

[UIKit](#) / UIButton

Class

UIButton

A control that executes your custom code in response to user interactions.

iOS 2.0+ | iPadOS 2.0+ | Mac Catalyst 13.1+ | tvOS | visionOS 1.0+

```
@MainActor
class UIButton
```

Mentioned in

- 📄 [Choosing a user interface idiom for your Mac app](#)
- 📄 [Adding user-focusable elements to a tvOS app](#)
- 📄 [Attaching gesture recognizers to UIKit controls](#)
- 📄 [Supporting VoiceOver in your app](#)

Overview

When you tap a button, or select a button that has focus, the button performs any actions attached to it. You communicate the purpose of a button using a text label, an image, or both. The appearance of buttons is configurable, so you can tint buttons or format titles to match the design of your app. You can add buttons to your interface programmatically or using Interface Builder.

Button  

When adding a button to your interface, perform the following steps:

- Set the type of the button at creation time.
- Supply a title string or image; size the button appropriately for your content.

- Connect one or more action methods to the button.
- Set up Auto Layout rules to govern the size and position of the button in your interface.
- Provide accessibility information and localized strings.

Important

An app built with Mac Catalyst running in macOS 11 throws an exception when calling a button's `addGestureRecognizer(_ :)` method when `buttonType` is `UIButton.ButtonType.system` and the user interface idiom is `UIUserInterfaceIdiom.mac`.

Respond to button taps

Buttons use the target-action design pattern to notify your app when the user taps the button. Rather than handle touch events directly, you assign action methods to the button and designate which events trigger calls to your methods. At runtime, the button handles all incoming touch events and calls your methods in response.

You connect a button to your action method using the `addTarget(_ :action:for:)` method or by creating a connection in Interface Builder. The signature of an action method takes one of three forms, as shown in the following code. Choose the form that provides the information that you need to respond to the button tap.

Swift Objective-C

```
@IBAction func doSomething()
@IBAction func doSomething(sender: UIButton)
@IBAction func doSomething(sender: UIButton, forEvent event: UIEvent)
```

Configure a button's appearance

A button's type defines its basic appearance and behavior. You specify the type of a button at creation time using the `init(type:)` method or in your storyboard file. After creating a button, you can't change its type. The most commonly used button types are the Custom and System types, but use the other types when appropriate.

Note

To configure the appearance of all buttons in your app, use the appearance proxy object. The `UIButton` class implements the `appearance()` class method, which you can use to fetch the appearance proxy for all buttons in your app.

Configure button states

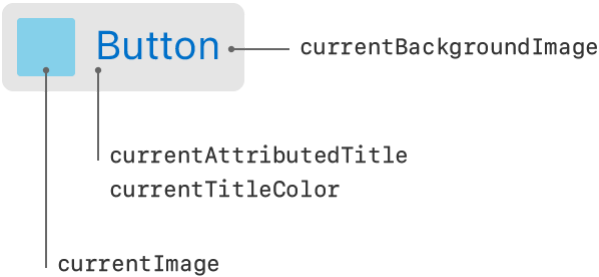
Buttons have five states that define their appearance: default, highlighted, focused, selected, and disabled. When you add a button to your interface, it's in the default state initially, which means the button is enabled and the user isn't interacting with it. As the user interacts with the button, its state changes to the other values. For example, when the user taps a button with a title, the button moves to the highlighted state.

When configuring a button either programmatically or in Interface Builder, you specify attributes for each state separately. In Interface Builder, use the State Config control in the Attributes inspector to choose the appropriate state and then configure the other attributes. If you don't specify attributes for a particular state, the `UIButton` class provides a reasonable default behavior. For example, a disabled button is normally dimmed and doesn't display a highlight when tapped. Other properties of this class, such as the `adjustsImageWhenHighlighted` and `adjustsImageWhenDisabled` properties, let you alter the default behavior in specific cases.

Provide content

The content of a button consists of a title string or image that you specify. The content you specify is used to configure the `UILabel` and `UIImageView` object managed by the button itself. You can access these objects using the `titleLabel` or `imageView` properties and modify their values directly. The methods of this class also provide a convenient shortcut for configuring the appearance of your string or image.

Normally, you configure a button using either a title or an image and size the button accordingly. Buttons can also have a background image, which is positioned behind the content you specify. It's possible to specify both an image and a title for buttons, which results in the appearance shown in the following image. You can access the current content of a button using the indicated properties.



When setting the content of a button, you must specify the title, image, and appearance attributes for each state separately. If you don't customize the content for a particular state, the button uses the values associated with the Default state and adds any appropriate customizations. For example, in the highlighted state, an image-based button draws a highlight on top of the default image if no custom image is provided.

Customize tint color

You can specify a custom button tint using the `tintColor` property. This property sets the color of the button image and text. If you don't explicitly set a tint color, the button uses its superview's tint color.

Specify edge insets

Use insets to add or remove space around the content in your custom or system buttons. You can specify separate insets for your button's title (`titleEdgeInsets`), image (`imageEdgeInsets`), and both the title and image together (`contentEdgeInsets`). When applied, insets affect the corresponding content rectangle of the button, which the Auto Layout engine uses to determine the button's position.

There should be no reason for you to adjust the edge insets for info, contact, or disclosure buttons.

Configure button attributes in Interface Builder

The following table lists the core attributes that you configure for buttons in Interface Builder.

Attribute	Description
Type	The button type. This attribute determines the default settings for many other button attributes. The value of this attribute can't be changed at runtime, but you can access it using the <code>buttonType</code> property.
State Config	The state selector. After selecting a value in this control, changes to the button's attributes apply to the specified state.
Title	The button's title. You can specify a button's title as a plain string or attributed string.
(Title Font and Attributes)	The font and other attributes to apply to the button's title string. The specific configuration options depends on whether you specified a plain string or attributed string for the button's title. For a plain string, you can customize the

Attribute	Description
	font, text color, and shadow color. For an attributed string, you can specify alignment, text direction, indentation, hyphenation, and many other options.
Image	The button's foreground image. Typically, you use template images for a button's foreground, but you may specify any image in your Xcode project.
Background	The button's background image. The background image is displayed behind its title and foreground image.

The following table lists attributes that affect the button's appearance.

Attribute	Description
Shadow Offset	The offsets and behavior of the button's shadow. Shadows affect title strings only. Enable the Reverses on Highlight option to change the highlighting of the shadow when the button state changes to or from the highlighted state. Configure the offsets programmatically using the <code>shadowOffset</code> property of the button's <code>titleLabel</code> object. Configure the highlighting behavior using the <code>reversesTitleShadowWhenHighlighted</code> property.
Drawing	The drawing behavior of the button. When the Shows Touch On Highlight (<code>showsTouchWhenHighlighted</code>) option is enabled, the button adds a white glow to the part of a button that the user touches. When the Highlighted Adjusts Image (<code>adjustsImageWhenHighlighted</code>) option is enabled, button images get darker when it's in the highlighted state. When the Disabled Adjusts Image (<code>adjustsImageWhenDisabled</code>) option is enabled, the image is dimmed when the button is disabled.
Line Break	The line breaking options for the button's text. Use this attribute to define how the button's title is modified to fit the available space.

The following table lists the edge inset attributes for buttons. Use edge inset buttons to alter the rectangle for the button's content.

Attribute	Description
Edge	The edge insets to configure. You can specify separate edge insets for the button's overall content, its title, and its image.

Attribute	Description
Inset	The inset values. Positive values shrink the corresponding edge, moving it closer to the center of the button. Negative values expand the edge, moving it away from the center of the button. Access these values at runtime using the contentEdgeInsets , titleEdgeInsets , and imageEdgeInsets properties.

For information about the button's inherited Interface Builder attributes, see [UIControl](#) and [UIView](#).

Support localization

To internationalize a button, specify a localized string for the button's title text. (You may also localize a button's image as appropriate.)

When using storyboards to build your interface, use Xcode's base internationalization feature to configure the localizations your project supports. When you add a localization, Xcode creates a strings file for that localization. When configuring your interface programmatically, use the system's built-in support for loading localized strings and resources. For more information about internationalizing your interface, see [Internationalization and Localization Guide](#).

Make buttons accessible

Buttons are accessible by default. The default accessibility traits for a button are Button and User Interaction Enabled.

The accessibility label, traits, and hint are spoken back to the user when VoiceOver is enabled on a device. The button's title overwrites its accessibility label; even if you set a custom value for the label, VoiceOver speaks the value of the title. VoiceOver speaks this information when a user taps the button once. For example, when a user taps the Options button in Camera, VoiceOver speaks the following:

- "Options. Button. Shows additional camera options."

For more information about making iOS controls accessible, see the accessibility information in [UIControl](#). For general information about making your interface accessible, see [Accessibility Programming Guide for iOS](#).

Topics

Creating buttons

```
init(frame: CGRect)
```

Creates a new button with the specified frame.

```
convenience init(frame: CGRect, primaryAction: UIAction?)
```

Creates a new button with the specified frame, registers the primary action event, and sets the title and image to the action's title and image.

```
init?(coder: NSCoder)
```

Creates a new button with data in an unarchiver.

Creating buttons of a specific type

```
convenience init(type: UIButton.ButtonType)
```

Creates and returns a new button of the specified type.

```
convenience init(type: UIButton.ButtonType, primaryAction: UIAction?)
```

Creates a new button with the specified type, registers the primary action event, and sets the title and image to the action's title and image.

```
enum ButtonType
```

Specifies the style of a button.

Creating system buttons

```
class func systemButton(with: UIImage, target: Any?, action: Selector?)  
-> Self
```

Creates and returns a system type button with specified image, target, and action.

Creating buttons from a configuration object

```
convenience init(configuration: UIButton.Configuration, primaryAction:  
UIAction?)
```

Creates a new button with the specified configuration and registers the primary action event.

```
struct Configuration
```

A configuration that specifies the appearance and behavior of a button and its contents.

Managing the appearance with a configuration object

```
var configuration: UIButton.Configuration?
```

The configuration for the button's appearance.

```
var automaticallyUpdatesConfiguration: Bool
```

A Boolean value that determines whether the button configuration changes when button's state changes.

```
func setNeedsUpdateConfiguration()
```

Requests the system update the button configuration.

```
func updateConfiguration()
```

Updates the button configuration in response to a button state change.

```
var configurationUpdateHandler: UIButton.ConfigurationUpdateHandler?
```

A closure that executes when the button state changes.

```
 typealias ConfigurationUpdateHandler
```

A closure to update the configuration of a button.

Managing the title

```
var titleLabel: UILabel?
```

A view that displays the value of the currentTitle property for a button.

```
func title(for: UIControl.State) -> String?
```

Returns the title associated with the specified state.

```
func setTitle(String?, for: UIControl.State)
```

Sets the title to use for the specified state.

```
func attributedTitle(for: UIControl.State) -> NSAttributedString?
```

Returns the styled title associated with the specified state.

```
func setAttributedTitle(NSAttributedString?, for: UIControl.State)
```

Sets the styled title to use for the specified state.

```
func titleColor(for: UIControl.State) -> UIColor?
```

Returns the title color used for a state.

```
func setTitleColor(UIColor?, for: UIControl.State)
```

Sets the color of the title to use for the specified state.

```
func titleShadowColor(for: UIControl.State) -> UIColor?
```

Returns the shadow color of the title used for a state.

```
func setTitleShadowColor(UIColor?, for: UIControl.State)
```

Sets the color of the title shadow to use for the specified state.

Managing images and tint color

```
func backgroundImage(for: UIControl.State) -> UIImage?
```

Returns the background image used for a button state.

```
func image(for: UIControl.State) -> UIImage?
```

Returns the image used for a button state.

```
func setBackgroundImage(UIImage?, for: UIControl.State)
```

Sets the background image to use for the specified button state.

```
func setImage(UIImage?, for: UIControl.State)
```

Sets the image to use for the specified state.

```
func preferredSymbolConfigurationForImage(in: UIControl.State) ->  
UIImage.SymbolConfiguration?
```

Returns the preferred symbol configuration for a button state.

```
func setPreferredSymbolConfiguration(UIImage.SymbolConfiguration?, for  
ImageIn: UIControl.State)
```

Sets the preferred symbol configuration for a button state.

```
var tint_color: UIColor!
```

The tint color to apply to the button title and image.

Specifying the role

```
var role: UIButton.Role
```

The role of the button.

```
enum Role
```

Constants that describe the role of the button.

Specifying the behavioral style

Determine how the button appears and behaves in your app built with Mac Catalyst.

`var behavioralStyle: UIBehavioralStyle`

The style that determines how the button behaves.

`var preferredBehavioralStyle: UIBehavioralStyle`

The preferred behavioral style.

`enum UIBehavioralStyle`

Constants that indicate how a control behaves in apps built with Mac Catalyst.

Getting the current state

`var buttonType: UIButton.ButtonType`

The button type.

`var currentTitle: String?`

The current title that is displayed on the button.

`var currentAttributedTitle: NSAttributedString?`

The current styled title that is displayed on the button.

`var currentTitleColor: UIColor`

The color used to display the title.

`var currentTitleShadowColor: UIColor?`

The color of the title's shadow.

`var currentImage: UIImage?`

The current image displayed on the button.

`var currentBackgroundImage: UIImage?`

The current background image displayed on the button.

`var currentPreferredSymbolConfiguration: UIImage.SymbolConfiguration?`

The current symbol size, style, and weight.

`var imageView: UIImageView?`

The button's image view.

`var subtitleLabel: UILabel?`

The label that displays the text of the subtitle.

Supporting pointer interactions

`var isPointerInteractionEnabled: Bool`

A Boolean that enables pointer interaction.

`var isHovered: Bool`

A Boolean value that indicates whether a pointer effect is active.

`var pointerStyleProvider: UIButton.PointerStyleProvider?`

A closure that returns the pointer style to use when the pointer hovers over the button.

`typealias PointerStyleProvider`

A type alias defining a closure that returns a pointer style to apply to a button.

`typealias UIButtonPointerStyleProvider`

A type alias defining a block that returns a pointer style to apply to a button.

Supporting menu and toggle buttons

Configure the button to display a pull-down menu, pop-up menu, or toggle a selection.

`var menu: UIMenu?`

A menu that the button displays.

`var isHeld: Bool`

A Boolean value that indicates whether the button menu is visible.

`var changesSelectionAsPrimaryAction: Bool`

A Boolean value that indicates whether the button tracks a selection, either through a menu or a toggle.

`var preferredMenuElementOrder: UIContextMenuConfiguration.ElementOrder`

The preferred menu-element ordering strategy for the menu.

Deprecated

 Deprecated symbols

Symbols that buttons no longer support.

Relationships

Inherits From

UIControl

Conforms To

CALayerDelegate

CVarArg

Copyable

CustomDebugStringConvertible

CustomStringConvertible

Equatable

Hashable

NSCoding

NSObjectProtocol

NSTouchBarProvider

Sendable

SendableMetatype

UIAccessibilityContentSizeCategoryImageAdjusting

UIAccessibilityIdentification

UIActivityItemsConfigurationProviding

UIAppearance

UIAppearanceContainer

UIContextMenuInteractionDelegate

UICoordinateSpace

UIDynamicItem

UIFocusEnvironment

UIFocusItem

UIFocusItemContainer

UILargeContentViewerItem

UIPasteConfigurationSupporting

UIPopoverPresentationControllerSourceItem

UIResponderStandardEditActions

UISpringLoadedInteractionSupporting

UITraitChangeObservable

UITraitEnvironment

UIUserActivityRestoring

See Also

Controls



Responding to control-based events using target-action

Handle user input by connecting buttons, sliders, and other controls to your app's code using the target-action design pattern.

`class UIControl`

The base class for controls, which are visual elements that convey a specific action or intention in response to user interactions.

`class UIColorWell`

A control that displays a color picker.

`class UIDatePicker`

A control for inputting date and time values.

`class UIPageControl`

A control that displays a horizontal series of dots, each of which corresponds to a page in the app's document or other data-model entity.

`class UISegmentedControl`

A horizontal control that consists of multiple segments, each segment functioning as a discrete button.

`class UISlider`

A control for selecting a single value from a continuous range of values.

`class UIStepper`

A control for incrementing or decrementing a value.

`class UISwitch`

A control that offers a binary choice, such as on/off.