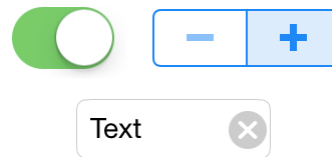


About Controls

On This Page

A control is a communication tool between a user and an app. Controls convey a particular action or intention to the app through user interaction, and can be used to manipulate content, provide user input, navigate within an app, or execute other pre-defined actions.

Controls are simple, straightforward, and familiar to users because they appear throughout many iOS apps. The `UIControl` class is the base class for all controls on iOS, and defines the functionality that is common to all controls. You should never use it directly; instead, use one of its subclasses. Each subclass of `UIControl` defines appearance, behavior, and intended usage specific to that particular control. By using controls carefully and consistently in your app, you can convey to users what they have the freedom and ability to do within the app.

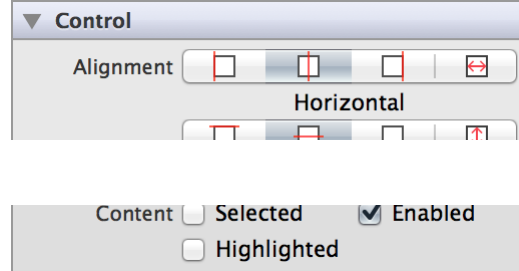


Purpose. Controls allow users to:

- Interact with an app
- Manipulate or edit app content
- Convey user intent to the app in a straightforward way

Implementation. Controls are implemented in the `UIControl` class and discussed in [UIControl Class Reference](#).

Configuration. Configure controls in Interface Builder, in the Control section of the Attributes Inspector. A few configurations cannot be made through the Attributes Inspector, so you must make them programmatically. You can set other configurations programmatically, too, if you prefer.



On This Page

Content of Controls

Each subclass of [UIControl](#) has different content or values that you can set. To learn about setting content for a particular control, read its corresponding chapter:

- Buttons
- Date Pickers
- Page Controls
- Segmented Controls
- Text Fields
- Sliders
- Steppers
- Switches

Behavior of Controls

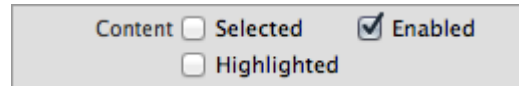
Control States

A **control state** describes the current interactive state of a control: normal, selected, enabled, or highlighted. A control can have more than one state at a time, and you can change a control's state at any point. For a full listing of control states, see [UIControlState](#).

When a user interacts with a control, the control's state changes appropriately. You can configure controls to have different appearances for different states to provide users with feedback about which state the control is in. For example, you might configure a button to display one image when it is in its normal state and a different image when it is highlighted.

On This Page

The fastest way to configure the initial state of a control is by using the Attributes Inspector:



When a control is enabled, a user can interact with it. When a control is disabled, it appears grayed out and does not respond to user interaction. Controls are enabled by default; to disable a control, uncheck the “Enabled” (enabled) box in the Attributes Inspector.

A control enters a temporary highlighted state when a touch enters and exits during tracking and when there is a touch up event. A highlighted state is temporary. You can customize the highlighted appearance of some controls, such as buttons. Controls are not highlighted by default; to set a control's initial state to highlighted, check the “Highlighted” (highlighted) box in the Attributes Inspector.

When a user taps on a control, the control enters the selected state. For many controls, this state has no effect on behavior or appearance. However, some subclasses may have different appearance depending on their selected state. For example, [UISegmentedControl](#) segments have a distinctly different appearance when selected. You can set a control to be selected using the “Selected” (selected) checkbox.

Control Events

A **control event** represents various physical gestures that users can make on controls, such as lifting a finger from a control, dragging a finger into a control, and touching down within a text field. For a full listing of control events, see [UIControlEvents](#).

Target-Action Mechanism

The **target-action mechanism** is a model for configuring a control to send an action message to a target object after a specific control event. For example, when a user interacts with a slider, it generates a [UIControlEventValueChanged](#) control event. You could use this event to update a label's text to the current

value of the slider. In this case, the sender is the slider, the control event is Value Changed, the action is updating the label's text, and the target is the controller file containing the label as an [IBOutlet](#).

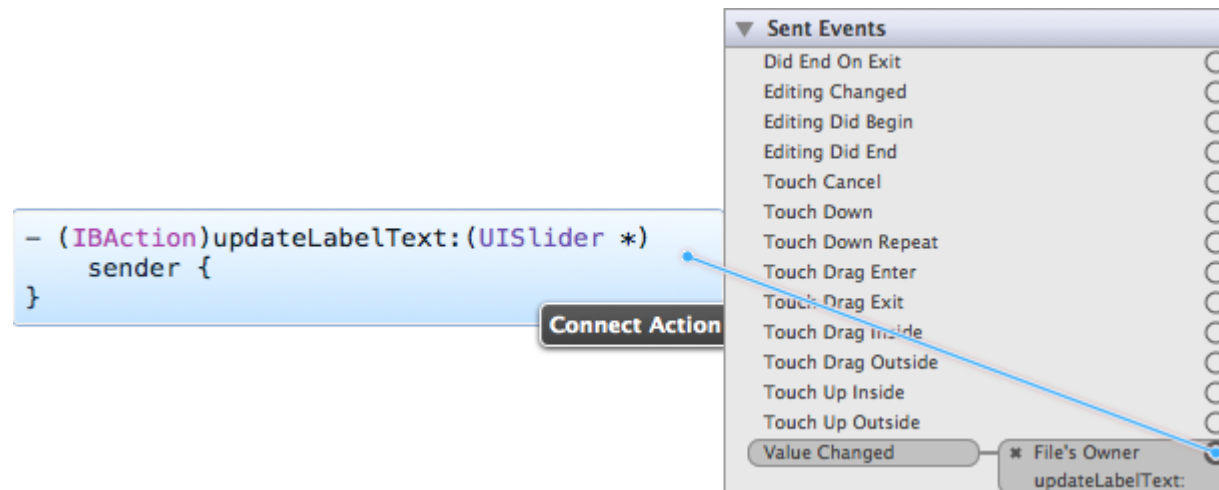
To create a relationship between the slider, the control event, the target, and the action, you can do one of two things:

On This Page

1. Call the `addTarget:action:forControlEvents:` method within your target file:

```
1 [self.mySlider addTarget:self
2                               action:@selector(myAction:)
3                               forControlEvents:UIControlEventValueChanged];
```

2. Use the Connections Inspector in Interface Builder to Control-drag the slider's Value Changed event to the action method in the target file.



3. Control-click the slider in Interface Builder, and drag its Value Changed event to the target object in your Storyboard. Select the appropriate action from the list of actions available for the target.

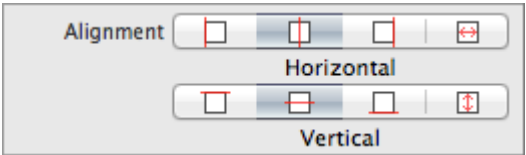
For more information, see [Target-Action in UIKit](#).

Appearance of Controls

Content Alignment

Certain controls—such as buttons and text fields—can contain custom images or text. For these controls, you can specify the **Horizontal Alignment** and **Vertical Alignment** options in Attributes Inspector. Using the horizontal alignment options, you can specify whether the content appears aligned with the left, center, or right of the control, or whether it fills the width of the control. Using the vertical alignment options, you can specify whether the content appears aligned with the top, center, or bottom of the control, or whether it fills the height of the control. This is a great tool for ensuring your content appears exactly where you want it to within your control (for example, centering text in a text field).

On This Page






















NOTE

These alignment options apply to the content of a control, not the control itself. For information about aligning controls with respect to other controls or views, see [Using Auto Layout with Controls](#).

Using Auto Layout with Controls

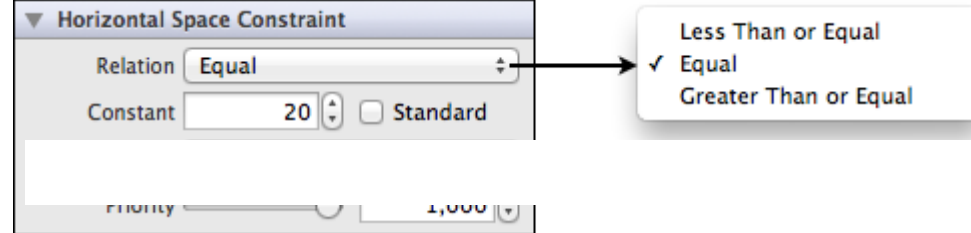
The auto layout system allows you to define layout constraints for user interface elements, such as views and controls. Constraints represent relationships between user interface elements. You can create auto layout constraints by selecting the appropriate element or group of elements and selecting an option from the menu in the bottom right corner of Xcode’s Interface Builder.

Auto layout contains two menus of constraints: pin and align. The Pin menu allows you to specify constraints that define some relationship based on a particular value or range of values. Some apply to the control itself (width) while others define relationships between elements (horizontal spacing). The following tables describes what each group of constraints in the auto layout menu accomplishes:

Constraint Name	Purpose
 Width  Height	Sets the width or height of a single element.
 Horizontal Spacing  Vertical Spacing	exactly two elements.
 Leading Space to Superview  Trailing Space to Superview  Top Space to Superview  Bottom Space to Superview	Sets the spacing from one or more elements to the leading, trailing, top, or bottom of their container view. Leading and trailing are the same as left and right in English, but the UI flips when localized in a right-to-left environment.
 Widths Equally  Heights Equally	Sets the widths or heights of two or more elements equal to each other.
 Left Edges  Right Edges  Top Edges  Bottom Edges	Aligns the left, right, top, or bottom edges of two or more elements.
 Horizontal Centers  Vertical Centers  Baselines	Aligns two or more elements according to their horizontal centers, vertical centers, or bottom baselines. Note that baselines are different from bottom edges. These values may not be defined for certain elements.
 Horizontal Center in Container  Vertical Center in Container	Aligns the horizontal or vertical centers of one or more elements with the horizontal or vertical center of their container view.

On This Page

The “Constant” value specified for any Pin constraints (besides Widths/Heights Equally) can be part of a “Relation.” That is, you can specify whether you want the value of that constraint to be equal to, less than or equal to, or greater than or equal to the value.



On This Page

For more information, see [Auto Layout Guide](#).

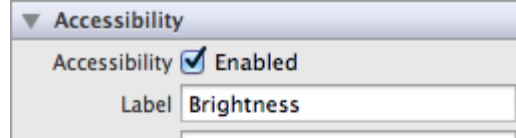
Making Controls Accessible

Controls are accessible by default. To be useful, an accessible user interface element must provide accurate and helpful information about its screen position, name, behavior, value, and type. This is the information VoiceOver speaks to users. Visually impaired users can rely on VoiceOver to help them use their devices.

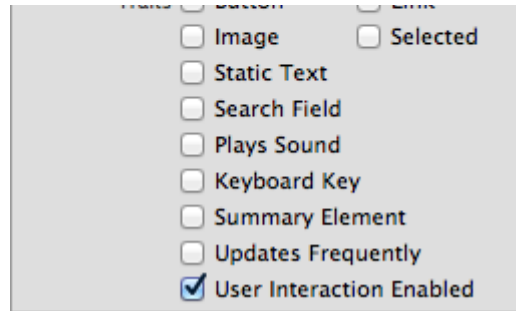
The iOS SDK contains a programming interface and tools that help you ensure that the user interface elements in your application are both accessible and useful. The UI Accessibility programming interface defines the following attributes:

- **Label.** A short, localized word or phrase that succinctly describes the control or view, but does not identify the element's type. Examples are “Add” or “Play.”
- **Traits.** A combination of one or more individual traits, each of which describes a single aspect of an element's state, behavior, or usage. For example, an element that behaves like a keyboard key and that is currently selected can be characterized by the combination of the Keyboard Key and Selected traits.
- **Hint.** A brief, localized phrase that describes the results of an action on an element. Examples are “Adds a title” or “Opens the shopping list.”
- **Frame.** The frame of the element in screen coordinates, which is given by the CGRect structure that specifies an element's screen location and size.
- **Value.** The current value of an element, when the value is not represented by the label. For example, the label for a slider might be “Speed,” but its current value might be “50%.”

Controls automatically provide value, frame, and default trait information. You can set a label, hint, and adjust the list of traits using the **Identity Inspector** in Interface Builder.



On This Page



For more information, see [Accessibility Programming Guide for iOS](#).