NSCell

Inherits From:	NSObject
Conforms To:	NSCoding (NSObject), NSCopying (NSObject), NSObject (NSObject)
Declared In:	AppKit/NSCell.h

Class Description

The NSCell class provides a mechanism for displaying text or images in an NSView without the overhead of a full NSView subclass. In particular, it provides much of the functionality of the NSText class by providing access to a shared NSText object used by all instances of NSCell in an application. NSCells are also extremely useful for placing text or images at various locations in a custom subclass of NSView.

NSCell is used heavily by most of the NSControl classes to implement their internal workings. For example, NSSlider uses an NSSliderCell, NSTextField uses an NSTextFieldCell, and NSBrowser uses an NSBrowserCell. Sending a message to the NSControl is often simpler than dealing directly with the corresponding NSCell. For instance, NSControls typically invoke **updateCell:** (causing the cell to be displayed) after changing a cell attribute; whereas if you directly call the corresponding method of the NSCell, the NSCell might not automatically display itself again.

Some subclasses of NSControl (notably NSMatrix) group NSCells in an arrangement where they act together in some cooperative manner. Thus, with an NSMatrix, you can implement a uniformly sized group of radio buttons without needing an NSView for each button (and without needing an NSText object as the field editor for the text on each button).

The NSCell class provides primitives for displaying text or an image, editing text, setting and getting object values, maintaining state, highlighting, and tracking the mouse. NSCell's method **trackMouse:inRect:ofView:untilMouseUp:** implements the mechanism that sends action messages to target objects. However, NSCell implements target/action features abstractly, deferring the details of implementation to NSActionCell and its subclasses.

Object Values and Formatters

Every NSCell that displays text has a value associated with it. The NSCell stores that value as an object of potentially any type, displays it as an NSString, and returns it as a primary value or string object, according to what's requested (**intValue**, **floatValue**, **stringValue**, and so on). Formatters are objects associated with NSCells (through **setFormatter:**) that translate a cell's object value to its it textual representation and that convert what users type into the underlying object. NSCells have built-in formatters to handle common

string and numeric (**int**, **float**, **double**) translations. In addition, you can specify date and numeric types more precisely with **setEntryType:** and specify floating-point format characteristics with **setFloatingPointFormat:left:right:**. You can also implement your own formatters to provide specialized object translation; see the NSFormatter specification for more information.

The text that an NSCell displays and stores can be an attributed string. Several methods help to set and get attributed-string values, including s**etAttributedStringValue:** and **setImportsGraphics:**.

Represented Objects

Represented objects are objects that an NSCell "stands for." (They're not to be confused with an NSCell's object value, which *is* the value of the cell.) By setting a represented object for an NSCell (using **setRepresentedObject:**) you make an association between the NSCell and that object. For instance, you could have a pop-up list, each cell of which lists a color as its title; when the user selects a cell, the represented NSColor object is displayed in a color well.

Subclassing NSCell

The **initImageCell:** method is the designated initializer for NSCells that display images. The **initTextCell:** method is the designated initializer for NSCells that display text. Override one or both of these methods if you implement a subclass of NSCell that performs its own initialization. If you need to use target and action behavior, you may prefer to subclass NSActionCell or one of its subclasses, which provide the default implementation of this behavior.

If you want to implement your own mouse-tracking or mouse-up behavior, consider overriding **startTrackingAt:inView:**, **continueTracking:at:inView:**, and **stopTracking:at:inView:mouseIsUp:**. If you want to implement your own drawing, override **drawWithFrame:inView:** or **drawInteriorWithFrame:inView:**.

For more information on how NSCell is used, see the NSControl class specification.

Method Types

Initializing an NSCell	– initImageCell: – initTextCell:

Setting and getting cell values	– setObjectValue: – objectValue – hasValidObjectValue – setIntValue: – intValue – setStringValue: – stringValue – setDoubleValue: – doubleValue – setFloatValue: – floatValue
Setting and getting cell attributes	– setCellAttribute:to: – cellAttribute: – setType: – type – setState: – state – setEnabled: – isEnabled – setBezeled: – isBezeled – setBordered: – isBordered – isOpaque
Modifying textual attributes of cells	– setEditable: – isEditable – setSelectable: – isSelectable – setScrollable: – isScrollable – setAlignment: – alignment – setFont: – font – setWraps: – wraps – setAttributedStringValue: – attributedStringValue – setAllowsEditingTextAttributes: – allowsEditingTextAttributes – setImportsGraphics: – importsGraphics – setUpFieldEditorAttributes:

Setting the target and action	– setAction: – action – setTarget: – target – setContinuous: – isContinuous – sendActionOn:
Setting and getting an image	– setImage: – image
Assigning a tag	– setTag: – tag
Formatting and validating data	– setFormatter: – formatter – setEntryType: – entryType – isEntryAcceptable: – setFloatingPointFormat:left:right:
Managing menus for cells	+ defaultMenu – setMenu: – menu – menuForEvent:inRect:ofView:
Comparing cells	– compare:
Making cells respond to keyboard events	– acceptsFirstResponder – setShowsFirstResponder: – showsFirstResponder – setTitleWithMnemonic: – mnemonic – setMnemonicLocation: – mnemonicLocation – performClick:
Deriving values from other cells	– takeObjectValueFrom: – takeIntValueFrom: – takeStringValueFrom: – takeDoubleValueFrom: – takeFloatValueFrom:
Representing an object with a cell	– setRepresentedObject: – representedObject

Tracking the mouse	– trackMouse:inRect:ofView:untilMouseUp: – startTrackingAt:inView: – continueTracking:at:inView: – stopTracking:at:inView:mouseIsUp: – mouseDownFlags + perfersTrackingUntilMouseUp – getPeriodicDelay:interval:
Managing the cursor	– resetCursorRect:inView:
Handling keyboard alternatives	– keyEquivalent
Determining cell sizes	– calcDrawInfo: – cellSize – cellSizeForBounds: – drawingRectForBounds: – imageRectForBounds: – titleRectForBounds:
Drawing and highlighting cells	– drawWithFrame:inView: – drawInteriorWithFrame:inView: – controlView – highlight:withFrame:inView: – isHighlighted
Editing and selecting cell text	– editWithFrame:inView:editor:delegate:event: – selectWithFrame:inView:editor:delegate:start:length: – endEditing:

Class Methods

defaultMenu

+ (NSMenu *)**defaultMenu**

Returns the default menu for instances of the receiver. The default implementation returns **nil**.

See also: – **menu**, – **setMenu:**

prefersTrackingUntilMouseUp

+ (BOOL)**prefersTrackingUntilMouseUp**

The default implementation returns NO, so tracking stops when the mouse leaves the NSCell; subclasses may override.

See also: – **trackMouse:inRect:ofView:untilMouseUp:**

Instance Methods

acceptsFirstResponder

– (BOOL)**acceptsFirstResponder**

The default implementation returns YES if the cell is enabled; subclasses can override.

See also: – **performClick:**, – **setShowsFirstResponder:**, – **setTitleWithMnemonic:**

action

– (SEL)**action**

Implemented by NSActionCell and its subclasses to return the selector of the cell's action method. The default implementation returns a null selector.

See also: – **setAction:**, – **setTarget:**, – **target**

alignment

– (NSTextAlignment)**alignment**

Returns the alignment of text in the cell: NSLeftTextAlignment, NSRightTextAlignment, NSCenterTextAlignment, NSJustifiedTextAlignment, or NSNaturalTextAlignment.

See also: – **setAlignment:**

allowsEditingTextAttributes

– (BOOL)**allowsEditingTextAttributes**

Returns whether the receiver allows the editing of textual attributes.

See also: – **setAllowsEditingTextAttributes:**

attributedStringValue

– (NSAttributedString *)**attributedStringValue**

Returns the value of the receiver as an attributed string, using the cell's formatter object (if one exists) to create the attributed string. The textual attributes are determined by the default paragraph style, the receiver's font and alignment, and whether the receiver is enabled and scrollable.

See also: – **setAttributedStringValue:**

calcDrawInfo:

– (void)**calcDrawInfo:**(NSRect)*aRect*

Implemented by subclasses to recalculate drawing sizes with reference to *aRect*. Objects (such as NSConrols) that manage NSCells generally maintain a flag that informs them if any of their cells has been modified in such a way that the location or size of the cell should be recomputed. If so, NSControl's **calcSize** method is automatically invoked prior to the display of the NSCell, and that method invokes the NSCell's **calcDrawInfo:** method. The default implementation does nothing.

See also: – **cellSize**, – **drawingRectForBounds:**

cellAttribute:

– (int)**cellAttribute:**(NSCellAttribute)*aParameter*

Depending on *aParameter*, returns a setting for a cell attribute, such as the receiver's state, and whether it's disabled, editable, or highlighted.

See also: – **setCellAttribute:**

cellSize

– (NSSize)**cellSize**

Returns the minimum size needed to display the NSCell, taking account of the size of the image or text within a certain offset determined by border type. If the receiving cell is neither of image or text type, an extremely large size is returned; if the receiving cell is of image type, and no image has been set, an extremely small size is returned.

See also: – **drawingRectForBounds:**

cellSizeForBounds:

– (NSSize)**cellSizeForBounds:**(NSRect)*aRect*

Returns the minimum size needed to display the NSCell, taking account of the size of the image or text within an offset determined by border type. If the receiving cell is of text type, the text is resized to fit within *aRect* (as much as *aRect* is within the bounds of the cell). If the receiving cell is neither of image or text type, an extremely large size is returned; if the receiving cell is of image type, and no image has been set, an extremely small size is returned.

See also: – **drawingRectForBounds:**

compare:

– (NSComparisonResult)**compare:**(id)*otherCell*

Compares the string values of this cell and *otherCell* (which must be a kind of NSCell), disregarding case. Raises NSBadComparisonException if *otherCell* is not of the NSCell class or if one of the cells being compared is not a text-type cell.

continueTracking:at:inView:

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

Returns whether mouse-tracking should continue in the receiving cell based on *lastPoint* and *currentPoint* within *controlView* (*currentPoint* is the current location of the mouse while *lastPoint* is either the initial location of the mouse or the previous *currentPoint*). This method is invoked in **trackMouse:inRect:ofView:untilMouseIsUp:**. The default implementation returns YES if the cell is set to continuously send action messages to its target when the mouse is down or is being dragged. Subclasses can override this method to provide more sophisticated tracking behavior.

See also: – **startTrackingAt:inView:**, – s**topTracking:at:inView:mouseIsUp:**

controlView

– (NSView *)**controlView**

Implemented by subclasses to return the NSView last drawn in (normally an NSControl). The default implementation returns **nil**.

See also: – **drawWithFrame:inView:**

doubleValue

– (double)**doubleValue**

Returns the NSCell's value as a **double**. If the receiver is not a text-type cell or the cell value is not scannable, the method returns zero.

drawInteriorWithFrame:inView:

– (void)**drawInteriorWithFrame:**(NSRect)*cellFrame* **inView:**(NSView *)*controlView*

Draws the "inside" of the receiving cell; this includes the image or text within the NSCell's frame in *controlView* (usually the cell's NSControl) but excludes the border. *cellFrame* is the frame of the NSCell

or (in some cases) a portion of it. Text-type NSCells display their contents in a rectangle slightly inset from *cellFrame* using a global NSText object; image-type NSCells display their contents centered within *cellFrame*. If the proper attributes are set, it also displays the dotted-line rectangle to indicate first responder and highlights the cell. This method is invoked from NSControl's **drawCellInside:** to visually update the what the NSCell displays when its contents change. This drawing is minimal, and becomes more complex in objects such as NSButtonCell and NSSliderCell.

Subclasses often override this method to provide more sophisticated drawing of cell contents. Because **drawWithFrame:inView:** invokes **drawInteriorWithFrame:inView**: after it draws the NSCell's border, don't invoke **drawWithFrame:inView:** in your override implementation.

See also: – **isHighlighted**, – **setShowsFirstResponder:**

drawWithFrame:inView:

– (void)**drawWithFrame:**(NSRect)*cellFrame* **inView:**(NSView *)*controlView*

Draws the receiver's regular or bezeled border (if those attributes are set) and then draws the interior of the cell by invoking **drawInteriorWithFrame:inView:**.

drawingRectForBounds:

– (NSRect)**drawingRectForBounds:**(NSRect)*theRect*

Returns the rectangle within which the cell draws itself; this rectangle is slightly inset from *aRect* on all sides to take the border into account.

See also: – **calcSize**

editWithFrame:inView:editor:delegate:event:

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

Begins editing of the receiver's text by using the field editor textObj; usually invoked in response to a mouse-down event. *aRect* must be the rectangle used for displaying the NSCell. *theEvent* is the NSMouseDown event. *anObject* is made the delegate of *textObj*, and so will receive various NSText delegation and notification messages.

If the receiver isn't a text-type NSCell, no editing is performed. Otherwise, *textObj* is sized to *aRect* and its superview is set to *aView*, so that it exactly covers the NSCell. Then it's activated and editing begins. It's

the responsibility of the delegate to end the editing when responding to **textShouldEndEditing:**; in doing this, it should remove any data from *textObj* and invoke **endEditing:**.

See also: – **endEditing:**, – **selectWithFrame:inView:editor:delegate:start:length:**

endEditing:

– (void)**endEditing:**(NSText *)*textObj*

Ends any editing of text occurring in the receiver begun with **editWithFrame:inView:editor:delegate:event:** and **selectWithFrame:inView:editor:delegate:start:length:**. Usually this method is invoked by the delegate of the field editor specified in one of these methods when that delegate's **textShouldEndEditing:** method is invoked.

entryType

– (int)**entryType**

Returns the type of data the user can type into the receiver. If the receiver is not a text-type cell, or if no type has been set, NSAnyType is returned. See **setEntryType:** for a list of type constants.

See also: – **isEntryAcceptable:**

floatValue

– (float)**floatValue**

Returns the NSCell's value as a **double**. If the receiver is not a text-type cell or the cell value is not scannable, the method returns zero.

font

– (NSFont *)**font**

Returns the font used to display text in the receiving cell or **nil** if the receiver is not a text-type cell.

See also: – **setFont:**

formatter

– (id)**formatter**

Returns the formatter object (a kind of NSFormatter) associated with the cell. This object handles translation of the cell's contents between it's on-screen representation and its object value.

See also: – **setFormatter:**

getPeriodicDelay:interval:

– (void)**getPeriodicDelay:**(float *)*delay* **interval:**(float *)*interval*

Returns initial delay and repeat values for continuous sending of action messages to target objects. Subclasses can override to supply their own delay and interval values.

See also: – **isContinuous**, – **setContinuous:**

hasValidObjectValue

– (BOOL)**hasValidObjectValue**

Returns whether the object associated with the receiver has a valid object value. A valid object value is one that the receiver's formatter can "understand." Objects that are "invalid" have been rejected by the formatter, but accepted by the delegate of the receiver's NSControl (in **control:didFailToFormatString:errorDescription:**).

See also: – **objectValue**, – **setObjectValue:**

highlight:withFrame:inView:

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

If the receiver's highlight status is different from *flag*, sets that status to *flag* and, if *flag* is YES, highlights the rectangle *cellFrame* in the NSControl (*controlView*).

Note that NSCell's 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.

See also: – **drawWithFrame:inView:**, – **isHighlighted**

image

– (NSImage *)**image**

Returns the image displayed by the receiver or nil if the receiver is not an image-type cell.

See also: – **setImage:**

imageRectForBounds:

– (NSRect)**imageRectForBounds:**(NSRect)*theRect*

Returns the rectangle that the cell's image is drawn in, which is slightly offset from *theRect*.

See also: – **cellSizeForBounds:**, – **drawingRectForBounds:**

importsGraphics

– (BOOL)**importsGraphics**

Sets whether the text of the receiver (if a text-type cell) is of Rich Text Format (RTF) and thus can import graphics.

See also: – **setImportsGraphics:**

initImageCell:

– (id)**initImageCell:**(NSImage *)*anImage*

Returns an NSCell object initialized with *anImage* and set to have the cell's default menu. If *anImage* is **nil**, no image is set.

initTextCell:

– (id)**initTextCell:**(NSString *)*aString*

Returns an NSCell object initialized with *aString* and set to have the cell's default menu. If no field editor (a shared NSText object) has been created for all NSCells, one is created.

intValue

– (int)**intValue**

Returns the receiver's value as an **int**. If the receiver is not a text-type cell or the cell value is not scannable, the method returns zero.

isBezeled

– (BOOL)**isBezeled**

Returns whether the receiving cell has a bezeled border.

See also: – **setBezeled:**

isBordered

– (BOOL)**isBordered**

Returns whether the receiving cell has a plain border.

See also: – **setBordered:**

isContinuous

– (BOOL)**isContinuous**

Returns whether the receiving cell sends its action message continuously on mouse down.

See also: – **setContinuous:**

isEditable

– (BOOL)**isEditable**

Returns whether the receiving cell is editable.

See also: – **setEditable:**

isEnabled

– (BOOL)**isEnabled**

Returns whether the receiving cell responds to mouse events.

See also: – **setEnabled:**

isEntryAcceptable:

– (BOOL)**isEntryAcceptable:**(NSString *)*aString*

Returns whether a string representing a numeric or date value (*aString*) is formatted in a way suitable to the entry type.

See also: – **entryType**, – **setEntryType:**

isHighlighted

– (BOOL)**isHighlighted**

Returns whether the receiving cell is highlighted.

See also: – **setHighlighted:**

isOpaque

– (BOOL)**isOpaque**

Returns whether the receiving cell is opaque (non-transparent).

isScrollable

– (BOOL)**isScrollable**

Returns whether the receiving cell scrolls typed text that exceeds the cell’s bounds.

See also: – **setScrollable:**

isSelectable

– (BOOL)**isSelectable**

Returns whether the text of the receiving cell can be selected.

See also: – **setSelectable:**

keyEquivalent

– (NSString *)**keyEquivalent**

Implemented by subclasses to return a key equivalent to clicking the cell. The default implementation returns an empty string object.

menu

– (NSMenu *)**menu**

Returns the menu with commands contextually related to the cell or **nil** if no menu is associated.

See also: – **setMenu:**

menuForEvent:inRect:ofView:

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

Returns the NSMenu associated with the receiver through the **setMenu:** method and related to *anEvent* when the mouse is detected within *cellFrame*. It is usually invoked by the NSControl (*aView*) managing the receiver. The default implementation simply invokes NSCell's **menu** method and will return **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.

mnemonic

– (NSString *)**mnemonic**

Returns the character in the cell title that appears underlined for use as a mnemonic. If there is no mnemonic character, returns an empty string.

See also: – **setTitleWithMnemonic:**

mnemonicLocation

– (unsigned int)**mnemonicLocation**

Returns the position of the underlined character in the cell title used as a mnemonic. If there is no mnemonic character, returns NSNotFound.

See also: – **setMnemonicLocation:**

mouseDownFlags

– (int)**mouseDownFlags**

Returns the modifier flags for the last (left) mouse-down event or zero if tracking hasn't occured yet for the cell or if no modifier keys accompanied the mouse-down event.

See also: – **modifierFlags** (NSEvent)

objectValue

– (id)**objectValue**

Returns the NSCell's value as an Objective-C object if a valid object has been associated with the receiver; otherwise, returns **nil**. To be valid, the cell must have a formatter capable of converting the object to and from its textual representation.

performClick:

– (void)**performClick:**(id)*sender*

Programmatically simulates a mouse click on the receiver, including the invocation of the action method in the target object. Raises an exception if the action message cannot be successfully sent.

representedObject

– (id)**representedObject**

Returns the object the receiving cell represents. For example, you could have a pop-up list of color names, and the represented objects could be the appropriate NSColor objects.

See also: – **setRepresentedObject:**

resetCursorRect:inView:

– (void)**resetCursorRect:**(NSRect)*cellFrame* **inView:**(NSView *)*controlView*

Sets the receiver to show the I-beam cursor within *cellFrame* while it tracks the mouse . The receiver must be an enabled and selectable (or editable) text-type cell. *controlView* is the NSControl that manages the cell.

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

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

Uses the field editor *textObj* to select text in a range marked by *selStart* and *selLength*, which will be highlighted and selected as though the user had dragged the cursor over it. This method is similar to **editWithFrame:inView:editor:delegate:event:**, except that it can be invoked in any situation, not only on a mouse-down event. *aRect* is the rectangle in which the selection should occur, *controlView* is the NSControl managing the receiver, and *anObject* is the delegate of the field editor. Returns without doing anything if *controlView*, *textObj*, or the receiver are **nil**, or if the receiver as no font set for it.

sendActionOn:

– (int)**sendActionOn:**(int)*mask*

Sets the conditions on which the receiver sends action messages to its target and returns a bit mask with which to detect the previous settings. *mask* is set with one or more of these bit masks:

NSLeftMouseUpMask	Don’t send action message on (left) mouse up.
	NSLeftMouseDownMaskSend action message on (left) mouse down.
	NSLeftMouseDraggedMaskSend action message when (left) mouse is dragged.
	NSPeriodicMaskSend action message continuously.

You can send **setContinuous:** method to turn on the flag corresponding to NSPeriodicMask or NSLeftMouseDraggedMask, whichever is appropriate to the given subclass of NSCell.

See also: – **action**

setAction:

– (void)**setAction:**(SEL)*aSelector*

In NSCell, raises NSInternalInconsistencyException. However, NSActionCell overrides this method to set the action method as part of the implementation of the target/action mechanism.

See also: – **action**, – **setTarget:**, – **target**

setAlignment:

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

Sets the alignment of text in the receiver. *mode* is one of five constants: NSLeftTextAlignment, NSRightTextAlignment, NSCenterTextAlignment, NSJustifiedTextAlignment, NSNaturalTextAlignment (the default alignment for the text).

See also: – **alignment**, – **setWrap:**

setAllowsEditingTextAttributes:

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

Sets whether the textual attributes of the receiver can be modified. If *flag* is NO, the receiver cannot import graphics (that is, it does not support RTFD text).

See also: – **allowsEditingTextAttributes**, – **setImportsGraphics:**

setAttributedStringValue:

– (void)**setAttributedStringValue:**(NSAttributedString *)*attribStr*

Sets the value of the receiver to the attributed string *attribStr*. 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's converted to one. 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]];
```

See also: – **attributedStingValue**, – **hasInvalidObject**

setBezeled:

– (void)**setBezeled:**(BOOL)*flag*

Sets whether the receiver draws itself with a bezeled border. The **setBezeled:** and **setBordered:** methods are mutually exclusive (that is, a border can be only plain or bezeled).

See also: – **isBezeled**

setBordered:

– (void)**setBordered:**(BOOL)*flag*

Sets whether the receiver draws itself outlined with a plain border. The **setBezeled:** and **setBordered:** methods are mutually exclusive (that is, a border can be only plain or bezeled).

See also: – **isBordered**

setCellAttribute:to:

– (void)**setCellAttribute:**(NSCellAttribute)*aParameter* **to:**(int)*value*

Sets a cell attribute identified by *aParameter*—such as the receiver's state, and whether it's disabled, editable, or highlighted—to *value*.

See also: – **cellAttribute**

setContinuous:

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

Sets whether the receiver continuously sends its action message to its target while it tracks the mouse. In practice, the continuous setting has meaning only for instances of 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.

See also: – **isContinuous**; – **sendActionOn:**

setDoubleValue:

– (void)**setDoubleValue:**(double)*aDouble*

Sets the value of the receiving cell to an object representing a **double**. Does nothing if the receiver is not a text-type cell.

See also: – **doubleValue**

setEditable:

– (void)**setEditable:**(BOOL)*flag*

Sets whether the receiver's text is both editable and selectable. If *flag* is NO, and the cell's text was not selectable before editing was last enabled (that is, before this message was last sent with an argument of YES), then the receiver's text is set to be unselectable.

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

setEnabled:

– (void)**setEnabled:**(BOOL)*flag*

Sets whether the receiver is enabled or disabled. 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 (**setState:**, for instance, will still work).

See also: – **isEnabled**

setEntryType:

– (void)**setEntryType:**(int)*aType*

Sets how numeric data are formatted in the receiver and places restrictions on acceptable input. *aType* can be one of the following constants:

Constant	Restrictions and Other Information
NSIntType	Must be between INT_MIN and INT_MAX
NSPositiveIntType	Must be between 1 and INT_MAX
NSFloatType	Must be between -FLT_MAX and FLT_MAX
NSPositiveFloatType	Must be between FLT_MIN and FLT_MAX
NSDoubleType	Must be between -DBL_MAX and DBL_MAX
NSPositiveDoubleType	Must be between DBL_MAX and DBL_MAX
NSAnyType	Any value is allowed.

If the receiver isn't a text-type cell, this method converts it to one; in the process, it makes its title "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:** method. NSControl subclasses also use **isEntryAcceptable:** 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:**). In custom

NSCell subclasses, you can also override **isEntryAcceptable:** to check for the validity of data entered into cells.

See also: – **entryType**

setFloatingPointFormat:left:right:

– (void)**setFloatingPointFormat:**(BOOL)*autoRange*
left:(unsigned)*leftDigits*
right:(unsigned)*rightDigits*

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];
```

See also: – **setEntryType:**

setFloatValue:

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

Sets the value of the receiving cell to an object representing a **float**. Does nothing if the receiver is not a text-type cell.

See also: – **floatValue**

setFont:

– (void)**setFont:**(NSFont *)*fontObj*

Sets the font to be used when the receiver displays text. If the receiver is not a text-type cell, the method converts it to that type. If *fontObj* is **nil** and the receiver is a text-type cell, the font currently held by the receiver is autoreleased.

See also: – **font**

setFormatter:

– (void)**setFormatter:**(NSFormatter *)*newFormatter*

Sets the formatter object used to format the textual representation of the receiver's object value and to validate cell input and convert it to that object value. If the new formatter cannot interpret the receiver's current object value, that value is converted to a string object. This method retains new formatters and releases replaced ones. If *newFormatter* is **nil**, the receiver is disassociated from the current formatter.

See also: – **formatter**

setImage:

– (void)**setImage:**(NSImage *)*image*

Sets the image to be displayed by the receiver. If the receiver is not an image-type cell, the method converts it to that type. If *image* is **nil** and the receiver is an image-type cell, the image currently held by the receiver is autoreleased.

See also: – **image**

setImportsGraphics:

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

Sets whether the receiver can import images into its text (that is, whether it supports RTFD text). If flag is YES, the receiver is also set to allow editing of text attributes (**setAllowsEditingTextAttributes:**).

See also: – **importsGraphics**

setIntValue:

– (void)**setIntValue:**(int)*anInt*

Sets the value of the receiving cell to an object representing an **int**. Does nothing if the receiver is not a text-type cell.

See also: – **intValue**

setMenu:

– (void)**setMenu:**(NSMenu *)*aMenu*

Associates a menu with the cell that has commands contextually related to the cell (a pop-up menu on Windows). The associated menu is retained. If aMenu is **nil**, any association with a previous menu is removed.

See also: – **menu**

setMnemonicLocation:

– (void)**setMnemonicLocation:**(unsigned int)*location*

Sets the character of the cell title identified by *location* that is to be underlined. This character identifies the access key on Windows by which users can access the cell. *location* must be between 0 and 254.

See also: – **mnemonicLocation**

setObjectValue:

– (void)**setObjectValue:**(id)*object*

Sets the receiver's object value to *object*.

See also: – **objectValue**, – **setRepresentedObject:**

setRepresentedObject:

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

Sets the object represented by the receiver, for example, an NSColor object for a cell with a title of "Blue."

See also: – **setObjectValue:**, – **representedObject**

setScrollable:

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

Sets whether excess text in the receiver is scrolled past the cell's bounds. If flag is YES, wrapping is turned off. When the scrollable attribute is turned on, the alignment of text in the cell is changed to left alignment.

See also: – **isScrollable**

setSelectable:

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

Sets whether text in the receiver can be selected; always makes the receiver's text uneditable.

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

setShowsFirstResponder:

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

Sets whether the receiver displays a dotted-line outline when it assumes first responder status.

See also: – **showsFirstResponder**

setState:

– (void)**setState:**(int)*value*

Sets the state of the receiver to 1 (YES) if *value* is positive and 0 (NO) if *value* is non-positive.

See also: – **state**

setStringValue:

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

Sets the value of the receiving cell to an NSString object. 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.

See also: – **stringValue**

setTag:

– (void)**setTag:**(int)*anInt*

Implemented by NSActionCell to set the receiver's tag integer. NSCell's implementation raises NSInternalInconsistencyException.

See also: – **tag**

setTitleWithMnemonic:

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

Sets the title of a cell with a character underlined to denote an access key (Windows only). Use an ampersand character to mark the character (the one following the ampersand) to be underlined. For example, the following message causes the "c" in "Receive" to be underlined:

```
[aCell setTitleWithMnemonic:NSLocalizedString(@"Re&ceive")];
```

See also: – **mnemonic**, – **setMnemonicLocation:**

setTarget:

– (void)**setTarget:**(id)*anObject*

Implemented by NSActionCell to set the receiver's target object receiving the action message. NSCell's implementation raises NSInternalInconsistencyException.

See also: – **target**

setType:

– (void)**setType:**(int)*aType*

If the type of the receiving cell is different from *aType*, sets it to *aType*, which must one of NSTextTypeCell, NSImageTypeCell, or NSNullCellType. If *aType* is NSTextTypeCell, 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 NSImageTypeCell, sets a **nil** image.

See also: – **type**

setUpFieldEditorAttributes:

– (NSText *)**setUpFieldEditorAttributes:**(NSText *)*textObj*

Sets textual and background attributes of the receiver, depending on certain attributes. If the receiver is disabled, sets the text color to dark gray; otherwise sets it to the default color. If the receiver has a bezeled border, sets the background to the default color for text backgrounds; otherwise, sets it to the color of the receiver's NSControl.

setWraps:

– (void)**setWraps:**(BOOL)*flag*

Sets whether text in the receiver wraps when its length exceeds the frame of the cell. If flag is YES, then it also sets the receiver to be non-scrollable.

See also: – **wraps**

showsFirstResponder

– (BOOL)**showsFirstResponder**

Returns whether the receiver displays a dotted-line outline when it assumes first responder status.

See also: – **setShowsFirstResponder:**

startTrackingAt:inView:

– (BOOL)**startTrackingAt:**(NSPoint)*startPoint* **inView:**(NSView *)*controlView*

NSCell's implementation of **trackMouse:inRect:ofView:untilMouseIsUp:** invokes this method when tracking begins. *startPoint* is the point the mouse is currently at and *controlView* is the NSControl managing the receiver. NSCell's default implementation returns YES if the receiver is set to respond continuously or when the mouse is dragged. Subclasses override this method to implement special mouse-tracking behavior at the beginning of mouse tracking, for example, displaying a special cursor.

See also: – **continueTracking:at:inView:**, – **stopTracking:at:inView:mouseIsUp:**

state

– (int)**state**

Returns the state of the receiver, either 1 (YES) or 0 (NO).

See also: – **setState:**

stopTracking:at:inView:mouseIsUp:

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

NSCell's implementation of **trackMouse:inRect:ofView:untilMouseIsUp:** invokes this method when the mouse has left the bounds of the receiver or the mouse goes up (in which case *flag* is YES). *lastPoint* is the point the mouse was at and *stopPoint* is its current point. *controlView* is the NSControl managing the receiver. NSCell's default implementation does nothing. Subclasses often override this method to provide customized tracking behavior. The following example increments the state of a tri-state cell when the mouse is clicked.

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

See also: – **startTracking:at:inView:**, – **stopTracking:at:inView:mouseIsUp:**

stringValue

– (NSString *)**stringValue**

Returns the receiver's value as an NSString as converted by the receiver's formatter, if one exists. If no formatter exists and the value is an NSString, returns the value as an plain, attributed or localized formatted string. If the value is not an NSString or can't be converted to one, returns an empty string.

See also: – **setStringValue:**

tag

– (int)**tag**

Implemented by NSActionCell to return the receiver's tag integer. NSCell's implementation returns -1.

See also: – **setTag:**

takeDoubleValueFrom:

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

Sets the receiver's own value as a **double** using the **double** value of *sender*.

See also: – **setDoubleValue:**

takeFloatValueFrom:

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

Sets the receiver's own value as a **float** using the **float** value of *sender*.

See also: – **setFloatValue:**

takeIntValueFrom:

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

Sets the receiver's own value as an **int** using the **int** value of *sender*. The following example shows this method being used to write the value taken from a slider (**sender**) to a text field cell:

```
- (void)sliderMoved:(id)sender
{
    [[valueField cell] takeIntValueFrom:[sender cell]];
    [valueField display];
}
```

See also: – **setIntValue:**

takeObjectValueFrom:

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

Sets the receiver's own value as an object using the object value of *sender*.

See also: – **setObjectValue:**

takeStringValueFrom:

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

Sets the receiver's own value as a string object using the NSString value of *sender*.

See also: – **setStringValue:**

target

– (id)**target**

Implemented by NSActionCell to return the target object to which the receiver's action message is sent. NSCell's implementation returns **nil**.

See also: – **setTarget:**

titleRectForBounds:

– (NSRect)**titleRectForBounds:**(NSRect)*theRect*

If the receiver is a text-type cell, 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.

See also: – **imageRectForBounds:**

trackMouse:inRect:ofView:untilMouseUp:

– (BOOL)**trackMouse:**(NSEvent *)*theEvent*
inRect:(NSRect)*cellFrame*
ofView:(NSView *)*controlView*
untilMouseUp:(BOOL)*flag*

Invoked by an NSControl to initiate the tracking behavior of one of its NSCells. It's generally not overriden since the default implementation invokes other NSCell methods that can be overriden to handle specific events in a dragging session. Returns YES if the mouse goes up within *cellFrame*, NO otherwise. The argument *theEvent* is typically the mouse event received by the initiating NSControl, usually identified by *controlView*. The *flag* argument indicates whether tracking should continue until the mouse button goes up; if *flag* is NO, tracking ends when the mouse is dragged after the initial mouse down.

This method first invokes **startTrackingAt:inView:**. If that method returns YES, then as mouse-dragged events are intercepted, **continueTracking:at:inView:** is invoked, and, finally, when the mouse leaves the bounds or if the mouse button goes up, s**topTracking:at:inView:mouseIsUp:** is invoked (if *cellFrame* is NULL, then the bounds are considered infinitely large). You usually override one or more of these methods to respond to specific mouse events.

type

– (int)**type**

Returns the type of the receiver, one of NSTextTypeCell, NSImageTypeCell, or NSNullCellType.

See also: – **setType:**

wraps

– (BOOL)**wraps**

Returns whether text of the receiver wraps when it exceeds the borders of the cell.

See also: – **setWraps:**