NSButton

Inherits From:	NSControl : NSView : NSResponder : NSObject
Conforms To:	NSCoding (from NSResponder) NSObject (from NSObject)
Declared In:	AppKit/NSButton.h

Class Description

NSButton is a subclass of NSControl that intercepts mouse-down events and sends an action message to a target object when it's clicked or pressed. By virtue of its NSButtonCell, NSButton is a two-state NSControl—it's either "off" or "on"—and it displays its state depending on the configuration of the NSButtonCell. NSButton acquires other attributes of NSButtonCell. The state is used as the value, so NSControl methods like **setIntValue:** actually set the state (the methods **setState:** and **state** are provided as a more conceptually accurate way of setting and getting the state). The NSButton can send its action continuously and display highlighting in several different ways. What's more, an NSButton can have a key equivalent that's eligible for triggering whenever the NSButton's NSPanel or NSWindow is the key window.

NSButton and NSMatrix both provide a control view, which is needed to display an NSButtonCell object. However, while NSMatrix requires you to access the NSButtonCells directly, most of NSButton's methods are "covers" for identically declared methods in NSButtonCell. (In other words, the implementation of the NSButton method invokes the corresponding NSButtonCell method for you, allowing you to be unconcerned with the NSButtonCell's existence.) The only NSButtonCell methods that don't have covers relate to the font used to display the key equivalent, and to specific methods for highlighting or showing the NSButton's state (these last are usually set together with NSButton's **setButtonType:** method).

Creating a Subclass of NSButton

Override the designated initializer (NSView's **initWithFrame:** method) if you create a subclass of NSButton that performs its own initialization. If you want to use a custom NSButtonCell subclass with your subclass of NSButton, you have to override the **setCellClass:** method, as described in "Creating New NSControls" in the NSControl class specification.

See the NSButtonCell class specification for more on NSButton's behavior.

Method Types

Initializing the NSButton factory	+ cellClass + setCellClass:
Setting the button type	– setButtonType:
Setting the state	– setState: – state
Setting the repeat interval	– getPeriodicDelay:interval: – setPeriodicDelay:interval:
Setting the titles	– alternateTitle – attributedAlternateTitle – attributedTitle – setAlternateTitle: – setAttributedAlternateTitle – setAttributedTitle – setTitle: – title
Setting the images	– alternateImage – image – imagePosition – setAlternateImage: – setImage: – setImagePosition:
Modifying graphic attributes	– isBordered – isTransparent – setBordered: – setTransparent:
Displaying	– highlight:
Setting the key equivalent	– keyEquivalent – keyEquivalentModifierMask – setKeyEquivalent: – setKeyEquivalentModifierMask:
Handling events and action messages	– performClick: – performKeyEquivalent:

Class Methods

cellClass

+ (Class)**cellClass**

Returns the class of cells used by the receiving class (which must be NSButtonCell or one of its subclasses). Returns **nil** if no cell class has been specified for the receiving class or any of its superclasses (up to NSButtonCell).

setCellClass:

+ (void)**setCellClass:**(Class)*classId*

Configures the NSButton class to use instances of *classId* for its NSCells. *classId* should be the **id** of a subclass of NSButtonCell, obtained by sending the **class** message to either the NSCell subclass object or to an instance of that subclass. The default NSCell class is NSButtonCell.

If this method isn't overridden by a subclass of NSButton, then when it's sent to that subclass, NSButton and any other subclasses of NSButton will use the new NSCell subclass as well. To safely set an NSCell class for your subclass of NSButton, override this method to store the NSCell class in a static **id**. Also, override the designated initializer to replace the NSButton subclass instance's NSCell with an instance of the NSCell subclass stored in that static **id**. See "Creating New NSControls" in the NSControl class specification's class description for more information.

Instance Methods

alternateImage

– (NSImage *)**alternateImage**

Returns the image that appears on the button when it's in its alternate state, or **nil** if there is no alternate image. Note that some button types don't display an alternate image. Buttons don't display images by default.

See also: **– image**, **– imagePosition**, **– keyEquivalent**, **– setButtonType:**

alternateTitle

– (NSString *)**alternateTitle**

Returns the string that appears on the button when it's in its alternate state, or the empty string if the button doesn't display an alternate title. Note that some button types don't display an alternate title. By default, a button's alternate title is "Button".

See also: **– attributedAlternateTitle**, **– setButtonType:**, **– title**

attributedAlternateTitle

– (NSAttributedString *)**attributedAlternateTitle**

Returns the string that appears on the button when it's in its alternate state as an NSAttributedString, or an empty attributed string if the button doesn't display an alternate title. Note that some button types don't display an alternate title. By default, a button's alternate title is "Button".

See also: **– setButtonType:**, **– attributedTitle**

attributedTitle

– (NSAttributedString *)**attributedTitle**

Returns the string that appears on the button when it's in its normal state as an NSAttributedString, or an empty attributed string if the button doesn't display a title. A button's title is always displayed if the button doesn't use its alternate contents for highlighting or displaying the alternate state. By default, a button's title is "Button".

See also: **– attributedAlternateTitle**, **– setButtonType:**

getPeriodicDelay:interval:

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

Returns by reference the delay and interval periods for a continuous button. *delay* is the amount of time (in seconds) that the button will pause before starting to periodically send action messages to the target object. *interval* is the amount of time (also in seconds) between those messages.

Default delay and interval values are taken from a user's defaults (60 seconds maximum for each); if the user hasn't specified default values, *delay* defaults to 0.4 seconds and *interval* defaults to 0.075 seconds.

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

highlight:

– (void)**highlight:**(BOOL)*flag*

Highlights (or unhighlights) the button according to *flag*. Highlighting may involve the button appearing "pushed in" to the screen, displaying its alternate title or image, or causing the button to appear to be "lit." If the current state of the button matches *flag*, no action is taken.

See also: – setButtonType:

image

– (NSImage *)**image**

Returns the image that appears on the button when it's in its normal state, or **nil** if there is no such image. This image is always displayed on a button that doesn't change its contents when highlighting or showing its alternate state. Buttons don't display images by default.

See also: – **alternateImage**, – **setButtonType:**

imagePosition

– (NSCellImagePosition)**imagePosition**

Returns the position of the button's image relative to its title. The return value is one of the following (these are defined in **NSCell.h**):

Return Value	Meaning
NSNoImage	The button doesn't display an image (this is the default)
NSImageOnly	The button displays an image, but not a title
NSImageLeft	The image is to the left of the title
NSImageRight	The image is to the right of the title
NSImageBelow	The image is below the title
NSImageAbove	The image is above the title
NSImageOverlaps	The image overlaps the title

If the title is above, below, or overlapping the image, or if there is no image, the text is horizontally centered within the button.

See also: – **setButtonType:**, **– setImage:**, **– setTitle:**

isBordered

– (BOOL)**isBordered**

Returns YES if the button has a border, NO otherwise. A button's border isn't the single line of most other controls' borders; instead, it's a raised bezel. By default, buttons are bordered.

isTransparent

– (BOOL)**isTransparent**

Returns YES if the button is transparent, NO otherwise. A transparent button never draws itself, but it receives mouse-down events and tracks the mouse properly.

keyEquivalent

– (NSString *)**keyEquivalent**

Returns the key-equivalent character of the button, or the empty string if one hasn't been defined. Buttons don't have a default key equivalent.

See also: **– keyEquivalentFont** (NSButtonCell), – **performKeyEquivalent:**

keyEquivalentModifierMask

– (unsigned int)**keyEquivalentModifierMask**

Returns the mask indicating the modifier keys that are applied to the button's key equivalent. Mask bits are defined in **NSEvent.h**; only NSControlKeyMask, NSAlternateKeyMask, and NSCommandKeyMask bits are relevant in button key-equivalent modifier masks.

See also: – keyEquivalent:

performClick:

– (void)**performClick:**(id)*sender*

Simulates the user's clicking the button with the mouse. This method essentially highlights the button, sends the button's action message to the target object, and then unhighlights the button. If an exception is raised while the target object is processing the action message, the button is unhighlighted before the exception is propagated out of **performClick:**.

See also: – performKeyEquivalent:

performKeyEquivalent:

– (BOOL)**performKeyEquivalent:**(NSEvent *)*anEvent*

If the character in *anEvent* matches the button's key equivalent, and the modifier flags in *anEvent* match the key-equivalent modifier mask, **performKeyEquivalent:** simulates the user clicking the button by sending **performClick:** to **self**, and returns YES. Otherwise, **performKeyEquivalent:** does nothing and returns NO. **performKeyEquivalent:** also returns NO in the event that the button is blocked by a modal panel or the button is disabled.

See also: – keyEquivalentModifierMask

setAlternateImage:

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

Sets the image that appears on the button when it's in its alternate state to *image* and, if necessary, redraws the contents of the button. Note that some button types don't display an alternate image.

See also: – **setImage:**, – **setButtonType:**

setAlternateTitle:

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

Sets the string that appears on the button when it's in its alternate state to *aString*. Note that some button types don't display an alternate title.

See also: – **setTitle:**, – **setButtonType:**, – **setFont:** (NSButtonCell)

setAttributedAlternateTitle:

– (void)**setAttributedAlternateTitle:**(NSAttributedString *)*aString*

Sets the string that appears on the button when it's in its alternate state to the attributed string *aString*. Note that some button types don't display an alternate title.

See also: – **setAttributedTitle:**, – **setButtonType:**, – **setFont:** (NSButtonCell)

setAttributedTitle:

– (void)**setAttributedTitle:**(NSAttributedString *)*aString*

Sets the string that appears on the button when it's in its normal state to the attributed string *aString* and redraws the button. The title is always shown on buttons that don't use their alternate contents when highlighting or displaying their alternate state.

See also: – **setAttributedAlternateTitle:**, – **setButtonType:**, – **setFont:** (NSButtonCell)

setBordered:

– (void)**setBordered:**(BOOL)*flag*

Sets whether the button has a bezeled border. If *flag* is YES, the button displays a border; if NO, the button doesn't display a border. A button's border is not the single line or most other controls' borders; instead, it's a raised bezel. This method redraws the button if **setBordered:** causes the bordered state to change.

setButtonType:

– (void)**setButtonType:**(NSButtonType)*aType*

Sets how the button highlights while pressed and how it shows its state. **setButtonType:** redisplays the button before returning.

The types available are for the most common button types, which are also accessible in Interface Builder. You can configure different behavior with NSButtonCell's **setHighlightsBy:** and **setShowsStateBy:** methods.

aType can be one of eight constants:

Button Type	Description
NSMomentaryLight	While the button is held down it's shown as "lit." This type of button is best for simply triggering actions, as it doesn't show its state; it always displays its normal image or title. This option is called "Momentary Light" in Interface Builder's Button Inspector. This is the default button type.
NSMomentaryPushButton	While the button is held down it's shown as "lit," and also "pushed in" to the screen if the button is bordered. This type of button is best for simply triggering actions, as it doesn't show its state; it always displays its normal image or title. This option is called "Momentary Push" in Interface Builder's Button Inspector.
NSMomentaryChangeButton	While the button is held down, the alternate image and alternate title are displayed. The normal image or title are displayed when the button isn't pressed. This option is called "Momentary Change" in Interface Builder's Button Inspector.
NSPushOnPushOffButton	The first click both highlights and causes the button to be "pushed in" if the button is bordered. A second click returns it to its normal state. This option is called "Push On/Push Off" in Interface Builder's Button Inspector.
NSOnOffButton	The first click highlights the button. A second click returns it to the normal (unhighlighted) state. This option is called "On/Off" in Interface Builder's Button Inspector.
NSToggleButton	The first click highlights the button, while a second click returns it to its normal state. Highlighting is performed by changing to the alternate title or image and showing the button as "pushed in" if the button is bordered. This option is called "Toggle" in Interface Builder's Button Inspector.
NSSwitchButton	This is a variant of NSToggleButton that has no border, with the default image set to "NSSwitch," and the alternate image set to "NSHighlightedSwitch" (these are system bitmaps). This type of button is available as a separate palette item in Interface Builder.
NSRadioButton	Like NSSwitchButton, but the default image is set to "NSRadioButton" and the alternate image is set to "NSHighlightedRadioButton" (these are system bitmaps). This type of button is available as a separate palette item in Interface Builder.

See also: **– setAlternateImage:**, – **setButtonType:** (NSButtonCell), **– setImage:**

setImage:

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

Sets the button's image to *anImage*, and redraws the button. A button's image is displayed when the button is in its normal state, or all the time for a button that doesn't change its contents when highlighting or displaying its alternate state.

See also: – **setImagePosition:**, – **setAlternateImage:**, – **setButtonType:**

setImagePosition:

– (void)**setImagePosition:**(NSCellImagePosition)*aPosition*

Sets the position of the button's image relative to its title. See the **imagePosition** method description for a listing of possible values for *aPosition*.

setKeyEquivalent:

– (void)**setKeyEquivalent:**(NSString *)*charCode*

Sets the key equivalent character of the button, and redraws the button's interior if it displays a key equivalent instead of an image. The key equivalent isn't displayed if the image position is set to NSNoImage, NSImageOnly or NSImageOverlaps; that is, the button must display both its title and its "image" (the key equivalent in this case), and they must not overlap.

To display a key equivalent on a button, set the image and alternate image to **nil**, then set the key equivalent, then set the image position.

See also: – **performKeyEquivalent:**, – **setAlternateImage:**, – **setImage:**, – **setImagePosition:**, **– setKeyEquivalentFont:** (NSButtonCell)

setKeyEquivalentModifierMask:

– (void)**setKeyEquivalentModifierMask:**(unsigned int)*mask*

Sets the mask indicating the modifier keys to be applied to the button's key equivalent. Mask bits are defined in **NSEvent.h**; only NSControlKeyMask, NSAlternateKeyMask, and NSCommandKeyMask bits are relevant in button key-equivalent modifier masks.

See also: – setKeyEquivalent:

setPeriodicDelay:interval:

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

Sets the message delay and interval for the button. These two values are used if the button is configured (by a **setContinuous:** message) to continuously send the action message to the target object while tracking the mouse. *delay* is the amount of time (in seconds) that a continuous button will pause before starting to periodically send action messages to the target object. *interval* is the amount of time (also in seconds) between those messages.

The maximum value allowed for both *delay* and *interval* is 60.0 seconds; if a larger value is supplied, it's ignored and 60.0 seconds is used.

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

setState:

– (void)**setState:**(int)*value*

Sets the button's state to *value* and, if necessary, redraws the button. 0 is the normal or "off" state, and any nonzero number is the alternate or "on" state.

setTitle:

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

Sets the title displayed by the button when in its normal state to *aString* and, if necessary, redraws the button's contents. This title is always shown on buttons that don't use their alternate contents when highlighting or displaying their alternate state.

See also: – **setAlternateTitle:**, – **setButtonType:**, – **setFont:** (NSButtonCell)

setTransparent:

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

Sets whether the button is transparent, and redraws the button if *flag* is NO and the button wasn't already transparent. A transparent button tracks the mouse and sends its action, but doesn't draw. A transparent button is useful for sensitizing an area on the screen so that an action gets sent to a target when the area receives a mouse click.

state

– (int)**state**

Returns the button's state: 0 for normal or "off," or 1 for alternate or "on."

title

– (NSString *)**title**

Returns the title displayed on the button when it's in its normal state (this title is always displayed if the button doesn't use its alternate contents for highlighting or displaying the alternate state). Returns the empty string if the button doesn't display a title. By default, a button's title is "Button".

See also: – **alternateTitle**, – **setButtonType:**