

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

See Also
- `setCellAttribute:to:` (page 56)

Declared In
`NSCell.h`

cellSize

Returns the minimum size needed to display the receiver.

```
- (NSSize)cellSize
```

Return Value
The size of the cell, or the size (10000, 10000) if the receiver is not a text or image cell. If the cell is an image cell but no image has been set, returns `NSZeroSize`.

Discussion
This method takes into account of the size of the image or text within a certain offset determined by the border type of the cell.

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

See Also
- `drawingRectForBounds:` (page 28)

Declared In
`NSCell.h`

cellSizeForBounds:

Returns the minimum size needed to display the receiver, constraining it to the specified rectangle.

```
- (NSSize)cellSizeForBounds:(NSRect)aRect
```

Parameters

aRect

The size of the cell, or the size of the `aRect` parameter if the cell is not a text or image cell. If the cell is an image cell but no image has been set, returns `NSZeroSize`.

Discussion
This method takes into account of the size of the image or text within a certain offset determined by the border type of the cell. If the receiver is of text type, the text is resized to fit within *aRect* (as much as *aRect* is within the bounds of the cell).

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

See Also
- `drawingRectForBounds:` (page 28)

Declared In
`NSCell.h`

compare:

Compares the string values of the receiver another cell, disregarding case.

```
- (NSComparisonResult)compare:(id)otherCell
```

Parameters

otherCell

The cell to compare against the receiver. This parameter must be of type `NSCell`; if it is not, this method raises `NSBadComparisonException`.

This value must not be `nil`. If the value is `nil`, the behavior is undefined and may change in future versions of Mac OS X.

Return Value

`NSOrderedAscending` if the string value of the receiver precedes the string value of *otherCell* in lexical ordering, `NSOrderedSame` if the string values are equivalent in lexical value, and `NSOrderedDescending` string value of the receiver follows the string value of *otherCell* in lexical ordering.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSCell.h`

continueTracking:at:inView:

Returns a Boolean value that indicates whether mouse tracking should continue in the receiving cell.

```
- (BOOL)continueTracking:(NSPoint)lastPoint at:(NSPoint)currentPoint inView:(NSView
    *)controlView
```

Parameters

lastPoint

Contains either the initial location of the cursor when tracking began or the previous current point.

currentPoint

The current location of the cursor.

controlView

The `NSControl` object managing the receiver.

Return Value

`YES` if mouse tracking should continue, otherwise `NO`.

Discussion

This method is invoked in `trackMouse:inRect:ofView:untilMouseUp:` (page 82). The default implementation returns `YES` if the cell is set to continuously send action messages to its target when the mouse button is down or the mouse is being dragged. Subclasses can override this method to provide more sophisticated tracking behavior.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- startTrackingAt:inView:` (page 75)

`- stopTracking:at:inView:mouseIsUp:` (page 76)

Declared In
NSCell.h

controlSize

Returns the size of the receiver.

```
- (NSControlSize)controlSize
```

Return Value
A value that specifies the size of the receiver (for possible values, see "Control Sizes" (page 94)).

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

See Also
- setControlSize: (page 56)

Declared In
NSCell.h

controlTint

Returns the receiver's control tint.

```
- (NSControlTint)controlTint
```

Return Value
An NSControlTint (page 93) value that specifies the tint of the receiver (see "Control Tints" (page 93) for possible values).

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

See Also
- setControlTint: (page 57)

Declared In
NSCell.h

controlView

Returns the receiver's control.

```
- (NSView *)controlView
```

Return Value
The view (normally an NSControl object) associated with this cell. The default implementation returns nil.

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

See Also
- `drawWithFrame:inView:` (page 30)
- `setControlView:` (page 57)

Declared In
`NSCell.h`

doubleValue

Returns the value of the receiver's cell as a double-precision floating-point number.

```
- (double)doubleValue
```

Return Value
The value of the cell interpreted as a double-precision floating-point number. If the receiver is not a text-type cell or the cell value is not scannable, returns `0`.

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

See Also
- `setDoubleValue:` (page 58)

Declared In
`NSCell.h`

drawingRectForBounds:

Returns the rectangle within which the receiver draws itself

```
- (NSRect)drawingRectForBounds:(NSRect)theRect
```

Parameters
theRect
The bounding rectangle of the receiver.

Return Value
The rectangle in which the receiver draws itself. This rectangle is slightly inset from the one in `theRect`.

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

See Also
- `calcSize` (`NSControl`)

Declared In
`NSCell.h`

drawInteriorWithFrame:inView:

Draws the interior portion of the receiver, which includes the image or text portion but does not include the border.

```
- (void)drawInteriorWithFrame:(NSRect)cellFrame inView:(NSView *)controlView
```

Parameters

cellFrame

The bounding rectangle of the receiver, or a portion of the bounding rectangle.

controlView

The control that manages the cell.

Discussion

Text-type `NSCell` objects display their contents in a rectangle slightly inset from *cellFrame* using a global `NSText` object. Image-type `NSCell` objects display their contents centered within *cellFrame*. If the proper attributes are set, this method also displays the dotted-line rectangle to indicate if the control is the first responder and highlights the cell. This method is invoked from the `drawCellInside:` method of `NSControl` to visually update what the cell displays when its contents change. The drawing done by the `NSCell` implementation is minimal and becomes more complex in objects such as `NSButtonCell` and `NSSliderCell`.

This method draws the cell in the currently focused view, which can be different from the *controlView* passed in. Taking advantage of this is not recommended.

Subclasses often override this method to provide more sophisticated drawing of cell contents. Because `drawWithFrame:inView:` (page 30) invokes `drawInteriorWithFrame:inView:` after it draws the cell's border, do not invoke `drawWithFrame:inView:` (page 30) in your override implementation.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `isHighlighted` (page 42)
- `setShowsFirstResponder:` (page 69)

Declared In

`NSCell.h`

drawWithExpansionFrame:inView:

Instructs the receiver to draw in an expansion frame.

```
- (void)drawWithExpansionFrame:(NSRect)cellFrame inView:(NSView *)view
```

Parameters

cellFrame

The frame in which to draw.

view

The view in which to draw. This view may be different from the original view that the cell appeared in.

Discussion

This method allows the cell to perform custom expansion tool tip drawing. By default, `NSCell` simply calls `drawWithFrame:inView:` (page 30).

Availability

Available in Mac OS X v10.5 and later.

See Also
- `expansionFrameWithFrame:inView:` (page 31)

Declared In
`NSCell.h`

drawWithFrame:inView:

Draws the receiver's border and then draws the interior of the cell.

```
- (void)drawWithFrame:(NSRect)cellFrame inView:(NSView *)controlView
```

Parameters

cellFrame
The bounding rectangle of the receiver.

controlView
The control that manages the cell.

Discussion
This method draws the cell in the currently focused view, which can be different from the *controlView* passed in. Taking advantage of this behavior is not recommended, however.

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

See Also
- `drawInteriorWithFrame:inView:` (page 28)

Declared In
`NSCell.h`

editWithFrame:inView:editor:delegate:event:

Begins editing of the receiver's text using the specified field editor.

```
- (void)editWithFrame:(NSRect)aRect inView:(NSView *)controlView editor:(NSText
    *)textObj delegate:(id)anObject event:(NSEvent *)theEvent
```

Parameters

aRect
The bounding rectangle of the cell.

controlView
The control that manages the cell.

textObj
The field editor to use for editing the cell.

anObject
The object to use as a delegate for the field editor (`textObj` parameter). This delegate object receives various `NSText` delegation and notification methods during the course of editing the cell's contents.

theEvent
The `NSLeftMouseDown` event that initiated the editing behavior.

Discussion
If the receiver isn't a text-type `NSCell` object, no editing is performed. Otherwise, the field editor (*`textObj`*) is sized to *`aRect`* and its superview is set to *`controlView`*, so it exactly covers the receiver. The field editor is then activated and editing begins. It's the responsibility of the delegate to end editing when responding to `textShouldEndEditing:`. Upon ending the editing session, the delegate should remove any data from the field editor.

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

See Also
- `endEditing:` (page 31)
- `selectWithFrame:inView:editor:delegate:start:length:` (page 49)

Declared In
`NSCell.h`

endEditing:

Ends the editing of text in the receiver using the specified field editor.

```
- (void)endEditing:(NSText *)textObj
```

Parameters
`textObj`
The field editor currently handling the editing of the cell's content.

Discussion
Ends any editing of text that began with a call to `editWithFrame:inView:editor:delegate:event:` (page 30) or `selectWithFrame:inView:editor:delegate:start:length:` (page 49).

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

Declared In
`NSCell.h`

expansionFrameWithFrame:inView:

Returns the expansion cell frame for the receiver.

```
- (NSRect)expansionFrameWithFrame:(NSRect)cellFrame inView:(NSView *)view
```

Parameters
`cellFrame`
The frame for the receiver.

`view`
The view in which the receiver will be drawn.

Return Value
The expansion cell frame for the receiver. If the frame is not too small, return an empty rect (`NSZeroRect`), and no expansion tool tip view will be shown.

Discussion
This method allows the cell to return an expansion cell frame if *cellFrame* is too small for the entire contents in the view. When the mouse is hovered over the cell in certain controls, the full cell contents are shown in a special floating tool tip view. By default, `NSCell` returns `NSZeroRect`, while some subclasses (such as `NSTextFieldCell`) will return the proper frame when required.

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

See Also
`- drawWithExpansionFrame:inView:` (page 29)

Declared In
`NSCell.h`

floatValue

Returns the value of the receiver's cell as a single-precision floating-point number.

```
- (float)floatValue
```

Return Value
The value of the cell interpreted as a single-precision floating-point number. If the receiver is not a text-type cell or the cell value is not scannable, returns `0`.

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

Declared In
`NSCell.h`

focusRingType

Returns the type of focus ring currently set for the receiver.

```
- (NSFocusRingType)focusRingType
```

Return Value
The type of focus ring currently set for the receiver (one of the values listed in `NSFocusRingType`).

Discussion
You can disable a view's focus ring drawing by overriding this method so it always returns `NSFocusRingTypeNone`, or by calling `setFocusRingType:` (page 61) with `NSFocusRingTypeNone`. You should only disable a view from drawing its focus ring if you want to draw your own focus ring, or if there isn't sufficient space to display a focus ring in the default location.

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

See Also
`- setFocusRingType:` (page 61)
`+ defaultFocusRingType` (page 19)

Declared In
NSCell.h

font

Returns the font used to display text in the receiver

```
- (NSFont *)font
```

Return Value
The receiver's current font, or `nil` if the receiver is not a text-type cell.

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

See Also
- `setFont:` (page 61)

Related Sample Code
QTKitMovieShuffler

Declared In
NSCell.h

formatter

Returns the receiver's formatter object.

```
- (id)formatter
```

Return Value
An object of type `NSFormatter` used to format the receiver's content.

Discussion
The returned object handles translation of the receiver’s contents between its onscreen representation and its object value.

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

See Also
- `setFormatter:` (page 62)

Declared In
NSCell.h

getPeriodicDelay:interval:

Returns the initial delay and repeat values for continuous sending of action messages to target objects.

```
- (void)getPeriodicDelay:(float *)delay interval:(float *)interval
```

Parameters

delay

On input, a pointer to a floating-point variable. On output, the variable contains the current delay (measured in seconds) before messages are sent. This parameter must not be NULL.

interval

On input, a pointer to a floating point variable. On output, the variable contains the interval (measured in seconds) at which messages are sent. This parameter must not be NULL.

Discussion

The default implementation returns a delay of 0.2 and an interval of 0.025 seconds. Subclasses can override this method to supply their own delay and interval values.

Availability

Available in Mac OS X v10.0 and later.

See Also

- isContinuous (page 40)
- setContinuous: (page 56)

Declared In

NSCell.h

hasValidObjectValue

Returns a Boolean value that indicates whether the receiver has a valid object value.

```
- (BOOL)hasValidObjectValue
```

Return Value

YES if the cell has a valid object value, otherwise NO.

Discussion

A valid object value is one that the receiver's formatter can "understand." Objects are always assumed to be valid unless they are rejected by the formatter. Invalid objects can still be accepted by the delegate of the receiver's NSControl object (using the control:didFailToFormatString:errorDescription: delegate method).

Availability

Available in Mac OS X v10.0 and later.

See Also

- objectValue (page 47)
- setObjectValue: (page 66)

Declared In

NSCell.h

highlight:withFrame:inView:

Redraws the receiver with the specified highlight setting.

```
- (void)highlight:(BOOL)flag withFrame:(NSRect)cellFrame inView:(NSView *)controlView
```

Parameters

flag

If YES, the cell is redrawn with a highlight; otherwise, if NO, the highlight is removed.

cellFrame

The bounding rectangle of the receiver.

controlView

The control that manages the cell.

Discussion

Note that the NSCell highlighting does not appear when highlighted cells are printed (although instances of NSTextFieldCell, NSButtonCell, and others can print themselves highlighted). Generally, you cannot depend on highlighting being printed because implementations of this method may choose (or not choose) to use transparency.

Availability

Available in Mac OS X v10.0 and later.

See Also

- drawWithFrame:inView: (page 30)
- isHighlighted (page 42)

Declared In

NSCell.h

highlightColorWithFrame:inView:

Returns the color the receiver uses when drawing the selection highlight.

```
- (NSColor *)highlightColorWithFrame:(NSRect)cellFrame inView:(NSView *)controlView
```

Parameters

cellFrame

The bounding rectangle of the receiver.

controlView

The control that manages the cell.

Return Value

The color the receiver uses when drawing the selection highlight.

Discussion

You should not assume that a cell would necessarily want to draw itself with the value returned from selectedControlColor. A cell may wish to draw with different a selection highlight color depending on such things as the key state of its *controlView*.

Availability

Available in Mac OS X v10.1 and later.

Declared In

NSCell.h

hitTestForEvent:inRect:ofView:

Returns hit testing information for the receiver.

```
- (NSUInteger)hitTestForEvent:(NSEvent *)event inRect:(NSRect)cellFrame
    ofView:(NSView *)controlView
```

Parameters

event

The current event.

cellFrame

The cell's frame.

controlView

The control object in which the cell is located.

Return Value

A constant that specifies the type of area in which the event occurred—see "Hit Testing" (page 95) for values.

Discussion

You can use a bit-wise mask to look for a specific value when calling this method—see "Hit Testing" (page 95) for values.

Generally, this method should be overridden by custom `NSCell` subclasses to return the correct result. Currently, it is called by some multi-cell views, such as `NSTableView`.

By default, `NSCell` looks at the cell type and does the following:

- `NSImageCellType`: If the image exists and the event point is in the image returns `NSCellHitContentArea`, otherwise `NSCellHitNone`.
- `NSTextCellType` (also applies to `NSTextFieldCell`):

  If there is text: If the event point hits in the text, return `NSCellHitContentArea`. Additionally, if the cell is enabled return `NSCellHitContentArea | NSCellHitEditableTextArea`.

  If there is not text: return `NSCellHitNone`.
- `NSNullCellType` (this is the default that applies to non text or image cells who don't override `hitTestForEvent:inRect:ofView:`):

  Return `NSCellHitContentArea` by default;

  If the cell not disabled, and it would track, return `NSCellHitContentArea | NSCellHitTrackableArea`.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`NSCell.h`

image

Returns the image displayed by the receiver (if any).

```
- (NSImage *)image
```

Return Value
The image displayed by the receiver, or `nil` if the receiver is not an image-type cell.

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

See Also
`- setImage:` (page 62)

Declared In
`NSCell.h`

imageRectForBounds:

Returns the rectangle in which the receiver draws its image.

```
- (NSRect)imageRectForBounds:(NSRect)theRect
```

Parameters
theRect
The bounding rectangle of the receiver.

Return Value
The rectangle in which the receiver draws its image. This rectangle is slightly offset from the one in `theRect`.

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

See Also
`- cellSizeForBounds:` (page 25)
`- drawingRectForBounds:` (page 28)

Declared In
`NSCell.h`

importsGraphics

Returns a Boolean value that indicates whether the text of the receiver can contain imported graphics.

```
- (BOOL)importsGraphics
```

Return Value
`YES` if the receiver's text is in the RTFD format and supports imported graphics, otherwise `NO`.

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

See Also
`- setImportsGraphics:` (page 63)

Declared In
`NSCell.h`

initImageCell:

Returns an `NSCell` object initialized with the specified image and set to have the cell's default menu.

```
- (id)initImageCell:(NSImage *)anImage
```

Parameters

anImage

The image to use for the cell. If this parameter is `nil`, no image is set.

Return Value
An initialized `NSCell` object, or `nil` if the cell could not be initialized.

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

Declared In
`NSCell.h`

initTextCell:

Returns an NSCell object initialized with the specified string and set to have the cell's default menu.

```
- (id)initTextCell:(NSString *)aString
```

Parameters

aString

The initial string to use for the cell.

Return Value
An initialized `NSCell` object, or `nil` if the cell could not be initialized.

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

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

Declared In
`NSCell.h`

integerValue

Returns the receiver's value as an `NSInteger`.

```
- (NSInteger)integerValue
```

Return Value
The value of the cell interpreted as an `NSInteger`. If the receiver is not a text-type cell or the cell value is not scannable, returns `0`.

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

See Also
- `setIntegerValue:` (page 63)
- `intValue` (page 39)

Declared In
`NSCell.h`

interiorBackgroundStyle

Returns the interior background style for the receiver.

```
- (NSBackgroundStyle)interiorBackgroundStyle
```

Return Value
Returns the interior background style for the receiver.

Discussion
The interior background style describes the surface drawn onto in `drawInteriorWithFrame:inView:` (page 28). This is often the same as the `backgroundStyle` (page 23), but a button that draws a bezel would have a different `interiorBackgroundStyle`.

This is both an override point and a useful method to call. In a custom button with a custom bezel you can override this method to describe that surface. A cell that has custom interior drawing might query this method to help pick an image that looks good on the cell. Calling this method gives you some independence from changes in framework art style.

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

Declared In
`NSCell.h`

intValue

Returns the receiver's value as an integer.

```
- (int)intValue
```

Return Value
The value of the cell interpreted as an integer. If the receiver is not a text-type cell or the cell value is not scannable, returns `0`.

Discussion
On Mac OS X v10.5 and later, you should use `integerValue` (page 38) instead.

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

See Also
- `integerValue` (page 38)
- `setIntValue:` (page 64)

Related Sample Code
VertexPerformanceTest

Declared In
`NSCell.h`

isBezeled

Returns a Boolean value that indicates whether the receiver has a bezeled border.

```
- (BOOL)isBezeled
```

Return Value
`YES` if the receiver has a bezeled border, otherwise `NO`.

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

See Also
`- setBezeled:` (page 55)

Declared In
`NSCell.h`

isBordered

Returns a Boolean value that indicates whether the receiver has a plain border.

```
- (BOOL)isBordered
```

Return Value
`YES` if the receiver has a plain border, otherwise `NO`.

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

See Also
`- setBordered:` (page 55)

Declared In
`NSCell.h`

isContinuous

Returns a Boolean value that indicates whether the receiver's cell sends its action message continuously to its target during mouse tracking.

```
- (BOOL)isContinuous
```

Return Value
`YES` if the action message should be sent continuously, otherwise `NO`.

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

See Also
- `setContinuous:` (page 56)

Related Sample Code
AnimatedSlider

Declared In
`NSCell.h`

isEditable

Returns a Boolean value that indicates whether the receiver is editable.

```
- (BOOL)isEditable
```

Return Value
`YES` if the receiver is editable, otherwise `NO`.

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

See Also
- `setEditable:` (page 58)

Declared In
`NSCell.h`

isEnabled

Returns a Boolean value that indicates whether the receiver is enabled or disabled.

```
- (BOOL)isEnabled
```

Return Value
`YES` if the receiver is enabled, otherwise `NO`.

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

See Also
- `setEnabled:` (page 59)

Declared In
`NSCell.h`

isEntryAcceptable:

Returns whether a string representing a numeric or date value is formatted in a suitable way for the cell's entry type.

```
- (BOOL)isEntryAcceptable:(NSString *)aString
```

Parameters

aString

A string containing the numeric or date value.

Return Value

YES if *aString* is formatted appropriately for the receiver, otherwise NO.

Discussion

This method is being deprecated in favor of a new class of formatter objects. For more information, see NSFormatter. This documentation is provided only for developers who need to modify older applications

Availability

Available in Mac OS X v10.0 and later.

See Also

- entryType (page 99)
- setEntryType: (page 99)

Declared In

NSCell.h

isHighlighted

Returns a Boolean value that indicates whether the receiver is highlighted.

```
- (BOOL)isHighlighted
```

Return Value

YES if the receiver has a highlight, otherwise NO.

Availability

Available in Mac OS X v10.0 and later.

Declared In

NSCell.h

isOpaque

Returns a Boolean value that indicates whether the receiver is opaque (nontransparent).

```
- (BOOL)isOpaque
```

Return Value

YES if the receiver is opaque, otherwise NO to indicate the receiver might have some transparency.

Availability

Available in Mac OS X v10.0 and later.

Declared In

NSCell.h

isScrollable

Returns a Boolean value that indicates whether the receiver scrolls excess text past the cell's bounds.

```
- (BOOL)isScrollable
```

Return Value
`YES` if excess text scrolls past the cell's bounds, otherwise `NO` (text wrapping is enabled).

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

See Also
`- setScrollable:` (page 67)

Declared In
`NSCell.h`

isSelectable

Returns a Boolean value that indicates whether the text of the receiver can be selected.

```
- (BOOL)isSelectable
```

Return Value
`YES` if the receiver's text can be selected, otherwise `NO`.

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

See Also
`- setSelectable:` (page 68)

Declared In
`NSCell.h`

keyEquivalent

Returns the key equivalent to clicking the cell.

```
- (NSString *)keyEquivalent
```

Return Value
An empty string object.

Discussion
Subclasses can override this method to return a string with a valid character for the key equivalent.

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

Declared In
`NSCell.h`

lineBreakMode

Returns the line break mode currently used when drawing text.

```
- (NSLineBreakMode)lineBreakMode
```

Return Value
The line break mode the receiver currently uses when drawing text (one of the following constants: `NSLineBreakByWordWrapping`, `NSLineBreakByCharWrapping`, `NSLineBreakByClipping`, `NSLineBreakByTruncatingHead`, `NSLineBreakByTruncatingTail`, or `NSLineBreakByTruncatingMiddle`).

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

See Also
- `setLineBreakMode:` (page 64)
- `truncatesLastVisibleLine` (page 83)

Declared In
`NSCell.h`

menu

Returns the receiver's contextual menu.

```
- (NSMenu *)menu
```

Return Value
The receiver's contextual menu, or `nil` if no menu is assigned.

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

See Also
- `setMenu:` (page 65)

Related Sample Code
Clock Control

Declared In
`NSCell.h`

menuForEvent:inRect:ofView:

Returns the menu associated with the receiver and related to the specified event and frame.

```
- (NSMenu *)menuForEvent:(NSEvent *)anEvent inRect:(NSRect)cellFrame ofView:(NSView
    *)aView
```

Parameters

anEvent
The event used to find the menu.

cellFrame
The cell's rectangle. This rectangle indicates the region containing the cursor.

aView
The view that manages the receiver. This is usually the control object that owns the cell.

Return Value
The menu associated with the cell and event parameters, or `nil` if no menu is set.

Discussion
This method is usually invoked by the `NSControl` object (*aView*) managing the receiver. The default implementation simply invokes `menu` (page 44) and returns `nil` if no menu has been set. Subclasses can override to customize the returned menu according to the event received and the area in which the mouse event occurs.

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

Declared In
`NSCell.h`

mnemonic

Returns the character in the receiver's title that appears underlined for use as a mnemonic.

```
- (NSString *)mnemonic
```

Return Value
A string containing the mnemonic character, or an empty string if no mnemonic character is set.

Discussion
Mnemonics are not supported in Mac OS X

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

See Also
`- setTitleWithMnemonic:` (page 72)

Declared In
`NSCell.h`

mnemonicLocation

Returns the position of the underlined mnemonic character in the receiver's title.

```
- (NSUInteger)mnemonicLocation
```

Return Value
A zero-based index into the receiver's title string indicating the position of the character. If there is no mnemonic character, this method returns `NSNotFound`.

Discussion
Mnemonics are not supported in Mac OS X.

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

See Also
- `setMnemonicLocation:` (page 65)

Declared In
`NSCell.h`

mouseDownFlags

Returns the modifier flags for the last (left) mouse-down event.

```
- (NSInteger)mouseDownFlags
```

Return Value
The modifier flags, or `0` if tracking has not yet occurred or no modifier keys accompanied the mouse-down event.

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

See Also
- `modifierFlags` (`NSEvent`)

Declared In
`NSCell.h`

nextState

Returns the receiver's next state.

```
- (NSInteger)nextState
```

Return Value
The receiver's next state (for possible values, see "`Cell States`" (page 91)).

Discussion
If the receiver has three states, it cycles through them in this order: on, off, mixed, on, and so forth. If the receiver has two states, it toggles between them.

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

See Also
- `allowsMixedState` (page 21)
- `setAllowsMixedState:` (page 52)
- `setNextState` (page 66)

Declared In
`NSCell.h`

objectValue

Returns the receiver's value as an Objective-C object

```
- (id)objectValue
```

Return Value
The receiver's object value, or `nil` if a valid object has not been associated with the receiver.

Discussion
To be valid object value, the receiver must have a formatter capable of converting the object to and from its textual representation.

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

Related Sample Code
Clock Control

Declared In
`NSCell.h`

performClick:

Simulates a single mouse click on the receiver.

```
- (void)performClick:(id)sender
```

Parameters

sender

The object to use as the sender of the event (if the receiver's control view is not valid). This object must be a subclass of `NSView`.

Discussion
This method performs the receiver's action on its target. The receiver must be enabled to perform the action. If the receiver's control view is valid, that view is used as the sender; otherwise, the value in *sender* is used.

The receiver of this message must be a cell of type `NSActionCell`. This method raises an exception if the action message cannot be successfully sent.

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

See Also
- controlView (page 27)

Declared In
`NSCell.h`

preparedImage

Returns the prepared image for the receiver.

```
- (NSImage *)preparedImage
```

Return Value
The prepared image for the receiver.

Discussion
By default, this method returns a processed version of the cell's image or alternate image that takes into account `interiorBackgroundStyle` (page 39) and state. For example, a button cell might display a darker version of the cell's image when pressed. You can override this method to return an image based on any arbitrary parameters.

refusesFirstResponder

Returns a Boolean value that indicates whether the receiver should not become the first responder.

```
- (BOOL)refusesFirstResponder
```

Return Value
`YES` if the receiver should never become the first responder, otherwise `NO` if the receiver can become the first responder.

Discussion
To find out whether the receiver can become first responder at this time, use the method `acceptsFirstResponder` (page 20).

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

See Also
`- setRefusesFirstResponder:` (page 66)

Declared In
`NSCell.h`

representedObject

Returns the object the receiver represents.

```
- (id)representedObject
```

Return Value
The object represented by the receiver.

Discussion
Represented objects let you link a cell to an appropriate object. For example, you could have a pop-up list of color names, and the represented objects could be the appropriate `NSColor` objects.

Special Considerations

Note that if you copy an `NSCell` instance, the represented object in the copy is set to `nil`.

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

See Also
`- setRepresentedObject:` (page 67)

Declared In
NSCell.h

resetCursorRect:inView:

Sets the receiver to show the I-beam cursor while it tracks the mouse.

```
- (void)resetCursorRect:(NSRect)cellFrame inView:(NSView *)controlView
```

Parameters

cellFrame
The rectangle in which to display the I-beam cursor.

controlView
The control that manages the cell.

Discussion
The receiver must be an enabled and selectable (or editable) text-type cell.

This method is invoked by `resetCursorRects` and in general you do not need to call this method unless you have a custom `NSView` that uses a cell.

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

Declared In
NSCell.h

selectWithFrame:inView:editor:delegate:start:length:

Selects the specified text range in the cell's field editor.

```
- (void)selectWithFrame:(NSRect)aRect inView:(NSView *)controlView editor:(NSText
    *)textObj delegate:(id)anObject start:(NSInteger)selStart
   length:(NSInteger)selLength
```

Parameters

aRect
The bounding rectangle of the cell.

controlView
The control that manages the cell.

textObj
The field editor to use for editing the cell.

anObject
The object to use as a delegate for the field editor (`textObj` parameter). This delegate object receives various `NSText` delegation and notification methods during the course of editing the cell's contents.

selStart
The start of the text selection.

selLength
The length of the text range.

Discussion
This method is similar to `editWithFrame:inView:editor:delegate:event:` (page 30), except that it can be invoked in any situation, not only on a mouse-down event. This method returns without doing anything if *controlView*, *textObj*, or the receiver is `nil`, or if the receiver has no font set for it.

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

Declared In
`NSCell.h`

sendActionOn:

Sets the conditions on which the receiver sends action messages to its target.

```
- (NSInteger)sendActionOn:(NSInteger)mask
```

Parameters

mask

A bit mask containing the conditions for sending the action. The only conditions that are actually checked are associated with the `NSLeftMouseDownMask`, `NSLeftMouseUpMask`, `NSLeftMouseDraggedMask`, and `NSPeriodicMask` bits.

Return Value
A bit mask containing the previous settings. This bit mask uses the same values as specified in the *mask* parameter.

Discussion
You use this method during mouse tracking when the mouse button changes state, the mouse moves, or if the cell is marked to send its action continuously while tracking. Because of this, the only bits checked in *mask* are `NSLeftMouseDownMask`, `NSLeftMouseUpMask`, `NSLeftMouseDraggedMask`, and `NSPeriodicMask`, which are declared in the `NSEvent` class reference.

You can use the `setContinuous:` (page 56) method to turn on the flag corresponding to `NSPeriodicMask` or `NSLeftMouseDraggedMask`, whichever is appropriate to the given subclass of `NSCell`.

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

See Also
`- action` (page 20)

Declared In
`NSCell.h`

sendsActionOnEndEditing

Returns a Boolean value that indicates whether the receiver's `NSControl` object sends its action message whenever the user finishes editing the cell's text.

```
- (BOOL)sendsActionOnEndEditing
```

Return Value
`YES` if the receiver's control sends its action message when editing is complete, otherwise `NO`.

Discussion
If this method returns `YES`, the receiver's `NSControl` object sends its action message when the user does one of the following:

- Presses the Return key
- Presses the Tab key to move out of the field
- Clicks another text field

If it returns `NO`, the cell's `NSControl` object sends its action message only when the user presses the Return key.

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

See Also
`- setSendsActionOnEndEditing:` (page 68)

Declared In
`NSCell.h`

setAction:

Sets the cell's action method to the specified selector.

```
- (void)setAction:(SEL)aSelector
```

Parameters

aSelector

The new action-message selector to associate with the receiver's cell. Specify `NULL` to prevent action messages from being sent to the receiver's target.

Discussion
The `NSCell` implementation of this method raises `NSInternalInconsistencyException`. Subclasses (such as `NSActionCell`) override this method to set the action method as part of the target/action implementation.

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

See Also
`- action` (page 20)
`- setTarget:` (page 71)
`- target` (page 81)

Declared In
`NSCell.h`

setAlignment:

Sets the alignment of text in the receiver.

```
- (void)setAlignment:(NSTextAlignment)mode
```

Parameters

mode

This value can be one of the following constants: `NSLeftTextAlignment`, `NSRightTextAlignment`,`NSCenterTextAlignment`, `NSJustifiedTextAlignment`, or`NSNaturalTextAlignment`.

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

See Also
- `alignment` (page 21)
- `setWraps:` (page 74)

Declared In
`NSCell.h`

setAllowsEditingTextAttributes:

Sets whether the receiver allows the user to edit textual attributes of its contents.

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

Parameters

flag

If `YES`, the user can modify the font and other textual attributes of the cell's text. If `NO`, the user cannot edit the text or import graphics, which effectively means the cell cannot support RTFD text.

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

See Also
- `allowsEditingTextAttributes` (page 21)
- `setImportsGraphics:` (page 63)

Declared In
`NSCell.h`

setAllowsMixedState:

Sets whether the receiver supports three states or just two.

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

Parameters

flag

If `YES`, the receiver supports three states (on, off, and mixed); otherwise, if `NO`, the receiver supports only two states (on and off).

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

See Also
- `allowsMixedState` (page 21)
- `nextState` (page 46)
- `setNextState` (page 66)

Declared In
`NSCell.h`

setAllowsUndo:

Sets whether the receiver assumes responsibility for undo operations within the cell.

```
- (void)setAllowsUndo:(BOOL)allowsUndo
```

Parameters

allowsUndo

If `YES`, the receiver handles undo operations; otherwise, if `NO`, the application's custom undo manager handles undo operations.

Discussion
Subclasses invoke this method to indicate their preference for handling undo operations; otherwise, you should not need to call this method directly.

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

See Also
- `allowsUndo` (page 22)

Declared In
`NSCell.h`

setAttributedStringValue:

Sets the value of the receiver's cell using an attributed string.

```
- (void)setAttributedStringValue:(NSAttributedString *)attribStr
```

Parameters

attribStr

The value of the cell interpreted as an attributed string.

Discussion
If a formatter is set for the receiver, but the formatter does not understand the attributed string, it marks *attribStr* as an invalid object. If the receiver is not a text-type cell, it is converted to one before the value is set.

For Mac OS X v10.3 and later: If you use a class that responds to the selector `attributedStringValue` (page 22) for the object value of a cell, then the cell uses that method to fetch the string to draw rather than using `stringValue` (page 77).

The following example sets the text in a cell to 14 points, red, in the system font.

```
NSColor *txtColor = [NSColor redColor];
NSFont *txtFont = [NSFont boldSystemFontOfSize:14];
NSDictionary *txtDict = [NSDictionary dictionaryWithObjectsAndKeys:
        txtFont, NSFontAttributeName, txtColor, NSForegroundColorAttributeName,
 nil];
NSAttributedString *attrStr = [[[NSAttributedString alloc]
        initWithString:@"Hello!" attributes:txtDict] autorelease];
[[attrStrTextField cell] setAttributedStringValue:attrStr];
[attrStrTextField updateCell:[attrStrTextField cell]];
```

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

See Also
`- attributedStringValue` (page 22)

Declared In
`NSCell.h`

setBackgroundStyle:

Sets the background style for the receiver.

```
- (void)setBackgroundStyle:(NSBackgroundStyle)style
```

Parameters

style
The background style for the receiver.

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

Declared In
`NSCell.h`

setBaseWritingDirection:

Sets the initial writing direction used to determine the actual writing direction for text .

```
- (void)setBaseWritingDirection:(NSWritingDirection)writingDirection
```

Parameters

writingDirection
One of the following values: `NSWritingDirectionNatural`, `NSWritingDirectionLeftToRight`, or `NSWritingDirectionRightToLeft`.

Discussion
If you know the base writing direction of the text you are rendering, you can use this method to specify that direction to the text system.

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

See Also
- `baseWritingDirection` (page 23)

Declared In
`NSCell.h`

setBezeled:

Sets whether the receiver draws itself with a bezeled border.

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

Parameters

flag

If `YES`, the receiver uses a bezeled border.

Discussion
The `setBezeled:` and `setBordered:` (page 55) methods are mutually exclusive (that is, a border can be only plain or bezeled). Invoking this method automatically removes any border that had already been set, regardless of the value in the `flag` parameter.

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

See Also
- `isBezeled` (page 40)

Declared In
`NSCell.h`

setBordered:

Sets whether the receiver draws itself outlined with a plain border.

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

Parameters

flag

If `YES`, the receiver uses a plain border.

Discussion
The `setBezeled:` (page 55) and `setBordered:` methods are mutually exclusive (that is, a border can be only plain or bezeled). Invoking this method automatically removes any bezel that had already been set, regardless of the value in the `flag` parameter.

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

See Also
- `isBordered` (page 40)

Declared In
`NSCell.h`

setCellAttribute:to:

Sets the value for the specified cell attribute.

```
- (void)setCellAttribute:(NSCellAttribute)aParameter to:(NSInteger)value
```

Parameters

aParameter

The cell attribute whose value you want to set. Attributes include the receiver's current state and whether it is disabled, editable, or highlighted.

value

The new value for the attribute.

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

See Also
- cellAttribute: (page 24)

Declared In
NSCell.h

setContinuous:

Sets whether the receiver's cell sends its action message continuously to its target during mouse tracking.

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

Parameters

flag

If YES, the action message should be sent continuously.

Discussion
In practice, the continuous setting of action messages has meaning only for NSActionCell and its subclasses, which implement the target/action mechanism. Some NSControl subclasses, notably NSMatrix, send a default action to a default target when a cell doesn't provide a target or action.

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

See Also
- isContinuous (page 40)
- sendActionOn: (page 50)

Declared In
NSCell.h

setControlSize:

Sets the size of the receiver.

```
- (void)setControlSize:(NSControlSize)size
```

Parameters

size

A value that specifies the size of the receiver (for possible values, see "Control Sizes" (page 94)).

Discussion

Changing the cell's control size does not change the font of the cell. Use the `systemFontSizeForControlSize:` class method of `NSFont` to obtain the system font based on the new control size and set it.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `controlSize` (page 27)

Related Sample Code

QTAudioExtractionPanel

Quartz Composer WWDC 2005 TextEdit

TextEditPlus

Declared In

`NSCell.h`

setControlTint:

Sets the receiver's control tint.

```
- (void)setControlTint:(NSControlTint)controlTint
```

Parameters

controlTint

An `NSControlTint` (page 93) value that specifies the tint of the receiver (see "Control Tints" (page 93) for possible values).

Availability

Available in Mac OS X v10.0 and later.

See Also

- `controlTint` (page 27)

Declared In

`NSCell.h`

setControlView:

Sets the receiver's control view.

```
- (void)setControlView:(NSView *)view
```

Parameters

view

The view (normally an `NSControl` object) to associate with the cell.

Discussion
The control view represents the control currently being rendered by the cell.

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

See Also
- `controlView` (page 27)

Declared In
`NSCell.h`

setDoubleValue:

Sets the value of the receiver's cell using a double-precision floating-point number.

```
- (void)setDoubleValue:(double)aDouble
```

Parameters

aDouble
The value of the cell interpreted as a double-precision floating-point number.

Discussion
In its implementation, this method invokes the `setObjectValue:` (page 66) method to set the actual value. This method does nothing if the receiver is not a text-type cell.

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

See Also
- `doubleValue` (page 28)

Declared In
`NSCell.h`

setEditable:

Sets whether the user can edit the receiver's text.

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

Parameters

flag
If `YES`, the user is allowed to edit the receiver's text. If this value is `YES`, the text is also made selectable. If it is `NO`, the selectable attribute is restored to the value it was before the cell was last made editable.

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

See Also
- `isEditable` (page 41)
- `setSelectable:` (page 68)

Related Sample Code
EnhancedDataBurn
ImageBackground
QTAudioExtractionPanel
QTKitMovieShuffler

Declared In
NSCell.h

setEnabled:

Sets whether the receiver is enabled or disabled.

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

Parameters

flag
If YES the receiver is enabled; otherwise, if NO, the receiver is disabled.

Discussion
The text of disabled cells is changed to gray. If a cell is disabled, it cannot be highlighted, does not support mouse tracking (and thus cannot participate in target/action functionality), and cannot be edited. However, you can still alter many attributes of a disabled cell programmatically. (The setState: (page 69) method, for instance, still works.)

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

See Also
- isEnabled (page 41)

Related Sample Code
QTKitMovieShuffler
TextLinks
VertexPerformanceTest

Declared In
NSCell.h

setFloatingPointFormat:left:right:

Sets the autoranging and floating point number format of the receiver's cell.

```
- (void)setFloatingPointFormat:(BOOL)autoRange left:(NSUInteger)leftDigits
    right:(NSUInteger)rightDigits
```

Parameters

autoRange
If YES, autoranging is enabled, otherwise it is disabled.

leftDigits
The number of digits to display to the left of the decimal point.

rightDigits

The number of digits to display to the right of the decimal point.

Discussion

Sets whether floating-point numbers are autoranged in the receiver and sets the sizes of the fields to the left and right of the decimal point. If *autoRange* is NO, *leftDigits* specifies the maximum number of digits to the left of the decimal point, and *rightDigits* specifies the number of digits to the right (the fractional digit places will be padded with zeros to fill this width). However, if a number is too large to fit its integer part in *leftDigits* digits, as many places as are needed on the left are effectively removed from *rightDigits* when the number is displayed.

If *autoRange* is YES, *leftDigits* and *rightDigits* are simply added to form a maximum total field width for the receiver (plus 1 for the decimal point). The fractional part will be padded with zeros on the right to fill this width, or truncated as much as possible (up to removing the decimal point and displaying the number as an integer). The integer portion of a number is never truncated—that is, it is displayed in full no matter what the field width limit is.

The following example sets a cell used to display dollar amounts up to 99,999.99:

```
[[currencyDollarsField cell] setEntryType:NSFloatType];
[[currencyDollarsField cell] setFloatingPointFormat:NO left:5  right:2];
```

Note: This method is being deprecated in favor of a new class of formatter objects. For more information, see NSFormatter. This documentation is provided only for developers who need to modify older applications.

Availability

Available in Mac OS X v10.0 and later.

See Also

- setEntryType: (page 99)

Declared In

NSCell.h

setFloatValue:

Sets the value of the receiver's cell using a single-precision floating-point number.

```
- (void)setFloatValue:(float)aFloat
```

Parameters

aFloat

The value of the cell interpreted as a single-precision floating-point number.

Discussion

In its implementation, this method invokes the setObjectValue: (page 66) method to set the actual value. This method does nothing if the receiver is not a text-type cell.

Availability

Available in Mac OS X v10.0 and later.

See Also

- floatValue (page 32)

Declared In
NSCell.h

setFocusRingType:

Sets the type of focus ring to be used.

```
- (void)setFocusRingType:(NSFocusRingType)focusRingType
```

Parameters

focusRingType

Possible values are listed in NSFocusRingType. To disable a view's focus ring, specify NSFocusRingTypeNone.

Discussion
You should only disable a view from drawing its focus ring if you want to draw your own focus ring, or if there is not sufficient space to display a focus ring in the default location.

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

See Also
- focusRingType (page 32)
+ defaultFocusRingType (page 19)

Declared In
NSCell.h

setFont:

Sets the font to use when the receiver displays text.

```
- (void)setFont:(NSFont *)fontObj
```

Parameters

fontObj

The font to use.

Discussion
If the receiver is not a text-type cell, the method converts it to that type before setting the font.

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

See Also
- font (page 33)

Related Sample Code
Mountains

Declared In
NSCell.h

setFormatter:

Sets the receiver's formatter object.

```
- (void)setFormatter:(NSFormatter *)newFormatter
```

Parameters

newFormatter

The formatter to use with the cell, or `nil` if you do not want the cell to use a formatter.

Discussion

Cells use a formatter object to format the textual representation of their object value and to validate cell input and convert that input to an object value. If the new formatter cannot interpret the receiver's current object value, that value is converted to a string object.

Availability

Available in Mac OS X v10.0 and later.

See Also

- formatter (page 33)

Declared In

NSCell.h

setHighlighted:

Sets whether the receiver has a highlighted appearance.

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

Parameters

flag

If YES, the receiver has a highlight.

Discussion

By default, this method does nothing. The `NSButtonCell` class overrides this method to draw the button with the appearance specified by `NSCellLightsByBackground`, `NSCellLightsByContents`, or `NSCellLightsByGray`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

NSCell.h

setImage:

Sets the image to be displayed by the receiver.

```
- (void)setImage:(NSImage *)image
```

Parameters

image

The image to display in the cell.

Discussion
If the receiver is not an image-type cell, the method converts it to that type of cell.

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

See Also
- `image` (page 36)
- `setType:` (page 73)

Related Sample Code
EnhancedDataBurn
ImageBackground
QTKitMovieShuffler
Transformed Image

Declared In
`NSCell.h`

setImportsGraphics:

Sets whether the receiver can import images into its text.

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

Parameters

flag

If `YES`, the receiver can import images into its text and support RTFD text. If `NO`, RTFD text is not supported.

Discussion
If *flag* is `YES`, the receiver is also set to allow editing of text attributes.

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

See Also
- `importsGraphics` (page 37)
- `setAllowsEditingTextAttributes:` (page 52)

Declared In
`NSCell.h`

setIntegerValue:

Sets the value of the receiver using an `NSInteger`.

```
- (void)setIntegerValue:(NSInteger)anInteger
```

Parameters

anInteger

The value of the cell interpreted as an `NSInteger`.

Discussion
In its implementation, this method invokes the `setObjectValue:` (page 66) method to set the actual value. This method does nothing if the receiver is not a text-type cell.

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

See Also
- `integerValue` (page 38)
- `setIntValue:` (page 64)

Declared In
`NSCell.h`

setIntValue:

Sets the value of the receiver using an integer.

```
- (void)setIntValue:(int)anInt
```

Parameters
anInt
The value of the cell interpreted as an integer.

Discussion
In its implementation, this method invokes the `setObjectValue:` (page 66) method to set the actual value. This method does nothing if the receiver is not a text-type cell.

On Mac OS X v10.5 and later, you should use `setIntegerValue:` (page 63) instead.

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

See Also
- `intValue` (page 39)
- `setIntegerValue:` (page 63)

Declared In
`NSCell.h`

setLineBreakMode:

Sets the line break mode to use when drawing text

```
- (void)setLineBreakMode:(NSLineBreakMode)mode
```

Parameters
mode
The desired line break mode, which should be one of the following constants: `NSLineBreakByWordWrapping`, `NSLineBreakByCharWrapping`, `NSLineBreakByClipping`, `NSLineBreakByTruncatingHead`, `NSLineBreakByTruncatingTail`, or `NSLineBreakByTruncatingMiddle`.

Discussion
The line break mode can also be modified by calling the `setWraps:` (page 74) method.

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

See Also
- `lineBreakMode` (page 44)
- `setWraps:` (page 74)

Declared In
`NSCell.h`

setMenu:

Sets the contextual menu for the cell.

```
- (void)setMenu:(NSMenu *)aMenu
```

Parameters

aMenu

A menu that has commands contextually related to the receiver. Specify `nil` to clear the previous menu.

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

See Also
- `menu` (page 44)

Declared In
`NSCell.h`

setMnemonicLocation:

Sets the character of the receiver's title to be used as a mnemonic character.

```
- (void)setMnemonicLocation:(NSUInteger)location
```

Parameters

location

The zero-based index into the cell's title string specifying the location of the mnemonic character. The specified character is underlined when the title is drawn.

Discussion
Mnemonics are not supported in Mac OS X.

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

See Also
- `mnemonicLocation` (page 45)

Declared In
NSCell.h

setNextState

Changes the state of the receiver to its next state.

```
- (void)setNextState
```

Discussion
If the receiver has three states, it cycles through them in this order: on, off, mixed, on, and so forth. If the receiver has two states, it toggles between them.

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

See Also
- `allowsMixedState` (page 21)
- `nextState` (page 46)
- `setAllowsMixedState:` (page 52)

Declared In
NSCell.h

setObjectValue:

Sets the receiver's object value.

```
- (void)setObjectValue:(id < NSCopying >)object
```

Parameters
object
The new object value for the cell.

Discussion
To be valid object value, the receiver must have a formatter capable of converting the object to and from its textual representation.

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

See Also
- `objectValue` (page 47)
- `setRepresentedObject:` (page 67)

Declared In
NSCell.h

setRefusesFirstResponder:

Sets whether the receiver should not become the first responder.

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

Parameters

flag

If YES, the receiver should never become the first responder; otherwise, it may become the first responder.

Discussion

If refusesFirstResponder (page 48) returns NO and the cell is enabled, the method acceptsFirstResponder (page 20) returns YES, allowing the cell to become first responder.

Availability

Available in Mac OS X v10.0 and later.

Declared In

NSCell.h

setRepresentedObject:

Sets the object represented by the receiver.

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

Parameters

anObject

The object to associate with the receiver.

Discussion

You can use this method to link two objects together. For example, if the receiver's title was "Blue", you could associate an NSColor object whose color was set to blue.

Special Considerations

Note that if you copy an NSCell instance, the represented object in the copy is set to nil.

Availability

Available in Mac OS X v10.0 and later.

See Also

- setObjectValue: (page 66)
- representedObject (page 48)

Declared In

NSCell.h

setScrollable:

Sets whether excess text in the receiver is scrolled past the cell's bounds.

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

Parameters

flag

If YES, text can be scrolled past the cell's bounds; otherwise, if NO, the text wrapping is enabled.

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

See Also
- `isScrollable` (page 43)

Declared In
`NSCell.h`

setSelectable:

Sets whether text in the receiver can be selected.

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

Parameters

flag

If `YES`, the receiver's text can be selected. If this value is `NO`, editability is also disabled; if it is `YES`, editability is not affected.

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

See Also
- `isSelectable` (page 43)
- `setEditable:` (page 58)

Declared In
`NSCell.h`

setSendsActionOnEndEditing:

Sets whether the receiver's NSControl object sends its action message whenever the user finishes editing the cell's text.

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

Parameters

flag

If `YES`, the receiver's control sends its action message when editing is complete; otherwise, if `NO`, it sends the action message only when the user presses the Return key.

Discussion
If *flag* is `YES`, the receiver's `NSControl` object sends its action message when the user does one of the following:

- Presses the Return key
- Presses the Tab key to move out of the field
- Clicks another text field

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

See Also
- `sendsActionOnEndEditing` (page 50)

Related Sample Code
Quartz Composer QCTV

Declared In
`NSCell.h`

setShowsFirstResponder:

Sets whether the receiver draws some indication of its first responder status.

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

Parameters

flag

If `YES`, the receiver draws an indication of its first responder status, otherwise it does not.

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

See Also
- `showsFirstResponder` (page 75)

Declared In
`NSCell.h`

setState:

Sets the receiver's state to the specified value.

```
- (void)setState:(NSInteger)value
```

Parameters

value

The possible state values are `NSOnState`, `NSOffState`, and `NSMixedState`. If the cell supports only two states and you specify `NSMixedState`, this method sets the state to `NSOnState`.

Discussion
The `NSOffState` state indicates the normal or unpressed state. The `NSOnState` state indicates the alternate or pressed state. The NSMixedState state indicates that the feature represented by the control is in effect somewhere.

Although using the enumerated constants is preferred, *value* can also be an integer. If the cell has two states, 0 is treated as `NSOffState`, and a nonzero value is treated as `NSOnState`. If the cell has three states, 0 is treated as `NSOffState`; a negative value, as `NSMixedState`; and a positive value, as `NSOnState`.

Note that the value `state` (page 76) returns may not be the same value you passed into the `value` parameter.

To check whether the cell has three states (and uses the mixed state), invoke the `allowsMixedState` (page 21) method.

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

See Also
- `state` (page 76)
- `setAllowsMixedState:` (page 52)

Related Sample Code
EnhancedDataBurn

Declared In
`NSCell.h`

setStringValue:

Sets the value of the receiver's cell using an `NSString` object.

```
- (void)setStringValue:(NSString *)aString
```

Parameters

aString
The string value of the cell.

Discussion
In its implementation, this method invokes the `setObjectValue:` (page 66) method to set the actual value. If no formatter is assigned to the receiver or if the formatter cannot "translate" *aString* to an underlying object, the receiver is flagged as having an invalid object. If the receiver is not a text-type cell, this method converts it to one before setting the object value.

For Mac OS X v10.3 and later: If you use a class that responds to the selector `attributedStringValue` (page 22) for the object value of a cell, the cell uses that method to fetch the string to draw rather than the `stringValue` (page 77) method.

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

See Also
- `stringValue` (page 77)

Related Sample Code
QTKitMovieShuffler

Declared In
`NSCell.h`

setTag:

Sets the tag of the receiver.

```
- (void)setTag:(NSInteger)anInteger
```

Parameters

anInteger

The new tag for the cell.

Discussion

The `NSCell` implementation of this method raises `NSInternalInconsistencyException`. The `NSActionCell` implementation sets the receiver's tag integer to *anInteger*.

Tags allow you to identify particular cells. Tag values are not used internally; they are only changed by external invocations of `setTag:`. You typically set tag values in Interface Builder and use them at runtime in your application. When you set the tag of a control with a single cell in Interface Builder, it sets the tags of both the control and the cell to the same value as a convenience.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- tag` (page 78)

Related Sample Code

Quartz Composer WWDC 2005 TextEdit

TextEditPlus

Declared In

`NSCell.h`

setTarget:

Sets the target object to receive action messages.

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

Parameters

anObject

The new target object to associate with the receiver's cell, or `nil` to remove the current target.

Discussion

The `NSCell` implementation of this method raises `NSInternalInconsistencyException`. Subclasses (such as `NSActionCell`) override this method to set the target object as part of the target/action implementation.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- target` (page 81)

Declared In

`NSCell.h`

setTitle:

Sets the title of the receiver.

```
- (void)setTitle:(NSString *)aString
```

Parameters

aString

The new string value for the cell.

Availability

Available in Mac OS X v10.0 and later.

See Also

- title (page 81)

Related Sample Code

ObjectPath

Quartz Composer WWDC 2005 TextEdit

TextEditPlus

TextLinks

Declared In

NSCell.h

setTitleWithMnemonic:

Sets the title of the receiver with one character in the string denoted as an access key.

```
- (void)setTitleWithMnemonic:(NSString *)aString
```

Parameters

aString

The new title of the cell. One character in the string should be preceded by an ampersand (&) character. The character that follows becomes the mnemonic character for the title.

Discussion

Mnemonics are not supported in Mac OS X.

Availability

Available in Mac OS X v10.0 and later.

See Also

- mnemonic (page 45)
- setMnemonicLocation: (page 65)

Declared In

NSCell.h

setTruncatesLastVisibleLine:

Sets whether the receiver truncates and adds the ellipsis character to the last visible line if the text doesn't fit into the cell bounds.

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

Parameters

flag

If YES, the receiver truncates the last line; if NO, it does not truncate.

Discussion

The line break mode must be either NSLineBreakByWordWrapping or NSLineBreakByCharWrapping. Otherwise, this setting is ignored.

Availability

Available in Mac OS X v10.5 and later.

See Also

- truncatesLastVisibleLine (page 83)
- setLineBreakMode: (page 64)

Declared In

NSCell.h

setType:

Sets the type of the cell, changing it to a text cell, image cell, or null cell.

```
- (void)setType:(NSCellType)aType
```

Parameters

aType

The new type of the cell (see "Cell Types" (page 85) for possible values).

Discussion

If the cell is already the same type as the one specified in the *aType* parameter, this method does nothing.

If *aType* is NSTextCellType, this method converts the receiver to a cell of that type, giving it a default title and setting the font to the system font at the default size. If *aType* is NSImageCellType, the cell type is not changed until you set a new non-nil image.

Availability

Available in Mac OS X v10.0 and later.

See Also

- type (page 83)
- setImage: (page 62)

Declared In

NSCell.h

setUpFieldEditorAttributes:

Configures the textual and background attributes of the receiver's field editor.

```
- (NSText *)setUpFieldEditorAttributes:(NSText *)textObj
```

Parameters

textObj

The field editor to configure. .

Return Value

The configured field editor.

Discussion

If the receiver is disabled, this method sets the text color to dark gray; otherwise the method sets it to the default color. If the receiver has a bezeled border, this method sets the background to the default color for text backgrounds; otherwise, the method sets it to the color of the receiver's `NSControl` object.

You should not use this method to substitute a new field editor. `setUpFieldEditorAttributes:` (page 73) is intended to modify the attributes of the text object (that is, the field editor) passed into it and return that text object. If you want to substitute your own field editor, use the `fieldEditor:forObject:` method or the `windowWillReturnFieldEditor:toObject:` delegate method of `NSWindow`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSCell.h`

setWraps:

Sets whether text in the receiver wraps when its length exceeds the frame of the cell.

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

Parameters

flag

If `YES`, the receiver wraps text and also makes the receiver nonscrollable; otherwise, if `NO`, text is not wrapped.

Discussion

If the text of the receiver is an attributed string value you must explicitly set the paragraph style line break mode. Calling this method with the value `YES` is equivalent to calling the `setLineBreakMode:` method with the value `NSLineBreakByWordWrapping`.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `setLineBreakMode:` (page 64)
- `wraps` (page 84)

Related Sample Code

Quartz Composer QCTV

Declared In

`NSCell.h`

showsFirstResponder

Returns a Boolean value that indicates whether the receiver should draw some indication of its first responder status.

```
- (BOOL)showsFirstResponder
```

Return Value
`YES` if the receiver should draw an indication of its first responder status, otherwise `NO`.

Discussion
The `NSCell` class itself does not draw a first-responder indicator. Subclasses may use the returned value to determine whether or not they should draw one, however.

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

See Also
`- setShowsFirstResponder:` (page 69)

Related Sample Code
Clock Control

Declared In
`NSCell.h`

startTrackingAt:inView:

Begins tracking mouse events within the receiver.

```
- (BOOL)startTrackingAt:(NSPoint)startPoint inView:(NSView *)controlView
```

Parameters
startPoint
The initial location of the cursor.

controlView
The `NSControl` object managing the receiver.

Return Value
`YES` if the receiver is set to respond continuously or set to respond when the mouse is dragged, otherwise `NO`.

Discussion
The `NSCell` implementation of `trackMouse:inRect:ofView:untilMouseUp:` (page 82) invokes this method when tracking begins. Subclasses can override this method to implement special mouse-tracking behavior at the beginning of mouse tracking—for example, displaying a special cursor.

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

See Also
`- continueTracking:at:inView:` (page 26)
`- stopTracking:at:inView:mouseIsUp:` (page 76)

Declared In
NSCell.h

state

Returns the receiver's state.

```
- (NSInteger)state
```

Return Value
The receiver's state (for possible values, see "Cell States" (page 91)).

Discussion
Cells can have two or three states. If the receiver has two states, it returns either NSOffState (the normal or unpressed state) or NSOnState (the alternate or pressed state). If it has three, it may also return NSMixedState, indicating the feature is in effect somewhere.

To check whether the receiver uses the mixed state, use the method allowsMixedState (page 21).

Note that the value state (page 76) returns may not be the same value you passed into setState: (page 69).

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

See Also
- setState: (page 69)
- setAllowsMixedState: (page 52)

Declared In
NSCell.h

stopTracking:at:inView:mouseIsUp:

Stops tracking mouse events within the receiver.

```
- (void)stopTracking:(NSPoint)lastPoint at:(NSPoint)stopPoint inView:(NSView
    *)controlView mouseIsUp:(BOOL)flag
```

Parameters

lastPoint
Contains the previous position of the cursor.

stopPoint
The current location of the cursor.

controlView
The NSControl object managing the receiver.

flag
If YES, this method was invoked because the user released the mouse button; otherwise, if NO, the cursor left the designated tracking rectangle.

Discussion
The default `NSCell` implementation of `trackMouse:inRect:ofView:untilMouseUp:` (page 82) invokes this method when the cursor has left the bounds of the receiver or the mouse button goes up. The default `NSCell` implementation of this method does nothing. Subclasses often override this method to provide customized tracking behavior. The following example increments the state of a tristate cell when the mouse button is clicked:

```
- (void)stopTracking:(NSPoint)lastPoint at:(NSPoint)stopPoint
    inView:(NSView *)controlView mouseIsUp:(BOOL)flag
{
    if (flag == YES) {
        [self setTriState:([self triState]+1)];
    }
}
```

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

See Also
- `startTrackingAt:inView:` (page 75)
- `continueTracking:at:inView:` (page 26)

Declared In
`NSCell.h`

stringValue

Returns the value of the receiver's cell as an `NSString` object.

```
- (NSString *)stringValue
```

Return Value
The string value of the cell. This value may be an interpreted version of the cell's actual value. Interpretations are performed by the cell's formatter.

Discussion
If no formatter exists and the cell's value is an `NSString` object, this method returns the value as a plain, attributed, or localized formatted string. If the value is not an `NSString` object or cannot be converted to one, this method returns an empty string.

For Mac OS X v10.3 and later: If you use a class that responds to the selector `attributedStringValue` (page 22) for the object value of a cell, the cell uses that method to fetch the string to draw rather than the `stringValue` method.

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

See Also
- `setStringValue:` (page 70)

Related Sample Code
QTKitMovieShuffler

Declared In
NSCell.h

tag

Returns the tag identifying the receiver.

```
- (NSInteger)tag
```

Return Value
The tag value. The NSCell implementation of this method returns -1.

Discussion
Tags allow you to identify particular cells. Tag values are not used internally; they are only changed by external invocations of setTag:. You typically set tag values in Interface Builder and use them at runtime in your application. When you set the tag of a control with a single cell in Interface Builder, it sets the tags of both the control and the cell to the same value as a convenience.

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

See Also
- setTag: (page 70)

Related Sample Code
EnhancedDataBurn
MyCustomColorPicker
Quartz Composer WWDC 2005 TextEdit
TextEditPlus
VertexPerformanceTest

Declared In
NSCell.h

takeDoubleValueFrom:

Sets the value of the receiver's cell to a double-precision floating-point value obtained from the specified object.

```
- (void)takeDoubleValueFrom:(id)sender
```

Parameters

sender
The object from which to take the value. This object must respond to the doubleValue (page 28) message.

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

See Also
- setDoubleValue: (page 58)

Declared In
NSCell.h

takeFloatValueFrom:

Sets the value of the receiver's cell to a single-precision floating-point value obtained from the specified object.

```
- (void)takeFloatValueFrom:(id)sender
```

Parameters

sender

The object from which to take the value. This object must respond to the `floatValue` (page 32) message.

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

See Also
- `setFloatValue:` (page 60)

Declared In
NSCell.h

takeIntegerValueFrom:

Sets the value of the receiver's cell to an integer value obtained from the specified object.

```
- (void)takeIntegerValueFrom:(id)sender
```

Parameters

sender

The object from which to take the value. This object must respond to the `integerValue` (page 38) message.

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

See Also
- `setIntValue:` (page 64)
- `setIntegerValue:` (page 63)

Declared In
NSCell.h

takeIntValueFrom:

Sets the value of the receiver's cell to an integer value obtained from the specified object.

```
- (void)takeIntValueFrom:(id)sender
```

Parameters

sender

The object from which to take the value. This object must respond to the `intValue` (page 39) message.

Discussion

On Mac OS X v10.5 and later you should use `takeIntegerValueFrom:` (page 79) instead.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `takeIntegerValueFrom:` (page 79)
- `setIntValue:` (page 64)
- `setIntegerValue:` (page 63)

Declared In

`NSCell.h`

takeObjectValueFrom:

Sets the value of the receiver's cell to the object value obtained from the specified object.

```
- (void)takeObjectValueFrom:(id)sender
```

Parameters

sender

The object from which to take the value. This object must respond to the `objectValue` (page 47) message.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `setObjectValue:` (page 66)

Declared In

`NSCell.h`

takeStringValueFrom:

Sets the value of the receiver's cell to the string value obtained from the specified object.

```
- (void)takeStringValueFrom:(id)sender
```

Parameters

sender

The object from which to take the value. This object must respond to the `stringValue` (page 77) message.

Availability

Available in Mac OS X v10.0 and later.

See Also
- `setStringValue:` (page 70)

Declared In
`NSCell.h`

target

Returns the target object of the receiver.

```
- (id)target
```

Return Value
The target object that receives action messages from the cell. The `NSCell` implementation of this method returns `nil`.

Discussion
Subclasses (such as `NSActionCell`) override this method to return the target object as part of the target/action implementation.

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

See Also
- `setTarget:` (page 71)

Declared In
`NSCell.h`

title

Returns the receiver's title.

```
- (NSString *)title
```

Return Value
The cell's string value.

Discussion
Subclasses (such as `NSButtonCell`) may override this method to return a different value.

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

See Also
- `setTitle:` (page 71)

Declared In
`NSCell.h`

titleRectForBounds:

Returns the rectangle in which the receiver draws its title text.

```
- (NSRect)titleRectForBounds:(NSRect)theRect
```

Parameters

theRect

The bounding rectangle of the receiver.

Return Value

The rectangle in which the receiver draws its title text.

Discussion

If the receiver is a text-type cell, this method resizes the drawing rectangle for the title (*theRect*) inward by a small offset to accommodate the cell border. If the receiver is not a text-type cell, the method does nothing.

Availability

Available in Mac OS X v10.0 and later.

See Also

- imageRectForBounds: (page 37)

Declared In

NSCell.h

trackMouse:inRect:ofView:untilMouseUp:

Initiates the mouse tracking behavior in a cell.

```
- (BOOL)trackMouse:(NSEvent *)theEvent inRect:(NSRect)cellFrame ofView:(NSView
    *)controlView untilMouseUp:(BOOL)untilMouseUp
```

Parameters

theEvent

The event that caused the mouse tracking to occur.

cellFrame

The receiver's frame rectangle.

controlView

The view containing the receiver. This is usually an NSControl object.

untilMouseUp

If YES, mouse tracking continues until the user releases the mouse button; otherwise, if NO, tracking continues until the cursor leaves the tracking rectangle, specified by the *cellFrame* parameter, regardless of the mouse button state. See the discussion for more information.

Return Value

YES if the mouse tracking conditions are met, otherwise NO.

Discussion

This method is generally not overridden because the default implementation invokes other NSCell methods that can be overridden to handle specific events in a dragging session. This method's return value depends on the *untilMouseUp* flag. If *untilMouseUp* is set to YES, this method returns YES if the mouse button goes up while the cursor is anywhere; NO, otherwise. If *untilMouseUp* is set to NO, this method returns YES if the mouse button goes up while the cursor is within *cellFrame*; NO, otherwise.

This method first invokes `startTrackingAt:inView:` (page 75). If that method returns `YES`, then as mouse-dragged events are intercepted, `continueTracking:at:inView:` (page 26) is invoked until either the method returns `NO` or the mouse is released. Finally, `stopTracking:at:inView:mouseIsUp:` (page 76) is invoked if the mouse is released. If *untilMouseUp* is `YES`, it's invoked when the mouse button goes up while the cursor is anywhere. If *untilMouseUp* is `NO`, it's invoked when the mouse button goes up while the cursor is within *cellFrame*. You usually override one or more of these methods to respond to specific mouse events.

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

Declared In
`NSCell.h`

truncatesLastVisibleLine

Returns a Boolean value indicating whether the receiver truncates and adds the ellipsis character to the last visible line if the text doesn't fit into the cell bounds.

```
- (BOOL)truncatesLastVisibleLine
```

Return Value
`YES` if the receiver truncates the last line; otherwise `NO`.

Discussion
The line break mode must be either `NSLineBreakByWordWrapping` or `NSLineBreakByCharWrapping`. Otherwise, this setting is ignored.

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

See Also
- `setTruncatesLastVisibleLine:` (page 72)
- `lineBreakMode` (page 44)

Declared In
`NSCell.h`

type

Returns the type of the receiver

```
- (NSCellType)type
```

Return Value
The type of the cell (see "`Cell Types`" (page 85) for possible values).

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

See Also
- `setType:` (page 73)

Related Sample Code
Clock Control

Declared In
`NSCell.h`

wantsNotificationForMarkedText

Returns a Boolean value that indicates whether the field editor initiated by the receiver should post text change notifications.

```
- (BOOL)wantsNotificationForMarkedText
```

Return Value
`YES` if the field editor initiated by the receiver should post text change notifications (`NSTextDidChangeNotification`) while editing marked text; otherwise, they are delayed until the marked text confirmation.

Discussion
`NSCell`'s implementation returns `NO`.

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

Declared In
`NSCell.h`

wraps

Returns a Boolean value that indicates whether the receiver wraps its text when the text exceeds the borders of the cell.

```
- (BOOL)wraps
```

Return Value
`YES` if the receiver wraps text, otherwise `NO`.

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

See Also
`- setWraps:` (page 74)

Declared In
`NSCell.h`

Constants

NSCellType

Defines a type for cell types.

```
typedef NSUInteger NSCellType;
```

Discussion
For possible values, see "Cell Types" (page 85).

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

Declared In
`NSCell.h`

Cell Types

These constants specify how a cell represents its data (as text or as an image).

```
enum {
    NSNullCellType  = 0,
    NSTextCellType  = 1,
    NSImageCellType = 2
};
```

Constants

`NSNullCellType`
Cell displays nothing.
Available in Mac OS X v10.0 and later.
Declared in `NSCell.h`.

`NSTextCellType`
Cell displays text.
Available in Mac OS X v10.0 and later.
Declared in `NSCell.h`.

`NSImageCellType`
Cell displays images.
Available in Mac OS X v10.0 and later.
Declared in `NSCell.h`.

Discussion
These constants are used by `setType:` (page 73) and `type` (page 83).

Declared In
`NSCell.h`

NSCellAttribute

Defines a type for cell attributes.

```
typedef NSUInteger NSCellAttribute;
```

Discussion
For possible values, see "Cell Attributes" (page 86).

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

Declared In
`NSCell.h`

Cell Attributes

These constants specify how a button behaves when pressed and how it displays its state.

```
typedef enum _NSCellAttribute {
    NSCellDisabled               = 0,
    NSCellState                  = 1,
    NSPushInCell                 = 2,
    NSCellEditable               = 3,
    NSChangeGrayCell             = 4,
    NSCellHighlighted            = 5,
    NSCellLightsByContents       = 6,
    NSCellLightsByGray           = 7,
    NSChangeBackgroundCell       = 8,
    NSCellLightsByBackground     = 9,
    NSCellIsBordered             = 10,
    NSCellHasOverlappingImage    = 11,
    NSCellHasImageHorizontal     = 12,
    NSCellHasImageOnLeftOrBottom = 13,
    NSCellChangesContents        = 14,
    NSCellIsInsetButton          = 15,
    NSCellAllowsMixedState       = 16
} NSCellAttribute;
```

Constants

`NSCellAllowsMixedState`
Lets the cell's state be `NSMixedState`, as well as `NSOffState` and `NSOnState`.
Available in Mac OS X v10.0 and later.
Declared in `NSCell.h`.

`NSChangeBackgroundCell`
If the cell's state is `NSMixedState` or `NSOnState`, changes the cell's background color from gray to white.
Available in Mac OS X v10.0 and later.
Declared in `NSCell.h`.

`NSCellChangesContents`
If the cell's state is `NSMixedState` or `NSOnState`, displays the cell's alternate image.
Available in Mac OS X v10.0 and later.
Declared in `NSCell.h`.

`NSChangeGrayCell`
If the cell's state is `NSMixedState` or `NSOnState`, displays the cell's image as darkened.

Available in Mac OS X v10.0 and later.

Declared in `NSCell.h`.

`NSCellDisabled`
Does not let the user manipulate the cell.

Available in Mac OS X v10.0 and later.

Declared in `NSCell.h`.

`NSCellEditable`
Lets the user edit the cell's contents.

Available in Mac OS X v10.0 and later.

Declared in `NSCell.h`.

`NSCellHasImageHorizontal`
Controls the position of the cell's image: places the image on the right of any text in the cell.

Together, `NSCellHasImageOnLeftOrBottom`, `NSCellHasImageHorizontal`, and `NSCellHasOverlappingImage` control the position of the cell's image and text. To place the image above, set none of them. To place the image below, set `NSCellHasImageOnLeftOrBottom`. To place the image to the right, set `NSCellHasImageHorizontal`. To place the image to the left, set `NSCellHasImageHorizontal` and `NSCellHasImageOnLeftOrBottom`. To place the image directly over, set `NSCellHasOverlappingImage`.

Available in Mac OS X v10.0 and later.

Declared in `NSCell.h`.

`NSCellHasImageOnLeftOrBottom`
Controls the position of the cell's image: places the image on the left of or below any text in the cell.

See `NSCellHasImageHorizontal` (page 87) for more details.

Available in Mac OS X v10.0 and later.

Declared in `NSCell.h`.

`NSCellHasOverlappingImage`
Controls the position of the cell's image: places the image over any text in the cell.

See `NSCellHasImageHorizontal` (page 87) for more details.

Available in Mac OS X v10.0 and later.

Declared in `NSCell.h`.

`NSCellHighlighted`

Draws the cell with a highlighted appearance. (**Deprecated.** Use `setHighlighted:` (page 62) instead.)

Available in Mac OS X v10.0 and later.

Declared in `NSCell.h`.

`NSCellIsBordered`
Draws a border around the cell.

Available in Mac OS X v10.0 and later.

Declared in `NSCell.h`.

`NSCellIsInsetButton`
: Insets the cell's contents from the border.

  By default, the cell's contents are inset by 2 points. This constant is ignored if the cell is unbordered.

  Available in Mac OS X v10.0 and later.

  Declared in `NSCell.h`.

`NSCellLightsByBackground`
: If the cell is pushed in, changes the cell's background color from gray to white.

  Available in Mac OS X v10.0 and later.

  Declared in `NSCell.h`.

`NSCellLightsByContents`
: If the cell is pushed in, displays the cell's alternate image.

  Available in Mac OS X v10.0 and later.

  Declared in `NSCell.h`.

`NSCellLightsByGray`
: If the cell is pushed in, displays the cell's image as darkened.

  Available in Mac OS X v10.0 and later.

  Declared in `NSCell.h`.

`NSPushInCell`
: Determines whether the cell's image and text appear to be shifted down and to the right.

  Available in Mac OS X v10.0 and later.

  Declared in `NSCell.h`.

`NSCellState`
: The cell's state.

  The cell's state can be `NSMixedState`, `NSOffState`, or `NSOnState`.

  Available in Mac OS X v10.0 and later.

  Declared in `NSCell.h`.

Discussion
These constants are used by the `NSButton` and `NSButtonCell` classes:

Declared In
`NSCell.h`

NSCellImagePosition

Defines a type to specify the position of an image in a cell.

```
typedef NSUInteger NSCellImagePosition;
```

Discussion
For possible values, see "Image Position" (page 89).

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

Declared In
`NSCell.h`

Image Position

These constants specify the position of a button's image relative to its title.

```
typedef enum _NSCellImagePosition {
    NSNoImage       = 0,
    NSImageOnly     = 1,
    NSImageLeft     = 2,
    NSImageRight    = 3,
    NSImageBelow    = 4,
    NSImageAbove    = 5,
    NSImageOverlaps = 6
} NSCellImagePosition;
```

Constants

`NSNoImage`

The cell doesn't display an image.

Available in Mac OS X v10.0 and later.

Declared in `NSCell.h`.

`NSImageOnly`

The cell displays an image, but not a title.

Available in Mac OS X v10.0 and later.

Declared in `NSCell.h`.

`NSImageLeft`

The image is to the left of the title.

Available in Mac OS X v10.0 and later.

Declared in `NSCell.h`.

`NSImageRight`

The image is to the right of the title.

Available in Mac OS X v10.0 and later.

Declared in `NSCell.h`.

`NSImageBelow`

The image is below the title.

Available in Mac OS X v10.0 and later.

Declared in `NSCell.h`.

`NSImageAbove`

The image is above the title.

Available in Mac OS X v10.0 and later.

Declared in `NSCell.h`.

`NSImageOverlaps`

The image overlaps the title.

Available in Mac OS X v10.0 and later.

Declared in `NSCell.h`.

Discussion

These constants are used by the `setImagePosition:` and `imagePosition` methods of `NSButton` and `NSButtonCell`.

Declared In
NSCell.h

NSImageScaling

Defines a type to specify the scaling behavior of an image in a cell.

```
typedef NSUInteger NSImageScaling;
```

Discussion
For possible values, see "Image Scaling" (page 90).

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

Declared In
NSCell.h

Image Scaling

These constants specify a cell's image scaling behavior.

```
enum {
    NSImageScaleProportionallyDown = 0,
    NSImageScaleAxesIndependently,
    NSImageScaleNone,
    NSImageScaleProportionallyUpOrDown
};
```

Constants

`NSImageScaleProportionallyDown`
If it is too large for the destination, scale the image down while preserving the aspect ratio.

Available in Mac OS X v10.5 and later.

Declared in NSCell.h.

`NSImageScaleAxesIndependently`
Scale each dimension to exactly fit destination.

This setting does not preserve the aspect ratio of the image.

Available in Mac OS X v10.5 and later.

Declared in NSCell.h.

`NSImageScaleNone`
Do not scale the image.

Available in Mac OS X v10.5 and later.

Declared in NSCell.h.

`NSImageScaleProportionallyUpOrDown`
Scale the image to its maximum possible dimensions while both staying within the destination area and preserving its aspect ratio.

Available in Mac OS X v10.5 and later.

Declared in NSCell.h.

Declared In
NSCell.h

Deprecated Constants

These are deprecated scaling constants. (**Deprecated.** Use "Image Scaling" (page 90) constants instead.)

```
enum {
    NSScaleProportionally = 0,
    NSScaleToFit,
    NSScaleNone
};
```

Constants

`NSScaleProportionally`
Use `NSImageScaleProportionallyDown` (page 90).
Available in Mac OS X v10.0 and later.
Declared in `NSCell.h`.

`NSScaleToFit`
Use `NSImageScaleAxesIndependently` (page 90).
Available in Mac OS X v10.0 and later.
Declared in `NSCell.h`.

`NSScaleNone`
Use `NSImageScaleNone` (page 90).
Available in Mac OS X v10.0 and later.
Declared in `NSCell.h`.

NSCellStateValue

Defines a type to specify the state of a cell.

```
typedef NSUInteger NSCellStateValue;
```

Discussion
For possible values, see "Cell States" (page 91).

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

Declared In
NSCell.h

Cell States

These constants specify a cell's state and are used mostly for buttons.

```
typedef enum _NSCellState {
    NSMixedState = -1,
    NSOffState   = 0,
    NSOnState    = 1
} NSCellStateValue;
```

Constants

`NSMixedState`

The corresponding feature is in effect somewhere.

Available in Mac OS X v10.0 and later.

Declared in `NSCell.h`.

`NSOffState`

The corresponding feature is in effect nowhere.

Available in Mac OS X v10.0 and later.

Declared in `NSCell.h`.

`NSOnState`

The corresponding feature is in effect everywhere.

Available in Mac OS X v10.0 and later.

Declared in `NSCell.h`.

Discussion

These constants are described in Cell States.

Declared In

`NSCell.h`

State Masks

These constants specify what happens when a button is pressed or is displaying its alternate state.

```
enum {
   NSNoCellMask               = 0,
   NSContentsCellMask         = 1,
   NSPushInCellMask           = 2,
   NSChangeGrayCellMask       = 4,
   NSChangeBackgroundCellMask = 8
};
```

Constants

`NSNoCellMask`

The button cell doesn't change.

Available in Mac OS X v10.0 and later.

Declared in `NSCell.h`.

`NSPushInCellMask`

The button cell "pushes in" if it has a border.

Available in Mac OS X v10.0 and later.

Declared in `NSCell.h`.

`NSContentsCellMask`
: The button cell displays its alternate icon and/or title.

  Available in Mac OS X v10.0 and later.

  Declared in `NSCell.h`.

`NSChangeGrayCellMask`
: The button cell swaps the "control color" (the `controlColor` method of `NSColor`) and white pixels on its background and icon.

  Available in Mac OS X v10.0 and later.

  Declared in `NSCell.h`.

`NSChangeBackgroundCellMask`
: Same as `NSChangeGrayCellMask`, but only background pixels are changed.

  Available in Mac OS X v10.0 and later.

  Declared in `NSCell.h`.

Discussion
These contents are used by the `highlightsBy` and `showsStateBy` methods of `NSButtonCell`.

Declared In
`NSCell.h`

NSControlTint

Defines a type to specify the tint of a cell.

```
typedef NSUInteger NSControlTint;
```

Discussion
For possible values, see "Control Tints" (page 93).

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

Declared In
`NSCell.h`

Control Tints

These constants specify a cell's tint.

```
typedef enum _NSControlTint {
    NSDefaultControlTint  = 0,
    NSBlueControlTint     = 1,
    NSGraphiteControlTint = 6,
    NSClearControlTint    = 7
} NSControlTint;
```

Constants

`NSDefaultControlTint`
The current default tint setting
Available in Mac OS X v10.0 and later.
Declared in `NSCell.h`.

`NSClearControlTint`
Clear control tint
Available in Mac OS X v10.0 and later.
Declared in `NSCell.h`.

`NSBlueControlTint`
Aqua control tint
Available in Mac OS X v10.3 and later.
Declared in `NSCell.h`.

`NSGraphiteControlTint`
Graphite control tint
Available in Mac OS X v10.3 and later.
Declared in `NSCell.h`.

Discussion
These constants are used by `controlTint` (page 27) and `setControlTint:` (page 57).

Declared In
`NSCell.h`

NSControlSize

Defines a type to specify the size of a cell.

```
typedef NSUInteger NSControlSize;
```

Discussion
For possible values, see "Control Sizes" (page 94).

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

Declared In
`NSCell.h`

Control Sizes

These constants specify a cell's size.

```
typedef enum _NSControlSize {
    NSRegularControlSize,
    NSSmallControlSize,
    NSMiniControlSize
} NSControlSize;
```

Constants

`NSRegularControlSize`

The control is sized as regular.

Available in Mac OS X v10.0 and later.

Declared in `NSCell.h`.

`NSSmallControlSize`

The control has a smaller size.

This constant is for controls that cannot be resized in one direction, such as push buttons, radio buttons, checkboxes, sliders, scroll bars, pop-up buttons, tabs, and progress indicators. You should use a small system font with a small control.

Available in Mac OS X v10.0 and later.

Declared in `NSCell.h`.

`NSMiniControlSize`

The control has a smaller size than `NSSmallControlSize`.

Available in Mac OS X v10.3 and later.

Declared in `NSCell.h`.

Discussion

These constants are used by `controlSize` (page 27) and `setControlSize:` (page 56).

Declared In

`NSCell.h`

Hit Testing

These constants are used by `hitTestForEvent:inRect:ofView:` (page 36) to determine the effect of an event.

```
enum {
    NSCellHitNone = 0,
    NSCellHitContentArea = 1 << 0,
    NSCellHitEditableTextArea = 1 << 1,
    NSCellHitTrackableArea = 1 << 2,
};
```

Constants

`NSCellHitNone`

An empty area, or did not hit in the cell.

Available in Mac OS X v10.5 and later.

Declared in `NSCell.h`.

`NSCellHitContentArea`

A content area in the cell.

Available in Mac OS X v10.5 and later.

Declared in `NSCell.h`.

`NSCellHitEditableTextArea`
An editable text area of the cell.

Available in Mac OS X v10.5 and later.

Declared in `NSCell.h`.

`NSCellHitTrackableArea`
A trackable area in the cell.

Available in Mac OS X v10.5 and later.

Declared in `NSCell.h`.

Declared In
`NSCell.h`

Data Entry Types

These constants specify how a cell formats numeric data.

```
enum {
   NSAnyType            = 0,
   NSIntType            = 1,
   NSPositiveIntType    = 2,
   NSFloatType          = 3,
   NSPositiveFloatType  = 4,
   NSDoubleType         = 6,
   NSPositiveDoubleType = 7
};
```

Constants
`NSIntType`
Must be between `INT_MIN` and `INT_MAX`.

Deprecated in Mac OS X v10.4 and later.

Declared in `NSCell.h`.

`NSPositiveIntType`
Must be between 1 and `INT_MAX`.

Deprecated in Mac OS X v10.4 and later.

Declared in `NSCell.h`.

`NSFloatType`
Must be between `-FLT_MAX` and `FLT_MAX`.

Deprecated in Mac OS X v10.4 and later.

Declared in `NSCell.h`.

`NSPositiveFloatType`
Must be between `FLT_MIN` and `FLT_MAX`.

Deprecated in Mac OS X v10.4 and later.

Declared in `NSCell.h`.

`NSDoubleType`
Must be between `-FLT_MAX` and `FLT_MAX`.

Deprecated in Mac OS X v10.4 and later.

Declared in `NSCell.h`.

`NSPositiveDoubleType`
Must be between `FLT_MIN` and `FLT_MAX`.

Deprecated in Mac OS X v10.4 and later.

Declared in `NSCell.h`.

`NSAnyType`
Any value is allowed.

Deprecated in Mac OS X v10.4 and later.

Declared in `NSCell.h`.

Discussion
These constants are used by `setEntryType:` (page 99) and `entryType` (page 99).

Declared In
`NSCell.h`

NSBackgroundStyle

Defines a type to specify the background style of a cell.

```
typedef NSUInteger NSBackgroundStyle;
```

Discussion
For possible values, see "Background Styles" (page 97).

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

Declared In
`NSCell.h`

Background Styles

Background styles used with `backgroundStyle` (page 23), `setBackgroundStyle:` (page 54), and `interiorBackgroundStyle` (page 39).

```
enum {
    NSBackgroundStyleLight = 0,
    NSBackgroundStyleDark,
    NSBackgroundStyleRaised,
    NSBackgroundStyleLowered
};
```

Constants
`NSBackgroundStyleLight`
The background is a light color.

Dark content contrasts well with this background.

Available in Mac OS X v10.5 and later.

Declared in `NSCell.h`.

`NSBackgroundStyleDark`
: The background is a dark color.

  Light content contrasts well with this background.

  Available in Mac OS X v10.5 and later.

  Declared in `NSCell.h`.

`NSBackgroundStyleRaised`
: The background is intended to appear higher than the content drawn on it.

  Content might need to be inset.

  Available in Mac OS X v10.5 and later.

  Declared in `NSCell.h`.

`NSBackgroundStyleLowered`
: The background is intended to appear lower than the content drawn on it.

  Content might need to be embossed.

  Available in Mac OS X v10.5 and later.

  Declared in `NSCell.h`.

Declared In
`NSCell.h`

Notifications

NSControlTintDidChangeNotification

Sent after the user changes control tint preference. The notification object is `NSApp`. This notification does not contain a *`userInfo`* dictionary.

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

Declared In
`NSCell.h`

Deprecated NSCell Methods

A method identified as deprecated has been superseded and may become unsupported in the future.

Deprecated in Mac OS X v10.0 and later

entryType

Returns the type of data the user can type into the receiver. (Deprecated in Mac OS X v10.0 and later. Use a formatter instead—see `setFormatter:` (page 62).)

```
- (NSInteger)entryType
```

Return Value
One of the types listed for this method in "Data Entry Types" (page 96). If the receiver is not a text-type cell, or if no type has been set, `NSAnyType` is returned.

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

See Also
`- isEntryAcceptable:` (page 41)

Declared In
`NSCell.h`

setEntryType:

Sets how numeric data is formatted in the receiver and places restrictions on acceptable input. (Deprecated in Mac OS X v10.0 and later. Use a formatter instead—see `setFormatter:` (page 62).)

```
- (void)setEntryType:(NSInteger)aType
```

Parameters

aType

One of the types listed for this method in "Data Entry Types" (page 96).

Discussion
The formatter associated with the receiver is replaced with a newly instantiated formatter appropriate to the entry type.

If the receiver isn't a text-type cell, this method converts it to one; in the process, it changes its title to "Cell" and sets its font to the user's system font at 12 points.

You can check whether formatted strings conform to the entry types of cells with the `isEntryAcceptable:` (page 41) method. `NSControl` subclasses also use `isEntryAcceptable:` (page 41) to validate what users have typed in editable cells. You can control the format of values accepted and displayed in cells by creating a custom subclass of `NSFormatter` and associating an instance of that class with cells (through `setFormatter:` (page 62)). In custom `NSCell` subclasses, you can also override `isEntryAcceptable:` (page 41) to check for the validity of data entered into cells.

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

See Also
`- entryType` (page 99)

Declared In
`NSCell.h`

Document Revision History

This table describes the changes to *NSCell Class Reference*.

Date	Notes
2009-02-04	Documented truncatesLastVisibleLine and setTruncatesLastVisibleLine: methods. Added deprecated image scaling constants.
2007-10-31	Updated the description of the compare: method.
2007-07-11	Updated for Mac OS X v10.5.
2006-11-07	Added abstract to definition of stopTracking:at:inView:mouseIsUp:.
2006-06-28	Made minor changes to conform to reference consistency guidelines.
2006-05-23	Marked entryType, setEntryType: and the associated constants as deprecated in Mac OS X v10.4 and later.
	First publication of this content as a separate document.

Document Revision History

Index

I

K

L

M

N

O

P

R

S

T

W