

---

# Button Programming Topics for Cocoa

[Cocoa > User Experience](#)



2008-10-15



Apple Inc.  
© 2008 Apple Inc.  
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

The Apple logo is a trademark of Apple Inc.

Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

Every effort has been made to ensure that the information in this document is accurate. Apple is not responsible for typographical errors.

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

Apple, the Apple logo, and Cocoa are trademarks of Apple Inc., registered in the United States and other countries.

Simultaneously published in the United States and Canada.

**Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.**

**IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY**

**DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.**

**THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.**

**Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.**

# Contents

---

## **Introduction to Buttons 7**

Organization of This Document 7

---

## **How Buttons Work 9**

---

## **Types of Button 11**

Push Buttons 11

Sticky Buttons 12

Radio Buttons and Checkboxes 12

---

## **Using Push Buttons 15**

---

## **Using Checkboxes 17**

Standard Checkboxes 17

Icon Checkboxes 17

---

## **Using Radio Buttons 19**

Standard Radio Buttons 19

Icon Radio Buttons 20

[Querying Button Matrices](#) 21

---

[Setting the Appearance of a Button's Border](#) 23

---

[Setting a Button's Title](#) 25

---

[Setting a Button's Image](#) 27

---

[Hiding a Button](#) 29

---

[Making a Button the Default Button](#) 31

---

[Setting a Button's Key Equivalent](#) 33

---

[Subclassing NSButton](#) 35

---

[Document Revision History](#) 37

---

# Listings

## **Using Radio Buttons 19**

---

Listing 1      Creating a radio-button matrix programmatically 19

## **Querying Button Matrices 21**

---

Listing 1      Querying a matrix object for the selected radio-button cell 21



# Introduction to Buttons

---

A button is a user interface object that sends an action message to a target when clicked

You should read this document to learn what are the different types of button provided by Cocoa and how you can use them.

## Organization of This Document

This document describes how to use a button. These articles give you basic information on the different types of button, and how you can customize buttons:

- [“How Buttons Work”](#) (page 9) describes how buttons work.
- [“Types of Button”](#) (page 11) describes the types of button, how they act, how it highlights when pressed and whether it shows its state.
- [“Using Push Buttons”](#) (page 15) explains how to use push buttons.
- [“Using Checkboxes”](#) (page 17) explains how to use checkboxes in matrix objects.
- [“Using Radio Buttons”](#) (page 19) explains how to use radio buttons in matrix objects.
- [“Querying Button Matrices”](#) (page 21) shows how to handle selections in radio buttons and checkboxes.
- [“Setting the Appearance of a Button’s Border”](#) (page 23) describes how to change the border’s shape and grading.
- [“Setting a Button’s Title”](#) (page 25) describes how to change the content and appearance of the button’s title.
- [“Setting a Button’s Image”](#) (page 27) describes how to change the content and appearance of the button’s image.
- [“Hiding a Button”](#) (page 29) describes how to make a button invisible or have it show its border only when the mouse is over it.
- [“Setting a Button’s Key Equivalent”](#) (page 33) describes how to attach a key equivalent to a button.
- [“Subclassing NSButton”](#) (page 35) describes how to subclass `NSButton`.

The following articles in other documents also contain relevant information:

- [Using a Continuous Control](#) describes how to set up a button so it sends its action message repeatedly while being pressed.
- [Cell States](#) describes the three states a button can have: on, off, or mixed.





# How Buttons Work

---

Buttons follow the target-action design pattern. A button is a user interface object that sends an action message to a target when clicked. For more the design pattern, see *The Target-Action Paradigm* in *Cocoa Fundamentals Guide*.

Most of the button's work is handled by the `NSButtonCell` class. An `NSButtonCell` instance 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 button cell. The button cell can show that it's being pressed by highlighting in several ways—for example, a bordered button cell can appear pushed into the screen, or the image or title can change to an alternate form while the button cell is pressed.

An `NSButtonCell` object must work with an instance of a subclass of `NSControl`. If you need one button, such as a push button, use an `NSButton` object that contains a single `NSButtonCell` instance. If you need a group of related buttons, such as a group of switches or radio buttons, use an `NSMatrix` object that contains several `NSButtonCell` instances.

`NSButton` and `NSMatrix` both provide a control view. However, while `NSMatrix` requires you to access the `NSButtonCell` objects 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` object'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).



# Types of Button

---

The button type determines how the button acts: how it highlights when pressed and whether it shows its state. The button types fall into three categories:

- [“Push Buttons”](#) (page 11)
- [“Sticky Buttons”](#) (page 12)
- [“Radio Buttons and Checkboxes”](#) (page 12)

You set the button type with `setButtonType:`.

## Push Buttons

These buttons are most useful for triggering actions, since they don't show their state. They change their appearance when the mouse button is held down and return to their original appearance when the mouse button is released.

- To let `NSButton` control the appearance of a button being pressed, use `NSMomentaryPushInButton` (called “Momentary Push” in Interface Builder’s Button Inspector). When the mouse button is down, the button appears to be pushed in.

Here’s an example of a `NSMomentaryPushInButton` button with a bezel style of `NSPushButtonBezelStyle`, in both the normal and the pushed-in appearance:



And here’s a sample of a `NSMomentaryPushInButton` button with a bezel style of `NSThickerSquareBezelStyle`. The bezel styles `NSRegularSquareBezelStyle` and `NSThickSquareBezelStyle` are similar.



- To control the appearance of a button being pressed yourself, use `NSMomentaryChangeButton` (called “Momentary Change” in Interface Builder’s Button Inspector). When the mouse button is down, it displays the alternate image and alternate title. When the mouse button is released, it displays the normal image and title. If you haven’t set an alternate image or name for the button, its appearance doesn’t change.

## Sticky Buttons

These buttons show their state and appear to stick when pressed. After you click one, it appears to stay pressed until you click it again.

- To let `NSButton` control the appearance of a pressed button, use `NSPushOnPushOffButton` (called “Push On/Push Off” in Interface Builder’s Button Inspector). After being clicked once, the button appears to be pushed in. After being clicked again, the button appears to pop back up. The popped-up appearance is for the off state (`NSOffState`), and the pressed-in appearance is for the on and mixed states (`NSOnState` and `NSMixedState`). This is useful for a button that displays the state of something in your application (for example, a button that displays whether the selected text is in boldface).
- To control the appearance of a button being pressed, use `NSToggleButton` (called “Toggle” in Interface Builder’s Button Inspector). After being clicked once, the button displays its alternate image and title. After being clicked again, the button displays its normal image and title. If there’s no alternate image or title, the button’s appearance doesn’t change. The normal image and title are for the off state (`NSOffState`), and the alternate image and title are for the on and mixed states (`NSOnState` and `NSMixedState`). This is useful for a button that toggles between two actions (for example, Stop and Start).

If you want a button to display different appearances for all three states, you must subclass `NSButton`.

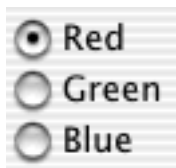
## Radio Buttons and Checkboxes

These buttons display the state of something in your application. They’re specialized versions of `NSToggleButton` which have system-defined images.

- To choose between two choices, use `NSSwitchButton`, which looks like a check box. This type of button is available as a separate palette item in Interface Builder.



- To choose among more than two choices, use a matrix of `NSRadioButtons`. The matrix and the radio buttons work together to make sure that only one button is on at a time. This type of button is available as a separate palette item in Interface Builder.



Changing the images used for these buttons could lead to unpredictable results. If you want a switch or radio button with a customized appearance, either customize a toggle button or subclass `NSButton`.

Although checkboxes and radio buttons can display different images for all three states, other types of buttons cannot.



# Using Push Buttons

---

A push button performs the action described by the button's name. Generally, it's a rounded rectangle that contains its name inside. For example, this button might appear in a dialog box that finds text in a document:



It's easiest to create a push button in Interface Builder. You can also create one programmatically by creating an `NSButton` instance with a type of `NSMomentaryPushInButton`, an image position of `NSNoImage`, and a border of `NSRoundedBezelStyle`.

You can also have a push button that's an icon button; that is, one that's primarily identified by its icon and has little or no text. It's rectangular, like this:



You can create an icon push button in either Interface Builder or programmatically. If you use Interface Builder, start with a regular push button. If you create it programmatically, create an `NSButton` instance, then set its type to `NSMomentaryPushInButton`, its image position to `NSImageOnly`, its bezel type to a square bezel type. Finally, set the image to what you want.

**Note:** A text button can have a small image on it, and an icon button can have a label. The important issue is what's most prominent. If it's the text, treat it as a text button. If it's the image, treat it as an icon button.

You can also have a push button that toggles between two states, with each state having its own title and image. For example, a button could toggle between Start and Stop. You can create one in the same way you create a regular push button with either Interface Builder or programmatically. Just change the button type to `NSToggleButton`. Then give the button an alternate title and image as well as a regular title and image. The button displays the regular title and image at first, then displays the alternate title and image after the user clicks it.





# Using Checkboxes

---

A checkbox displays the setting of something in your application. Another name for a checkbox is switch button. A checkbox is identified square with a line of text. If the button's off, the box is empty. If the button is on, the box has a checkmark in it. If the button is mixed-state, the box has a dash in it.

## Standard Checkboxes

It's easiest to create a checkbox in Interface Builder. You can also create one programmatically by creating an instance of `NSButton` with a type of `NSSwitchButton`.

Unlike a group of radio buttons, more than one item can be on in a group of checkboxes. This group of buttons displays that all of the selected characters are bold, none are italic, and some are underlined:



## Icon Checkboxes

You can also have a checkbox that's an icon button; that is, one that's primarily identified by its icon and has little or no text. If the button's off, it appears to be sticking out. If the button's on, it appears to be pressed in. (An icon button cannot display the mixed state.)

You can create an icon checkbox in either Interface Builder or programmatically. If you use Interface Builder, start with a push button. If you create it programmatically, create an instance of `NSButton`. Then change it by setting its type to `NSPushOnPushOffButton`, its image position to `NSImageOnly`, its bezel type to a square bezel type. Then set the image to what you want.



# Using Radio Buttons

---

A radio button displays the setting of something in your application and is part of a group in which only one button can be on at a time. Use a group of radio buttons to choose among several options which are mutually exclusive.

## Standard Radio Buttons

A standard radio button is a small circle followed by a line of text. If the button's off, the circle is empty. If the button is on, the circle is filled in. If the button is mixed-state, the circle has a dash in it.

For example, this group of buttons displays that all the selected objects are green:



And this group displays that some of the selected objects are red and some are green:



A group of radio buttons is implemented with an `NSMatrix` object that contains several `NSButtonCell` instances and has a tracking mode of `NSRadioModeMatrix`. Whenever one of the matrix's buttons is clicked, the matrix turns off the previously selected button and turns on the newly clicked one.

It's easiest to create a group of switch buttons in Interface Builder. You can also make one programmatically by allocating an `NSMatrix` object and initializing it (in an invocation of `initWithFrame:mode:prototype:numberOfRows:numberOfColumns:`) with a prototype cell and a tracking mode of `NSRadioModeMatrix`. For the prototype object, create a `NSButtonCell` object with a type of `NSRadioButton`. Listing 1 illustrates how you might do this.

### Listing 1 Creating a radio-button matrix programmatically

```
- (void)awakeFromNib {  
  
    NSButtonCell *prototype = [[NSButtonCell alloc] init];  
    [prototype setTitle:@"Watermelons"];  
    [prototype setButtonType:NSRadioButton];  
}
```

```
NSRect matrixRect = NSMakeRect(20.0, 20.0, 125.0, 125.0);
NSMatrix *myMatrix = [[NSMatrix alloc] initWithFrame:matrixRect
                    mode:NSRadioModeMatrix
                    prototype:(NSCell *)prototype
                    numberOfRows:3
                    numberOfColumns:1];
[[[typeField window] contentView] addSubview:myMatrix];
NSArray *cellArray = [myMatrix cells];
[[cellArray objectAtIndex:0] setTitle:@"Apples"];
[[cellArray objectAtIndex:1] setTitle:@"Oranges"];
[[cellArray objectAtIndex:2] setTitle:@"Pears"];
[prototype release];
[myMatrix release];
}
```

## Icon Radio Buttons

You can also have a radio button that's an icon button; that is, one that's primarily identified by its icon and has little or no text. If the button's off, it appears to be sticking in. If the button's on, it appears to be pressed in. (An icon button cannot display the mixed state.)

You can create a group of icon radio buttons in either Interface Builder or programmatically. If you use Interface Builder, start with a matrix of push buttons. If you create it programmatically, create an matrix of buttons. Then change the matrix's tracking mode to `NSRadioModeMatrix`. Change the buttons' types to `NSPushOnPushOffButton`, their image positions to `NSImageOnly`, their bezel types to a square bezel type. Finally set their images to what you want.

# Querying Button Matrices

---

A group of radio buttons or checkboxes is programmatically an `NSMatrix` object whose constituent objects are `NSButtonCell` objects. Matrix objects are a special kind of control. Each of its cells can have its own target object and action selector specified. Additionally, an `NSMatrix` may have its own target and action selector. (For more on target-action in relation to matrix objects, see *Matrix Programming Guide for Cocoa*.)

To find out which radio button or checkbox a user selected—at the moment he or she clicks it—you could specify a target and a different action selector for each cell in the matrix, and then implement the corresponding action method. However, a more efficient way to query the current selection in matrices of radio buttons or checkboxes is to implement target-action for the `NSMatrix` object itself, and in the action method determine which cell (or cells) are now selected. The `NSMatrix` methods for this are `selectedCell` and `selectedCells`.

Listing 1 shows an implementation of an action method that responds to a selection in a matrix of radio buttons.

## Listing 1 Querying a matrix object for the selected radio-button cell

```
- (IBAction)findSelectedButton:(id)sender { // sender is NSMatrix object
    NSButtonCell *selCell = [sender selectedCell];
    NSLog(@"Selected cell is %d", [selCell tag]);
}
```

This code snippet illustrates another technique you can apply when handling selection of cells in matrices. You can assign numeric tags to each cell in a matrix to identify it, and then query for those tag values when handling selections.



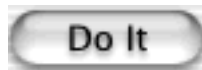
# Setting the Appearance of a Button's Border

---

You can control what your button's border looks like by changing its shape and shading. Note that the border doesn't appear if `isBordered` returns `NO`. Use `setBordered:` to change its value.

To change the border's shape, change the button's bezel type with `setBezelStyle:`. There are two major categories of bezel type.

- If your button is identified mainly by text, use `PushButtonBezelStyle`. It uses the appropriate bezel style for a text button, which is a rounded rectangle, like this:



- If your button is identified mainly by an icon, use `NSRegularSquareBezelStyle`, `NSThickSquareBezelStyle`, or `NSThickerSquareBezelStyle`. These types use a rectangular button with a border. The small style has a 2-pixel border; the medium style, 3-pixel; the large style, 4-pixel. The three types are shown here:







# Setting a Button's Title

---

A button can have two titles associated with it: normal and alternate. If the button type is `NSMomentaryPushInButton`, `NSPushOnPushOffButton`, `NSMomentaryLightButton`, or `NSOnOffButton`, only the normal title is ever displayed. If the button type is `NSMomentaryChangeButton` or `NSToggleButton`, the normal title is displayed when the button's state is off (`NSOffState`) and the alternate title is displayed when the button's state is on or mixed (`NSOnState` or `NSMixedState`). If you want a button to display different titles for all three states, you must subclass `NSButton`.

If you want the title to contain plain text, use `setTitle:` to set the normal title and `setAlternateTitle:` to set the alternate title. If you want the title to contain styled text (for example, italics or bold), use `setAttributedTitle:` and `setAttributedAlternateTitle:`.

To set how the title is positioned relative to the button's image, use `setImagePosition:`, described in ["Setting a Button's Image"](#) (page 27). If there is no image, the title is centered horizontally and vertically within the button. If the title is above, below, or overlapping the image, the title is centered horizontally within the button. To hide the title, use `setImagePosition:` with an argument of `NSImageOnly`.

To set the title's font, send `setFont:` to the button's button cell.

If you want a button to display its title in a tag when the mouse is over it, you need to use Tool Tips.










# Setting a Button's Image

A button can have two images associated with it: normal and alternate. If the button type is `NSMomentaryPushInButton`, `NSPushOnPushOffButton`, `NSMomentaryLightButton`, or `NSOnOffButton`, only the normal image is ever displayed. If the button type is `NSMomentaryChangeButton` or `NSToggleButton`, the normal image is displayed when the button's state is off (`NSOffState`) and the alternate image is displayed when the button's state is on or mixed (`NSOnState` or `NSMixedState`). If you want a button to display different image for all three states, you must subclass `NSButton`. (Although switch and radio buttons can display different images for all three states, there is no public interface for this feature.)

To set the normal image, use `setImage:`. To set the alternate image, use `setAlternateImage:`.

**Note:** If a button is a checkbox or a radio button, do not change its images. The images for these buttons are system-defined and changing them could lead to unpredictable results. If you want a switch or radio button with a customized appearance, either customize a toggle button (a button whose type is `NSToggleButton`) or subclass `NSButton`.

To set the position for a button's image, use `setImagePosition:`, with one of the following values below. The default is `NSNoImage`.

<code>NSNoImage</code>		<code>NSImageOnly</code>		<code>NSImageOverlaps</code>	
<code>NSImageLeft</code>		<code>NSImageRight</code>			
<code>NSImageBelow</code>		<code>NSImageAbove</code>			



# Hiding a Button

---

There are two ways to hide a button from view: It can be completely transparent, or it can display its border only when the mouse is over it.

- To make a button transparent, use `setTransparent:`. A transparent button tracks the mouse and sends its action, but doesn't draw itself. This 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.
- To have a button display its border only if it's active and the mouse is over it, use `setShowsBorderOnlyWhileMouseInside:`. The rest of the button's components are always drawn. Here's an example of some buttons that show their borders only when the mouse is over them:





# Making a Button the Default Button

---

If a button has the `NSRoundedBezelStyle` bezel type, you can mark it as the default button. A default button pulses, and its action message is invoked when the user presses Return. It looks like this:



To mark a button as the default, set its key equivalent to Return with `setKeyEquivalent:`, like this:

```
[myButton setKeyEquivalent:@"\r"];
```

You can also set the button's key equivalent in Interface Builder, as described in ["Setting a Button's Key Equivalent"](#) (page 33).

The default button has a thick outline drawn around it, outside the button's border; your interface design should account for that extra space.





# Setting a Button's Key Equivalent

---

A button can have a key equivalent, so that when the user presses that key, the button responds as though it's been clicked.

Note that if you set the key equivalent to Return, that button becomes the default button.

You typically set a button's key equivalent in Interface Builder. To do so, select the button and open the attributes pane of the inspector. Disclose the attributes for the button, click in the Key Equiv. field, and type the key or key combination you want to associate with the button. (You remove the key equivalent by pressing Clear.)

To set the key equivalent programmatically, use `setKeyEquivalent:` with the character. For example, to set it to Return, use:

```
[myButton setKeyEquivalent:@"\r"];
```

To set the button's key equivalent to non-print character, you can use the key constants defined by `NSResponder`, as in the following example, which sets a button's key equivalent to the left arrow key.

```
unichar arrowKey = NSLeftArrowFunctionKey;  
[button setKeyEquivalent:[NSString stringWithCharacters:&arrowKey length:1]];
```



# Subclassing 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 `cellClass:` method, as described in Subclassing NSControl.



# Document Revision History

---

This table describes the changes to *Button Programming Topics for Cocoa*.

Date	Notes
2008-10-15	Added "Querying Button Matrices" plus a example of programmatically creating a matrix of radio buttons. Also removed "Deprecated Buttons" from "Types of Buttons".
2007-12-11	Renamed "Using Switch Buttons" to "Using Checkboxes."
2006-09-05	Made a small change in the organization of information. Changed title from "Buttons."
	Moved information on making a button the default into a separate article.
	Fixed typos.
2003-01-15	Removed obsolete references to mnemonics.
2002-11-12	Revision history was added to existing topic. It will be used to record changes to the content of the topic.

