

[SwiftUI](#) / Controls and indicators

API Collection

Controls and indicators

Display values and get user selections.

Overview

SwiftUI provides controls that enable user interaction specific to each platform and context. For example, people can initiate events with buttons and links, or choose among a set of discrete values with different kinds of pickers. You can also display information to the user with indicators like progress views and gauges.



Use these built-in controls and indicators when composing custom views, and style them to match the needs of your app's user interface. For design guidance, see [Menus and actions](#), [Selection and input](#), and [Status](#) in the Human Interface Guidelines.

Topics

Creating buttons

`struct Button`

A control that initiates an action.

`func buttonStyle(_:_)`

Sets the style for buttons within this view to a button style with a custom appearance and standard interaction behavior.

`func buttonBorderShape(ButtonBorderShape) -> some View`

Sets the border shape for buttons in this view.

`func buttonRepeatBehavior(ButtonRepeatBehavior) -> some View`

Sets whether buttons in this view should repeatedly trigger their actions on prolonged interactions.

`var buttonRepeatBehavior: ButtonRepeatBehavior`

Whether buttons with this associated environment should repeatedly trigger their actions on prolonged interactions.

`struct ButtonBorderStyle`

A shape used to draw a button's border.

`struct ButtonRole`

A value that describes the purpose of a button.

`struct ButtonRepeatBehavior`

The options for controlling the repeatability of button actions.

`struct ButtonSizing`

The sizing behavior of Buttons and other button-like controls.

Creating special-purpose buttons

`struct EditButton`

A button that toggles the edit mode environment value.

`struct PasteButton`

A system button that reads items from the pasteboard and delivers it to a closure.

```
struct RenameButton
```

A button that triggers a standard rename action.

Linking to other content

```
struct Link
```

A control for navigating to a URL.

```
struct ShareLink
```

A view that controls a sharing presentation.

```
struct SharePreview
```

A representation of a type to display in a share preview.

```
struct TextFieldLink
```

A control that requests text input from the user when pressed.

```
struct HelpLink
```

A button with a standard appearance that opens app-specific help documentation.

Getting numeric inputs

```
struct Slider
```

A control for selecting a value from a bounded linear range of values.

```
struct Stepper
```

A control that performs increment and decrement actions.

```
struct Toggle
```

A control that toggles between on and off states.

```
func toggleStyle<S>(S) -> some View
```

Sets the style for toggles in a view hierarchy.

Choosing from a set of options

```
struct Picker
```

A control for selecting from a set of mutually exclusive values.

```
func pickerStyle<S>(S) -> some View
```

Sets the style for pickers within this view.

```
func horizontalRadioGroupLayout() -> some View
```

Sets the style for radio group style pickers within this view to be horizontally positioned with the radio buttons inside the layout.

```
func defaultWheelPickerItemHeight(CGFloat) -> some View
```

Sets the default wheel-style picker item height.

```
var defaultWheelPickerItemHeight: CGFloat
```

The default height of an item in a wheel-style picker, such as a date picker.

```
func paletteSelectionEffect(PaletteSelectionEffect) -> some View
```

Specifies the selection effect to apply to a palette item.

```
struct PaletteSelectionEffect
```

The selection effect to apply to a palette item.

Choosing dates

```
struct DatePicker
```

A control for selecting an absolute date.

```
func datePickerStyle<S>(S) -> some View
```

Sets the style for date pickers within this view.

```
struct MultiDatePicker
```

A control for picking multiple dates.

```
var calendar: Calendar
```

The current calendar that views should use when handling dates.

```
var timeZone: TimeZone
```

The current time zone that views should use when handling dates.

Choosing a color

```
struct ColorPicker
```

A control used to select a color from the system color picker UI.

Indicating a value

```
struct Gauge
```

A view that shows a value within a range.

```
func gaugeStyle<S>(S) -> some View
```

Sets the style for gauges within this view.

```
struct ProgressView
```

A view that shows the progress toward completion of a task.

```
func progressViewStyle<S>(S) -> some View
```

Sets the style for progress views in this view.

```
struct DefaultDateProgressLabel
```

The default type of the current value label when used by a date-relative progress view.

```
struct DefaultButtonLabel
```

The default label to use for a button.

Indicating missing content

```
struct ContentUnavailableView
```

An interface, consisting of a label and additional content, that you display when the content of your app is unavailable to users.

Providing haptic feedback

```
func sensoryFeedback<T>(SensoryFeedback, trigger: T) -> some View
```

Plays the specified feedback when the provided `trigger` value changes.

```
func sensoryFeedback(trigger:_:)
```

Plays feedback when returned from the feedback closure after the provided `trigger` value changes.

```
func sensoryFeedback<T>(SensoryFeedback, trigger: T, condition: (T, T) -> Bool) -> some View
```

Plays the specified feedback when the provided `trigger` value changes and the `condition` closure returns `true`.

```
struct SensoryFeedback
```

Represents a type of haptic and/or audio feedback that can be played.

Sizing controls

```
func controlSize(_:)
```

Sets the size for controls within this view.

```
var controlSize: ControlSize
```

The size to apply to controls within a view.

```
enum ControlSize
```

The size classes, like regular or small, that you can apply to controls within a view.

See Also

Views

☰ View fundamentals

Define the visual elements of your app using a hierarchy of views.

☰ View configuration

Adjust the characteristics of views in a hierarchy.

☰ View styles

Apply built-in and custom appearances and behaviors to different types of views.

☰ Animations

Create smooth visual updates in response to state changes.

☰ Text input and output

Display formatted text and get text input from the user.

☰ Images

Add images and symbols to your app's user interface.

☰ Menus and commands

Provide space-efficient, context-dependent access to commands and controls.

☰ Shapes

Trace and fill built-in and custom shapes with a color, gradient, or other pattern.

☰ Drawing and graphics

Enhance your views with graphical effects and customized drawings.