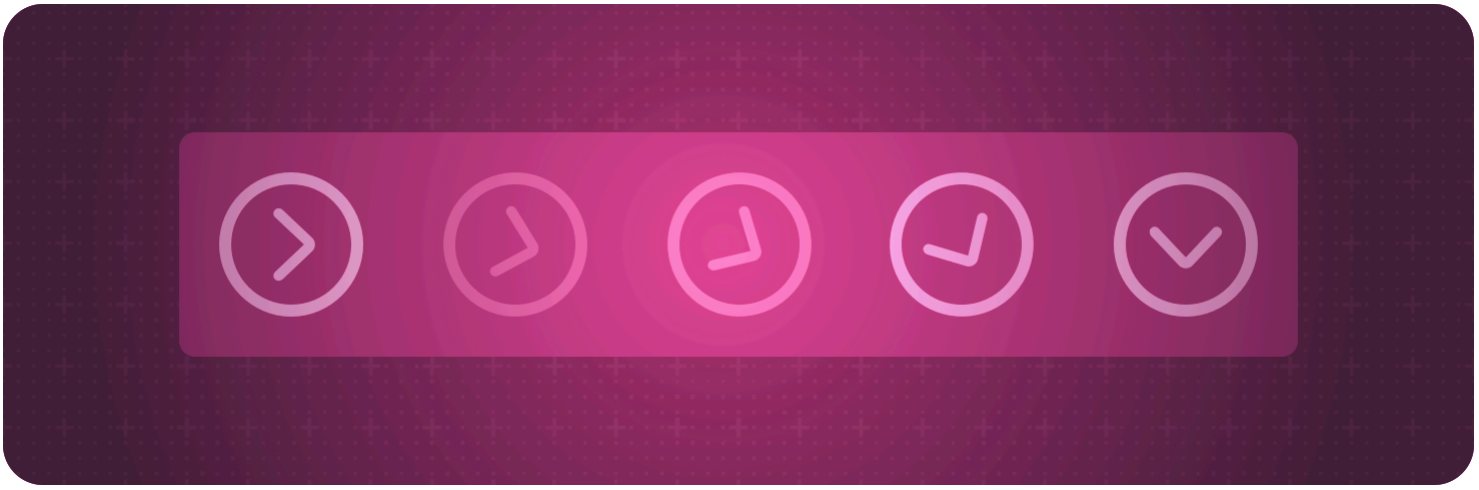
API Collection

Animations

Create smooth visual updates in response to state changes.

Overview

You tell SwiftUI how to draw your app's user interface for different states, and then rely on SwiftUI to make interface updates when the state changes.



To avoid abrupt visual transitions when the state changes, add animation in one of the following ways:

- Animate all of the visual changes for a state change by changing the state inside a call to the `withAnimation(_:_:)` global function.
- Add animation to a particular view when a specific value changes by applying the `animation(_:value:)` view modifier to the view.
- Animate changes to a `Binding` by using the binding's `animation(_:)` method.

SwiftUI animates the effects that many built-in view modifiers produce, like those that set a scale or opacity value. You can animate other values by making your custom views conform to the `Animatable` protocol, and telling SwiftUI about the value you want to animate.

When an animated state change results in adding or removing a view to or from the view hierarchy, you can tell SwiftUI how to transition the view into or out of place using built-in transitions that `AnyTransition` defines, like `slide` or `scale`. You can also create custom transitions.

For design guidance, see Motion in the Human Interface Guidelines.

Topics

Adding state-based animation to an action

`func withAnimation<Result>(Animation?, () throws -> Result) rethrows -> Result`

Returns the result of recomputing the view's body with the provided animation.

`func withAnimation<Result>(Animation?, completionCriteria: AnimationCompletionCriteria, () throws -> Result, completion: () -> Void) rethrows -> Result`

Returns the result of recomputing the view's body with the provided animation, and runs the completion when all animations are complete.

`struct AnimationCompletionCriteria`

The criteria that determines when an animation is considered finished.

`struct Animation`

The way a view changes over time to create a smooth visual transition from one state to another.

Adding state-based animation to a view

`func animation(_:)`

Applies the given animation to this view when this view changes.

`func animation<V>(Animation?, value: V) -> some View`

Applies the given animation to this view when the specified value changes.

`func animation<V>(Animation?, body: (PlaceholderContentView<Self>) -> V) -> some View`

Applies the given animation to all animatable values within the `body` closure.

Creating phase-based animation

{} Controlling the timing and movements of your animations

Build sophisticated animations that you control using phase and keyframe animators.

```
func phaseAnimator<Phase>(some Sequence, content: (PlaceholderContent
View<Self>, Phase) -> some View, animation: (Phase) -> Animation?) ->
some View
```

Animates effects that you apply to a view over a sequence of phases that change continuously.

```
func phaseAnimator<Phase>(some Sequence, trigger: some Equatable,
content: (PlaceholderContentView<Self>, Phase) -> some View, animation:
(Phase) -> Animation?) -> some View
```

Animates effects that you apply to a view over a sequence of phases that change based on a trigger.

```
struct PhaseAnimator
```

A container that animates its content by automatically cycling through a collection of phases that you provide, each defining a discrete step within an animation.

Creating keyframe-based animation

```
func keyframeAnimator<Value>(initialValue: Value, repeating: Bool,
content: (PlaceholderContentView<Self>, Value) -> some View, keyframes:
(Value) -> some Keyframes) -> some View
```

Loops the given keyframes continuously, updating the view using the modifiers you apply in `body`.

```
func keyframeAnimator<Value>(initialValue: Value, trigger: some
Equatable, content: (PlaceholderContentView<Self>, Value) -> some View,
keyframes: (Value) -> some Keyframes) -> some View
```

Plays the given keyframes when the given trigger value changes, updating the view using the modifiers you apply in `body`.

```
struct KeyframeAnimator
```

A container that animates its content with keyframes.

```
protocol Keyframes
```

A type that defines changes to a value over time.

```
struct KeyframeTimeline
```

A description of how a value changes over time, modeled using keyframes.

`struct` KeyframeTrack

A sequence of keyframes animating a single property of a root type.

`struct` KeyframeTrackContentBuilder

The builder that creates keyframe track content from the keyframes that you define within a closure.

`struct` KeyframesBuilder

A builder that combines keyframe content values into a single value.

`protocol` KeyframeTrackContent

A group of keyframes that define an interpolation curve of an animatable value.

`struct` CubicKeyframe

A keyframe that uses a cubic curve to smoothly interpolate between values.

`struct` LinearKeyframe

A keyframe that uses simple linear interpolation.

`struct` MoveKeyframe

A keyframe that immediately moves to the given value without interpolating.

`struct` SpringKeyframe

A keyframe that uses a spring function to interpolate to the given value.

Creating custom animations

`protocol` CustomAnimation

A type that defines how an animatable value changes over time.

`struct` AnimationContext

Contextual values that a custom animation can use to manage state and access a view's environment.

`struct` AnimationState

A container that stores the state for a custom animation.

`protocol` AnimationStateKey

A key for accessing animation state values.

`struct` UnitCurve

A function defined by a two-dimensional curve that maps an input progress in the range [0,1] to an output progress that is also in the range [0,1]. By changing the shape of the curve, the effective speed of an animation or other interpolation can be changed.

`struct Spring`

A representation of a spring's motion.

Making data animatable

`protocol Animatable`

A type that describes how to animate a property of a view.

`struct AnimatableValues`

`struct AnimatablePair`

A pair of animatable values, which is itself animatable.

`protocol VectorArithmetic`

A type that can serve as the animatable data of an animatable type.

`struct EmptyAnimatableData`

An empty type for animatable data.

Updating a view on a schedule

Updating watchOS apps with timelines

Seamlessly schedule updates to your user interface, even while it's inactive.

`struct TimelineView`

A view that updates according to a schedule that you provide.

`protocol TimelineSchedule`

A type that provides a sequence of dates for use as a schedule.

`typealias TimelineViewDefaultContext`

Information passed to a timeline view's content callback.

Synchronizing geometries

`func matchedGeometryEffect<ID>(id: ID, in: Namespace.ID, properties: MatchedGeometryProperties, anchor: UnitPoint, isSource: Bool) -> some View`

Defines a group of views with synchronized geometry using an identifier and namespace that you provide.

`struct MatchedGeometryProperties`

A set of view properties that may be synchronized between views using the `View.matchedGeometryEffect()` function.

`protocol GeometryEffect`

An effect that changes the visual appearance of a view, largely without changing its ancestors or descendants.

`struct Namespace`

A dynamic property type that allows access to a namespace defined by the persistent identity of the object containing the property (e.g. a view).

`func geometryGroup() -> some View`

Isolates the geometry (e.g. position and size) of the view from its parent view.

Defining transitions

`func transition(_:)`

Associates a transition with the view.

`protocol Transition`

A description of view changes to apply when a view is added to and removed from the view hierarchy.

`struct TransitionProperties`

The properties a `Transition` can have.

`enum TransitionPhase`

An indication of which the current stage of a transition.

`struct AsymmetricTransition`

A composite `Transition` that uses a different transition for insertion versus removal.

`struct AnyTransition`

A type-erased transition.

`func contentTransition(ContentTransition) -> some View`

Modifies the view to use a given transition as its method of animating changes to the contents of its views.

`var contentTransition: ContentTransition`

The current method of animating the contents of views.

`var contentTransitionAddsDrawingGroup: Bool`

A Boolean value that controls whether views that render content transitions use GPU-accelerated rendering.

`struct ContentTransition`

A kind of transition that applies to the content within a single view, rather than to the insertion or removal of a view.

`struct PlaceholderContentView`

A placeholder used to construct an inline modifier, transition, or other helper type.

`func navigationTransition(some NavigationTransition) -> some View`

Sets the navigation transition style for this view.

`protocol NavigationTransition`

A type that defines the transition to use when navigating to a view.

`func matchedTransitionSource(id: some Hashable, in: Namespace.ID) -> some View`

Identifies this view as the source of a navigation transition, such as a zoom transition.

`func matchedTransitionSource(id: some Hashable, in: Namespace.ID, configuration: (EmptyMatchedTransitionSourceConfiguration) -> some MatchedTransitionSourceConfiguration) -> some View`

Identifies this view as the source of a navigation transition, such as a zoom transition.

`protocol MatchedTransitionSourceConfiguration`

A configuration that defines the appearance of a matched transition source.

`struct EmptyMatchedTransitionSourceConfiguration`

An unstyled matched transition source configuration.

Moving an animation to another view

`func withTransaction<Result>(Transaction, () throws -> Result) rethrows -> Result`

Executes a closure with the specified transaction and returns the result.

`func withTransaction<R, V>(WritableKeyPath<Transaction, V>, V, () throws -> R) rethrows -> R`

Executes a closure with the specified transaction key path and value and returns the result.

`func transaction((inout Transaction) -> Void) -> some View`

Applies the given transaction mutation function to all animations used within the view.

`func transaction(value: some Equatable, (inout Transaction) -> Void) -> some View`

Applies the given transaction mutation function to all animations used within the view.

`func transaction<V>((inout Transaction) -> Void, body: (PlaceholderContentView<Self>) -> V) -> some View`

Applies the given transaction mutation function to all animations used within the `body` closure.

`struct Transaction`

The context of the current state-processing update.

`macro Entry()`

Creates an environment values, transaction, container values, or focused values entry.

`protocol TransactionKey`

A key for accessing values in a transaction.

Deprecated types

~~`protocol AnimatableModifier`~~

A modifier that can create another modifier with animation.

Deprecated

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.

- 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.