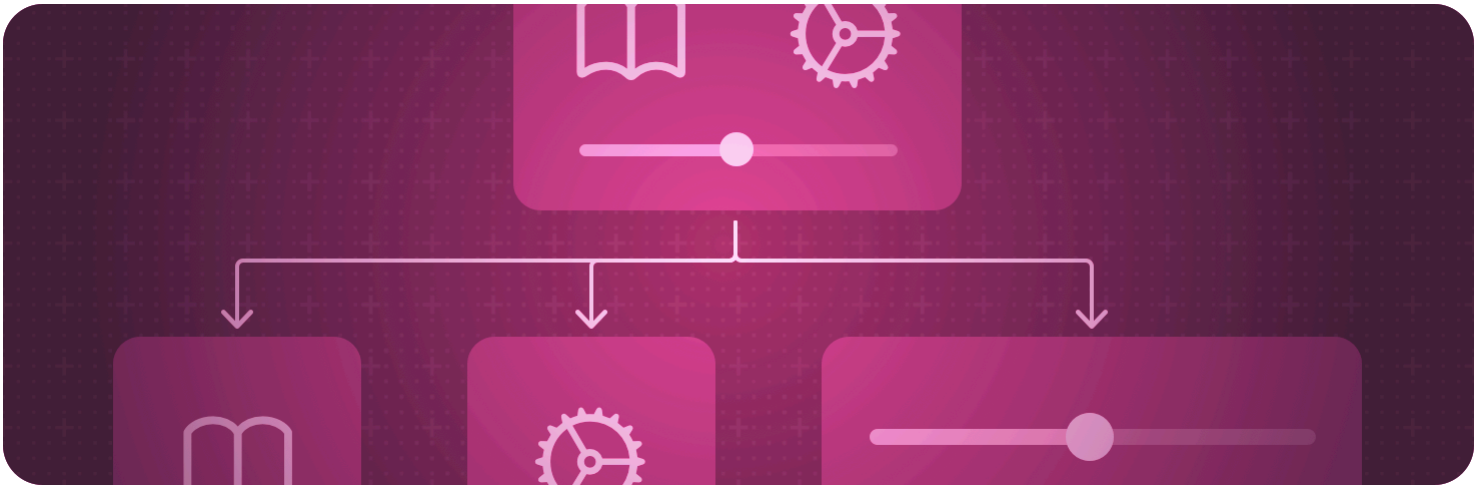
API Collection

# View fundamentals

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

## Overview

Views are the building blocks that you use to declare your app's user interface. Each view contains a description of what to display for a given state. Every bit of your app that's visible to the user derives from the description in a view, and any type that conforms to the `View` protocol can act as a view in your app.



Compose a custom view by combining built-in views that SwiftUI provides with other custom views that you create in your view's `body` computed property. Configure views using the view modifiers that SwiftUI provides, or by defining your own view modifiers using the `ViewModifier` protocol and the `modifier(_:)` method.

## Topics

## Creating a view

Declaring a custom view

Define views and assemble them into a view hierarchy.

`protocol View`

A type that represents part of your app's user interface and provides modifiers that you use to configure views.

`struct ViewBuilder`

A custom parameter attribute that constructs views from closures.

## Modifying a view

Configuring views

Adjust the characteristics of a view by applying view modifiers.

Reducing view modifier maintenance

Bundle view modifiers that you regularly reuse into a custom view modifier.

`func modifier<T>(T) -> ModifiedContent<Self, T>`

Applies a modifier to a view and returns a new view.

`protocol ViewModifier`

A modifier that you apply to a view or another view modifier, producing a different version of the original value.

`struct EmptyModifier`

An empty, or identity, modifier, used during development to switch modifiers at compile time.

`struct ModifiedContent`

A value with a modifier applied to it.

`protocol EnvironmentalModifier`

A modifier that must resolve to a concrete modifier in an environment before use.

`struct ManipulableModifier`

`struct ManipulableResponderModifier`

`struct ManipulableTransformBindingModifier`

`struct ManipulationGeometryModifier`

`struct ManipulationGestureModifier`

`struct ManipulationUsingGestureStateModifier`

`enum Manipulable`

A namespace for various manipulable related types.

## Responding to view life cycle updates

`func onAppear(perform: (() -> Void)?) -> some View`

Adds an action to perform before this view appears.

`func onDisappear(perform: (() -> Void)?) -> some View`

Adds an action to perform after this view disappears.

`func task(priority: TaskPriority, () async -> Void) -> some View`

Adds an asynchronous task to perform before this view appears.

`func task<T>(id: T, priority: TaskPriority, () async -> Void) -> some View`

Adds a task to perform before this view appears or when a specified value changes.

## Managing the view hierarchy

`func id<ID>(ID) -> some View`

Binds a view's identity to the given proxy value.

`func tag<V>(V, includeOptional: Bool) -> some View`

Sets the unique tag value of this view.

`func equatable() -> EquatableView<Self>`

Prevents the view from updating its child view when its new value is the same as its old value.

## Supporting view types

`struct AnyView`

A type-erased view.

`struct EmptyView`

A view that doesn't contain any content.

`struct` `EquatableView`

A view type that compares itself against its previous value and prevents its child updating if its new value is the same as its old value.

`struct` `SubscriptionView`

A view that subscribes to a publisher with an action.

`struct` `TupleView`

A View created from a swift tuple of View values.

---

# See Also

## 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.

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.