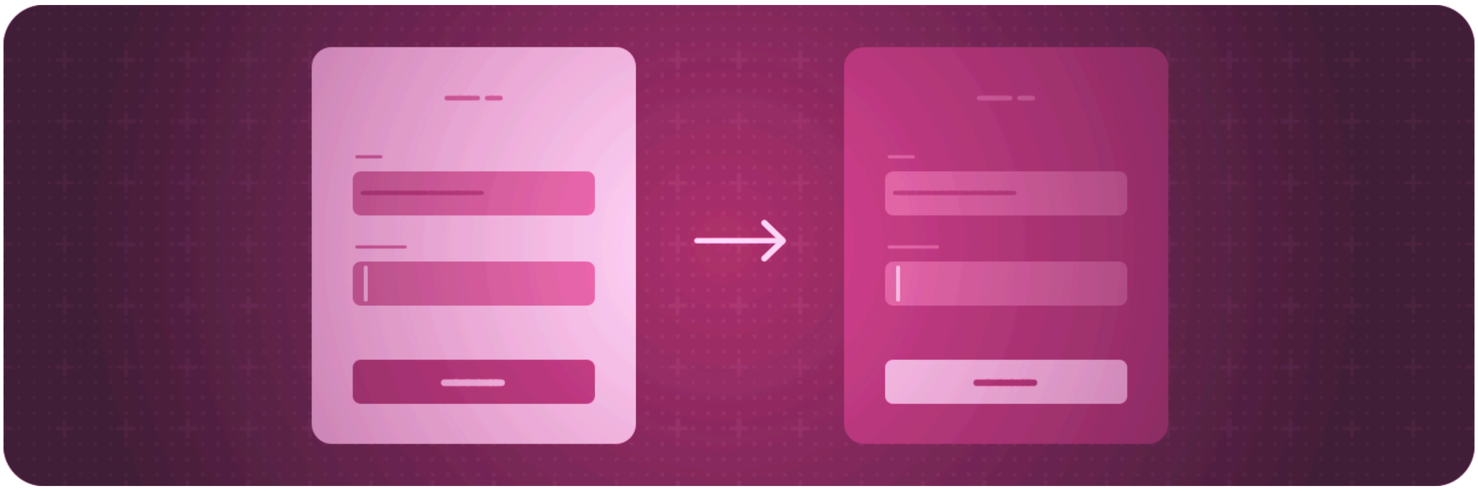
API Collection

View configuration

Adjust the characteristics of views in a hierarchy.

Overview

SwiftUI enables you to tune the appearance and behavior of views using view modifiers.



Many modifiers apply to specific kinds of views or behaviors, but some apply more generally. For example, you can conditionally hide any view by dynamically setting its opacity, display contextual help when people hover over a view, or request the light or dark appearance for a view.

Topics

Hiding views

```
func opacity(Double) -> some View
```

Sets the transparency of this view.

`func hidden() -> some View`

Hides this view unconditionally.

Hiding system elements

`func labelsHidden() -> some View`

Hides the labels of any controls contained within this view.

`func labelsVisibility(Visibility) -> some View`

Controls the visibility of labels of any controls contained within this view.

`var labelsVisibility: Visibility`

The labels visibility set by `labelsVisibility(_:)`.

`func menuIndicator(Visibility) -> some View`

Sets the menu indicator visibility for controls within this view.

`func statusBarHidden(Bool) -> some View`

Sets the visibility of the status bar.

`func persistentSystemOverlays(Visibility) -> some View`

Sets the preferred visibility of the non-transient system views overlaying the app.

`enum Visibility`

The visibility of a UI element, chosen automatically based on the platform, current context, and other factors.

Managing view interaction

`func disabled(Bool) -> some View`

Adds a condition that controls whether users can interact with this view.

`var isEnabled: Bool`

A Boolean value that indicates whether the view associated with this environment allows user interaction.

`func interactionActivityTrackingTag(String) -> some View`

Sets a tag that you use for tracking interactivity.

`func invalidatableContent(Bool) -> some View`

Mark the receiver as their content might be invalidated.

Providing contextual help

`func help(_:)`

Adds help text to a view using a text view that you provide.

Detecting and requesting the light or dark appearance

`func preferredColorScheme(ColorScheme?) -> some View`

Sets the preferred color scheme for this presentation.

`var colorScheme: ColorScheme`

The color scheme of this environment.

`enum ColorScheme`

The possible color schemes, corresponding to the light and dark appearances.

Getting the color scheme contrast

`var colorSchemeContrast: ColorSchemeContrast`

The contrast associated with the color scheme of this environment.

`enum ColorSchemeContrast`

The contrast between the app’s foreground and background colors.

Configuring passthrough

`func preferredSurroundingsEffect(SurroundingsEffect?) -> some View`

Applies an effect to passthrough video.

`struct SurroundingsEffect`

Effects that the system can apply to passthrough video.

`struct BreakthroughEffect`

Redacting private content

Designing your app for the Always On state

Customize your watchOS app’s user interface for continuous display.

```
func privacySensitive(Bool) -> some View
```

Marks the view as containing sensitive, private user data.

```
func redacted(reason: RedactionReasons) -> some View
```

Adds a reason to apply a redaction to this view hierarchy.

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

Removes any reason to apply a redaction to this view hierarchy.

```
var redactionReasons: RedactionReasons
```

The current redaction reasons applied to the view hierarchy.

```
var isSceneCaptured: Bool
```

The current capture state.

```
struct RedactionReasons
```

The reasons to apply a redaction to data displayed on screen.

See Also

Views

View fundamentals

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

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.

Controls and indicators

Display values and get user selections.

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.