# NSButtonCell

**Inherits From:** NSActionCell : NSCell : NSObject

**Conforms To:** NSCoding (from NSCell)
NSCopying (from NSCell)
NSObject (from NSObject)

**Declared In:** AppKit/NSButtonCell.h

## Class Description

NSButtonCell is a subclass of NSActionCell used to implement the user interfaces of push buttons, switches, and radio buttons. It can also be used for any other region of a view that's designed to send a message to a target when clicked. The NSButton subclass of NSControl uses a single NSButtonCell. To create groups of switches or radio buttons, use an NSMatrix holding a set of NSButtonCells.

An NSButtonCell is a two-state cell; it's either "off" or "on," and can be configured to display the two states differently, with a separate title and/or image for either state. The two states are more often referred to as "normal" and "alternate." An NSButtonCell's state is also used as its value, so NSCell methods that set the value (**setIntValue:** and so on) actually set the NSButtonCell's state to "on" if the value provided is non-zero (or non-null for strings), and to "off" if the value is zero or null. Similarly, methods that retrieve the value return 1 for the "on" or alternate state (**stringValue** returns an NSString containing a single character "1"), or 0 for the "off" or normal state (**stringValue** returns an NSString containing a single character "0"). You can also use NSCell's **setState:** and **state** methods to set or retrieve the state directly. After changing the state, send a **display** message to show the NSButtonCell's new appearance. (NSButton does this automatically.)

An NSButtonCell sends its action message to its target once if its view is clicked and it gets the mouse-down event, but can also send the action message continuously as long as the mouse is held down with the cursor inside the NSButtonCell. The NSButtonCell can show that it's being pressed by highlighting in several ways—for example, a bordered NSButtonCell can appear pushed into the screen, or the image or title can change to an alternate form while the NSButtonCell is pressed.

An NSButtonCell can also have a key equivalent (like a menu item). If the NSButtonCell is displayed in the key window, the NSButtonCell gets the first chance to receive events related to key equivalents. This feature is used quite often in modal panels that have an "OK" button. An NSButtonCell can either display a graphical image representing the key equivalent, or you can mark the keyboard "mnemonic" character in the NSButtonCell's title using **setTitleWithMnemonic:**, **setAlternateTitleWithMnemonic:**, or **setAlternateMnemonicLocation:**.

For more information on NSButtonCell's behavior, see the NSButton and NSMatrix class specifications.

### Exceptions

In its implementation of the **compare:** method (declared in NSCell), NSButtonCell raises an NSBadComparisonException if the *otherCell* argument is not of the NSButtonCell class.

## Method Types

| | |
|---|---|
| Setting the titles | – alternateMnemonic<br>– alternateMnemonicLocation<br>– alternateTitle<br>– attributedAlternateTitle<br>– attributedTitle<br>– setAlternateMnemonicLocation:<br>– setAlternateTitle:<br>– setAlternateTitleWithMnemonic:<br>– setAttributedAlternateTitle:<br>– setAttributedTitle:<br>– setFont:<br>– setTitle:<br>– setTitleWithMnemonic:<br>– title |
| Setting the images | – alternateImage<br>– imagePosition<br>– setAlternateImage:<br>– setImagePosition: |
| Setting the repeat interval | – getPeriodicDelay:interval:<br>– setPeriodicDelay:interval: |
| Setting the key equivalent | – keyEquivalent<br>– keyEquivalentFont<br>– keyEquivalentModifierMask<br>– setKeyEquivalent:<br>– setKeyEquivalentModifierMask:<br>– setKeyEquivalentFont:<br>– setKeyEquivalentFont:size: |
| Modifying graphic attributes | – imageDimsWhenDisabled<br>– isOpaque<br>– isTransparent<br>– setImageDimsWhenDisabled:<br>– setTransparent: |

| | |
|---|---|
| Displaying | – highlightsBy<br>– setHighlightsBy:<br>– setShowsStateBy:<br>– setButtonType:<br>– showsStateBy: |
| Simulating a click | – performClick: |

## 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** (NSCell), **– imagePosition**, **– keyEquivalent**, **– setButtonType:**

### alternateMnemonic

– (NSString *)**alternateMnemonic**

Returns the character in the alternate title (the title displayed on the button cell when it's in its alternate state) that's marked as the "keyboard mnemonic." If the alternate title doesn't have a keyboard mnemonic, the empty string is returned.

**See also:** **– alternateMnemonicLocation**, **– mnemonic** (NSCell), **setAlternateTitleWithMnemonic:**

### alternateMnemonicLocation

– (unsigned)**alternateMnemonicLocation**

Returns an unsigned integer indicating the character in the alternate title (the title displayed on the button cell when it's in its alternate state) that's marked as the "keyboard mnemonic." If the alternate title doesn't have a keyboard mnemonic, NSNotFound is returned.

**See also:** **– alternateMnemonic**, **– mnemonicLocation** (NSCell), **setAlternateTitleWithMnemonic:**

## alternateObjectValue

– (id)**alternateObjectValue**

## 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:** **– alternateMnemonic**, **– 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:** **– alternateMnemonic**, **– attributedTitle**, **– setButtonType:**

## 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**, **– mnemonic** (NSCell), **– 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** (NSCell)

### highlightsBy

– (int)**highlightsBy**

Returns the logical OR of flags that indicate the way the button cell highlights when it receives a mouse-down event. See **setHighlightsBy:** for the list of flags.

**See also:** – **showStateBy**

### imageDimsWhenDisabled

– (BOOL)**imageDimsWhenDisabled**

Returns whether the button cell's image and text appear "dim" when the button cell is disabled. By default, all button types except NSSwitchButton and NSRadioButton do dim when disabled. When NSSwitchButtons and NSRadioButtons are disabled, only the associated text dims.

**See also:** – 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:** (NSCell), **– setTitle:**

## isOpaque

– (BOOL)**isOpaque**

Returns YES if the button cell draws over every pixel in its frame, NO if not. The button cell is opaque only if it isn't transparent and if it has a border.

**See also:** – isTransparent

## 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.

**See also:** – isOpaque

## 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**, – **performKeyEquivalent:**

## keyEquivalentFont

– (NSFont *)**keyEquivalentFont**

Returns the font used to draw the key equivalent, or **nil** if the button cell doesn't have a key equivalent. The default font is the same as that used to draw the title.

**See also:** – setFont:

### 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:**.

### 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:** (NSCell), – **setButtonType:**

### setAlternateMnemonicLocation:

– (void)**setAlternateMnemonicLocation:**(unsigned)*location*

Sets the character in the alternate title (the title displayed on the button cell when it's in its alternate state) that's to be marked as the "keyboard mnemonic." The character specified by *location* will be underlined; *location* can be any integer from 0 to 254. If you don't want the alternate title to have a keyboard mnemonic, specify a location of NSNotFound.

**setAlternateMnemonicLocation:** doesn't cause the button cell to be redisplayed.

**See also:** – setAlternateTitleWithMnemonic:

### setAlternateTitle:

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

Sets the title that's displayed 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:** **– setAlternateMnemonicLocation:**, **– setAlternateTitleWithMnemonic:**, – **setTitle:**, – **setButtonType:**, – **setFont:**

### setAlternateTitleWithMnemonic:

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

Sets the title that is displayed on the button cell when it's in its alternate state to *aString*, taking into account the fact that an embedded "&" character is not a literal but instead marks the alternate state's "keyboard mnemonic." The character in the title that immediately follows the "&" character will be underlined.

If necessary, **setAlternateTitleWithMnemonic:** redraws the button cell. Note that some button types don't display an alternate title.

**See also:** **– setAlternateMnemonicLocation:**, **– setTitleWithMnemonic:**

### 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:** **– setAlternateMnemonicLocation:**, **– setAlternateTitleWithMnemonic:**, – **setAttributedTitle:**, – **setButtonType:**, – **setFont:**

### 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:**, **– setMnemonicLocation:** (NSCell)

### 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 the **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 and 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:**, **– setImage:** (NSCell)

### setFont:

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

Sets the font used to display the title and alternate title. Does nothing if the button cell has no title or alternate title.

If the button cell has a key equivalent, its font is not changed, but the key equivalent's font size is changed to match the new title font.

**See also:** **– Font** (NSCell), – **setKeyEquivalentFont:**, – **setKeyEquivalentFont:size:**

### setHighlightsBy:

– (void)**setHighlightsBy:**(int)*aType*

Sets the way the button cell highlights itself while pressed. *aType* can be the logical OR of one or more of the following constants:

| | |
|---|---|
| NSNoCellMask | The button cell doesn't change. This flag is ignored if any others are set in *aType*. |
| NSPushInCellMask | The button cell "pushes in" when pressed if it has a border. This is the default behavior. |
| NSContentsCellMask | The button cell displays its alternate icon and/or title. |
| NSChangeGrayCellMask | The button cell swaps the "control color" (NSColor's controlColor) and white pixels on the its background and icon. |
| NSChangeBackgroundCellMask | Same as NSChangeGrayCellMask, but only background pixels are changed. |

If both NSChangeGrayCellMask and NSChangeBackgroundCellMask are specified, both are recorded, but which behavior is used depends on the button cell's image. If the button has no image, or if the image has no alpha (transparency) data, NSChangeGrayCellMask is used. If the image does have alpha data, NSChangeBackgroundCellMask is used; this allows the color swap of the background to show through the image's transparent pixels.

**See also:** – **setShowsStateBy:**

### setImageDimsWhenDisabled:

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

Sets whether the button cell's image and text appear "dim" when the button cell is disabled. By default, all button types except NSSwitchButton and NSRadioButton do dim when disabled. When NSSwitchButtons and NSRadioButtons are disabled, only the associated text associated dims. The default setting for this condition is reasserted whenever you invoke **setButtonType:**, so be sure to specify the button cell's type before you invoke **setImageDimsWhenDisabled:**.

### 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 *)*aKeyEquivalent*

Sets the key equivalent character of the button, and redraws the button's inside 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:** (NSCell), – **setImagePosition:**, **– setKeyEquivalentFont:**

### setKeyEquivalentFont:

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

Sets the font used to draw the key equivalent, and redisplays the button cell if necessary. Does nothing if the button cell doesn't have a key equivalent associated with it. The default font is the same as that used to draw the title.

**See also:** – setFont:

### setKeyEquivalentFont:size:

– (void)**setKeyEquivalentFont:**(NSString *)*fontName* **size:**(float)*fontSize*

Sets by name and size the font used to draw the key equivalent, and redisplays the button cell if necessary. Does nothing if the button cell doesn't have a key equivalent associated with it. The default font is the same as that used to draw the title.

**See also:** – setFont:

### 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** (NSCell)

### setShowsStateBy:

– (void)**setShowsStateBy:**(int)*aType*

Sets the way the button cell indicates its alternate state. *aType* should be the logical OR of one or more of the following constants:

| | |
|---|---|
| NSNoCellMask | The button cell doesn't change. This mask is ignored if any others are set in *aType*. This is the default. |
| NSContentsCellMask | The button cell displays its alternate icon and/or title. |
| NSChangeGrayCellMask | The button cell swaps the "control color" (NSColor's controlColor) and white pixels on its background and icon. |
| NSChangeBackgroundCellMask | Same as NSChangeGrayCellMask, but only the background pixels are changed. |

If both NSChangeGrayCellMask and NSChangeBackgroundCellMask are specified, both are recorded, but the actual behavior depends on the button cell's image. If the button has no image, or if the image has no alpha (transparency) data, NSChangeGrayCellMask is used. If the image exists and has alpha data, NSChangeBackgroundCellMask is used; this allows the color swap of the background to show through the image's transparent pixels.

**See also:** – **setHighlightsBy:**

### setTitle:

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

Sets the title displayed by the button cell 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:**, **– setTitleWithMnemonic:**

### setTitleWithMnemonic:

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

Sets the title displayed on the button cell when it's in its normal state to *aString*, taking into account the fact that an embedded "&" character is not a literal but instead marks the normal state's "keyboard mnemonic." If necessary, **setTitleWithMnemonic:** redraws the button cell. This title is always shown on buttons that don't use their alternate contents when highlighting or displaying their alternate state.

**See also:** **– setAlternateTitleWithMnemonic:**, **– setMnemonicLocation:** (NSCell), **– setTitleWithMnemonic:** (NSCell)

### 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.

### showsStateBy

– (int)**showsStateBy**

Returns the logical OR of flags that indicate the way the button cell shows its alternate state. See **setShowsStateBy:** for the list of flags.

**See also:** – **highlightsBy**

### 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**, **– mnemonic**, **– mnemonicLocation**, – **setButtonType:**