NSActionCell

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

Class Description

An NSActionCell defines an active area inside a control (an instance of NSControl or one of its subclasses). As an NSControl's active area, an NSActionCell does three things: it usually performs display of text or an icon; it provides the NSControl with a target and an action; and it handles mouse (cursor) tracking by properly highlighting its area and sending action messages to its target based on cursor movement. The only way to specify the NSControl for a particular NSActionCell is to send the NSActionCell a **drawWithFrame:inView:** message, passing the NSControl as the argument for the **inView:** keyword of the method.

NSActionCell implements the target object and action method as defined by its superclass, NSCell. As a user manipulates an NSControl, NSActionCell's **trackMouse:inRect:ofView:untilMouseUp:** method (inherited from NSCell) updates its appearance and sends the action message to the target object with the NSControl object as the only argument. See "Target and Action" below more more on this paradigm.

Usually, the responsibility for an NSControl's appearance and behavior is completely given over to a corresponding NSActionCell. (NSMatrix, and its subclass NSForm, are NSControls that don't follow this rule.)

A single NSControl may have more than one NSActionCell. To help identify it in this case, every NSActionCell has an integer tag. Note, however, that no checking is done by the NSActionCell object itself to ensure that the tag is unique. See the NSMatrix class for an example of a subclass of NSControl that contains multiple NSActionCells.

Many of the methods that define the contents and look of an NSActionCell, such as **setFont:** and **setBordered:**, are reimplementations of methods inherited from NSCell. They're overriden to ensure that the NSActionCell is redisplayed when "visual" attributes change.

Target and Action

Target objects and action methods (or messages) are part of the mechanism by which NSControls respond to user actions and enable users to communicate their intentions to an application. A target is an object that an NSControl uses as the receiver of action messages. The target's class defines an action method to enable its instances to respond to these messages, which are sent as users click or otherwise manipulate the

NSControl. NSControl's **sendAction:to:** asks the NSApplication object, NSApp, to send an action message to the NSControl's target object.

An action method takes only one argument: the **id** of the sender. The sender may be either the NSControl that sends the action message or, on occassion, another object that the target should treat as the sender. When it receives an action message, a target can return messages to the sender requesting additional information about its status.

You can also set the target to **nil** and allow it to be determined at run time. When the target is **nil**, the NSApplication object must look for an appropriate receiver. It conducts its search in a prescribed order, by following the responder chain until it finds an object that can respond to the message:

- It begins with the first responder in the key window and follows **nextResponder** links up the responder chain to the NSWindow's content view.
- It tries the NSWindow object and then the NSWindow's delegate.
- If the main window is different from the key window, it then starts over with the first responder in the main window and works its way up the main window's responder chain to the NSWindow object and its delegate.
- Next, the NSApplication object tries to respond itself. If it can't respond, it tries its own delegate. NSApp and its delegate are the receivers of last resort.

NSControl provides methods for setting and using the target object and the action method. However, these methods require that an NSControl's cell (or cells) be NSActionCells or custom cells that hold action and target as instance variables and can respond to the NSControl methods.

Method Types

Configuring an NSActionCell	– setAlignment: – setBezeled: – setBordered: – setEnabled: – setFloatingPointFormat:left:right: – setFont: – setImage:
Obtaining and setting cell values	– doubleValue – floatValue – intValue – stringValue – setStringValue: – setObjectValue:

Displaying the NSActionCell	– drawWithFrame:inView: – controlView
Assigning target and action	– setAction: – action – setTarget: – target
Assigning a tag	– setTag: – tag

Instance Methods

action

– (SEL)**action**

Returns the NSActionCell's action-message selector.

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

controlView

– (NSView *)**controlView**

Returns the view (normally an NSControl) in which the NSActionCell was last drawn or **nil** if the NSActionCell has no control view (usually because it hasn't yet been placed in the view hierarchy).

See also: – **drawWithFrame:inView:**

doubleValue

– (double)**doubleValue**

Returns the NSActionCell's value as a **double** after validating any editing of cell content. If the receiver is not a text-type cell or the cell value is not scannable, the method returns zero.

See also: – **validateEditing** (NSControl)

drawWithFrame:inView:

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

Draws the NSActionCell's regular or bezeled border (if those attributes are set) and then draws the interior of the cell. NSActionCell's method overrides this method to replace its controlling control with *controlView* (if they're different) before invoking NSCell's **drawWithFrame:inView:**.

See also: – **controlView**

floatValue

– (float)**floatValue**

Returns the NSActionCell's value as a **float** after validating any editing of cell content. If the receiver is not a text-type cell or the cell value is not scannable, the method returns zero.

See also: – **validateEditing** (NSControl)

intValue

– (int)**intValue**

Returns the NSActionCell's value as a **int** after validating any editing of cell content. If the receiver is not a text-type cell or the cell value is not scannable, the method returns zero.

See also: – **validateEditing** (NSControl)

setAction:

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

Sets the selector used for the action message to *aSelector*.

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

setAlignment:

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

Sets the alignment of text in the receiving NSActionCell; *mode* is one of five constants: NSLeftTextAlignment, NSRightTextAlignment, NSCenterTextAlignment, NSJustifiedTextAlignment, NSNaturalTextAlignment (the default alignment for the text). The method marks the receiving NSActionCell as needing redisplay after discarding any editing changes that were being made to cell text.

setBezeled:

– (void)**setBezeled:**(BOOL)*flag*

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

setBordered:

– (void)**setBordered:**(BOOL)*flag*

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

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. The method marks the receiving NSActionCell as needing redisplay after discarding any editing changes that were being made to cell text.

setFloatingPointFormat:left:right:

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

Sets the NSActionCell's floating point format as described in the NSCell class specification for the **setFloatingPointFormat:left:right:** method. NSActionCell's implementation of the method supplements NSCell's by marking the receiving NSActionCell as needing redisplay after discarding any editing changes that were being made to cell text.

setFont:

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

Sets the font to be used when the NSActionCell 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. NSActionCell supplements NSCell's implementation of this method by

marking the updated cell as needing redisplay; if the receiving NSActionCell was converted to a text-type cell and is selected, it also updates the field editor with *fontObj*.

setImage:

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

Sets the font to be used when the NSActionCell 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. NSActionCell supplements NSCell's implementation of this method by marking the updated cell as needing redisplay.

setObjectValue:

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

Discards any editing of the receiving NSActionCell's text and sets its object value to *object*. If the object value is afterwards different from what it was before the method was invoked, the method marks the NSActionCell as needing redisplay.

See also: – **objectValue**

setTag:

– (void)**setTag:**(int)*anInt*

Sets the receiving NSActionCell's tag to *anInt*.

See also: – **tag**

setTarget:

– (void)**setTarget:**(id)*anObject*

Sets the receiving NSActionCell's target object to *anObject*.

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

stringValue

– (NSString *)**stringValue**

Returns the receiving NSActionCell's value as a string object as converted by the cell'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. The method supplements NSCell's implementation by validating and retaining any editing changes being made to cell text.

See also: – **validateEditing** (NSControl)

tag

– (int)**tag**

Returns the receiving NSActionCell's tag.

See also: – **setTag:**

target

– (id)**target**

Returns the receiving NSActionCell's target object.

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