

[Updates](#) / SwiftUI updates

Article

SwiftUI updates

Learn about important changes to SwiftUI.



Overview

Browse notable changes in [SwiftUI](#).

June 2025

General

- Apply Liquid Glass effects to views using `doc://com.apple.documentation/documentation/swiftui/view/glasseffect(_in:isEnabled:)`.
- Use `glass` with the `buttonStyle(_:)` modifier to apply Liquid Glass to instances of `Button`.
- `ToolbarSpacer` creates a visual break between items in toolbars containing Liquid Glass.
- Use `scrollEdgeEffectStyle(_:for:)` to configure the scroll edge effect style for scroll views.
- `backgroundExtensionEffect()` duplicates, mirrors, and blurs views placed around edges with available safe areas.
- Set behavior for tab bar minimization with `tabBarMinimizeBehavior(_:)`.
- Set the `search` role on a tab to take someone to a search tab and have a search field take the place of the tab bar.
- Adjust the content of accessory views based on the placement in a tab view with `TabViewBottomAccessoryPlacement`.
- Connect a `WebView` with a `WebPage` to fully control the browsing experience in your app.

- Drag multiple items using the `doc://com.apple.documentation/documentation/swiftui/view/draggable(_:_:)` modifier. Make a view a container for draggable views using the `doc://com.apple.documentation/documentation/swiftui/view/dragcontainer(for:id:in:selection:_:)` modifier.
- Use the `Animatable()` macro to have SwiftUI synthesize custom animatable data properties.
- `Slider` now supports tick marks. Tick marks appear automatically when initializing a `Slider` with the `step` parameter.
- Use `windowResizeAnchor(_:)` to set the window anchor point when a window must resize.

Text

- `TextEditor` now supports `AttributedString`.
- Handle text selection with attributed text using `AttributedTextSelection`.
- `AttributedTextFormattingDefinition` defines how text can be styled in specific contexts.
- Use `FindContext` to create a find navigator in views that support text editing.

Accessibility

- Support Assistive Access in iOS and iPadOS scenes with `AssistiveAccess`.

HDR

- `Color.ResolvedHDR` is a set of RGBA values that represent a color that can be shown, including HDR headroom information.

UIKit and AppKit integration

- Host and present SwiftUI scenes in UIKit with `UIHostingSceneDelegate` and in AppKit with `NSHostingSceneRepresentation`.
- Incorporate gesture recognizers in SwiftUI views from AppKit with `NSGestureRecognizerRepresentable`.

Immersive spaces

- Manipulate views using common hand gestures with `manipulable(coordinateSpace:operations:inertia:isEnabled:onChangeed:)`.
- Snap volumes to horizontal surfaces and windows to vertical surfaces using `SurfaceSnappingInfo`.
- Use `RemoteImmersiveSpace` to render stereo content from your Mac app on Apple Vision Pro.
- Use `SpatialContainer` to create a layout container that aligns overlapping content in 3D space.
- Depth-based variants of modifiers allow easier volumetric layouts in SwiftUI. For example, `aspectRatio3D(_:contentMode:)`, `rotation3DLayout(_:)`, and `depthAlignment(_:)`.

June 2024

Volumes

- Specify the alignment of a volume when moved in the world using the `volumeWorldAlignment(_:)` scene modifier.
- Specify the default world scaling behavior of your scene using the `defaultWorldScaling(_:)` scene modifier.
- Adjust the visibility of a volume's baseplate using the `volumeBaseplateVisibility(_:)` view modifier.
- Define a custom action to execute when the viewpoint of a volume changes using the `onVolumeViewpointChange(updateStrategy:initial: :)` view modifier.

Windows

- Change the default initial size and position of a window using the `defaultWindowPlacement(_:)` modifier.
- Change the default behavior for how windows behave when performing a zoom using `WindowIdealSize` and provide the placement for the zoomed window with the `windowIdealPlacement(_:)` modifier.
- Create utility windows in SwiftUI using the new `UtilityWindow` scene type and toggle the window's visibility using the `WindowVisibilityToggle`.
- Customize the style of a window using the new `window` container background placement, the `toolbar(removing:)` view modifier, and the `plain` window style.

- Set the default launch behavior for a scene using the defaultLaunchBehavior(_:) modifier.
- Replace one scene with another using the pushWindow method.

Immersive spaces

- Add an action to perform when the state of the immersion changes using the onImmersionChange(_:) modifier.
- Apply a custom color or dim a passthrough video in an immersive space using the colorMultiply(_:) and dim(intensity:) initializers.

Documents

- Customize the launch experience of document-based applications using DocumentGroupLaunchScene and NewDocumentButton.

Navigation

- Specify the appearance and interaction of TabView with the tabViewStyle(_:) modifier using values like sidebarAdaptable, tabBarOnly, and grouped.
- Build hierarchy by nesting tabs as a tab item within TabSection.
- Enable people to customize a TabView using the tabViewCustomization(_:) modifier and persist customization state in AppStorage with TabViewCustomization.

Modal presentations

- Use built-in presentation sizes for sheets like form or page with the presentationSizing(_:) modifier or create custom sized sheets using the PresentationSizing protocol.

Toolbars

- Specify the display mode of toolbars in macOS using the ToolbarLabelStyle type.
- Configure the foreground style in the toolbar environment in watchOS using the toolbarForegroundStyle(_:for:) view modifier.
- Anchor ornaments relative to the depth of your volume — in addition to the height and width — using the scene(_:) method that takes a UnitPoint3D.

Views

- Create custom container views like Picker, List, and TabView using new Group and ForEach initializers, like init(subviews:transform:) and init(subviews:content:), respectively.
- Declare a custom container value by defining a key that conforms to the ContainerValueKey protocol, and set the container value for a view using the containerValue(: :) modifier.
- Create EnvironmentValues, Transaction, ContainerValues, and FocusedValues entries by using the Entry() macro to the variable declaration.

Animation

- Customize the transition when pushing a view onto a navigation stack or presenting a view with the navigationTransition(:) view modifier.
- Add new symbols effects and configurations like wiggle, rotate, and breathe using the symbolEffect(_:options:value:) modifier.

Text input and output

- Add text suggestions support to any text field using textInputSuggestions(:) and textInputCompletion(:) view modifiers.
- Access and modify selected text using a new TextSelection binding for TextField and TextEditor.
- Bind to the focus state of an app's search field using the searchFocused(_:equals:) view modifier.

Drawing and graphics

- Precompile shaders at build time using the compile(as:) method.
- Create mesh gradients with a grid of points and colors using the new MeshGradient type.
- Extend SwiftUI Text views with custom rendering effects and interaction behaviors using TextAttribute, Text.Layout, and TextRenderer.
- Create a new Color by mixing two colors using the mix(with:by:in:) method.

Layout

- Enable custom spacing between views in a ZStack along the depth axis with the init(alignment:spacing:content:) initializer.

Scrolling

- Scroll to a view, offset, or edge in a scroll view using the scrollPosition(_:anchor:) view modifier and specifying one of the ScrollPosition values.
- Limit the number of views that can be scrolled by a single interaction using the limit behavior value alwaysByFew or alwaysByOne.
- Add an action to be called when a view crosses a provided threshold using the onScrollVisibilityChange(threshold:_:) modifier.
- Access both the old and new values when a scroll view's phase changes by using the onScrollPhaseChange(_:) modifier.

Gestures

- Conditionally disable a gesture using the `isEnabled` parameter in a modifier like gesture(_:isEnabled:).
- Create extra drag areas of a window in macOS when you add a WindowDragGesture gesture.
- Create a hand gesture shortcut for Double Tap in watchOS using the HandGestureShortcut structure.
- Enable whether gestures can handle events that activate the containing window using the allowsWindowActivationEvents(_:) view modifier.

Input events

- Create a group of hover effects that activate together using HoverEffectGroup and apply them to a view using the hoverEffect(in:isEnabled:body:) view modifier.
- Customize the appearance of the system pointer in macOS, iPadOS, and visionOS with new pointer styles using pointerStyle(_:) or the visibility with the pointerVisibility(_:) modifier.
- Access keyboard modifier flags using the onModifierKeysChanged(mask:initial:_:).
- Replace the primary view with one or more alternative views when pressing a specified set of modifier keys using the modifierKeyAlternate(_:_:) view modifier.
- Enable the hand pointer for custom drawing and markup applications using the handPointerBehavior(_:) modifier.

Previews in Xcode

- Write dynamic properties inline in previews using the new `Previewable()` macro.
- Inject shared environment objects, model containers, or other dependencies into previews using the `PreviewModifier` protocol.

Accessibility

- Specify that your accessibility element behaves as a tab bar using the `isTabBar` accessibility trait with the `accessibilityAddTraits(_:)` modifier. In UIKit, use `tabBar`.
- Generate a localized description of a color in a string interpolation by adding `accessibilityName:`, such as `"\((accessibilityName: myColor)\"`. Pass that string to any accessibility modifier.

Framework interoperability

- Reuse existing UIKit gesture recognizer code in SwiftUI. In SwiftUI, create UIKit gesture recognizers using `UIGestureRecognizerRepresentable`. In UIKit, refer to SwiftUI gestures by name using `name`.
 - Share menu content definitions between SwiftUI and AppKit by using the `NSHostingMenu` in your AppKit view hierarchy.
-

June 2023, visionOS

Scenes

- Create a volume that can display 3D models by applying the `volumetric` window style to an app's window.
- Make use of a Full Space by opening an `ImmersiveSpace` scene. You can use the `mixed` immersion style to place objects in a person's surroundings, or the `full` style to completely control the visual experience.
- Display 3D models in a volume or a Full Space using RealityKit entities that you load with that framework's `Model3D` or `RealityView` structure.

Toolbars and ornaments

- Display a toolbar item in an ornament using the bottomOrnament toolbar item placement.
- Add an ornament to a window directly using the ornament(visibility:attachmentAnchor:contentAlignment:ornament:) view modifier.

Drawing and graphics

- Detect view geometry in three dimensions using a GeometryReader3D.
- Add a 3D visual effect using the visualEffect3D(_:) view modifier.
- Rotate or scale in three dimensions with view modifiers like rotation3DEffect(_:anchor:) and scaleEffect(x:y:z:anchor:), respectively.
- Convert between display points and physical distances using a PhysicalMetricsConverter.

View configuration

- Add a glass background effect to a view using the glassBackgroundEffect(displayMode:) view modifier.
- Dim passthrough when appropriate by applying a preferredSurroundingsEffect(:) modifier.

View layout

- Make 3D adjustments to layout with view modifiers like offset(z:), padding3D(_:), and frame(depth:alignment:).

Gestures

- Enable people to rotate objects in three dimensions when you add a RotateGesture3D gesture.
-

June 2023

Scenes

- Close windows by their identifier using the dismissWindow action stored in the environment.

- Enable people to open a settings window by presenting a SettingsLink button.

Navigation

- Control views of a navigation split view or stack using a new overload of the navigationDestination(item:destination:) view modifier.
- Manage column visibility of a navigation split view using new overloads of the view's initializer, like init(columnVisibility:preferredCompactColumn:sidebar:content:detail:).

Modal presentations

- Use new overloads of the file export, import, and move modifiers, like fileExporter(isPresented:document:contentTypes:defaultFilename:onCompletion:onCancellation:), to access new file management features. For example, you can:
 - Configure a file import or export dialog to open on a default directory, enable only certain file types, display hidden files, and so on.
 - Retain file interface configuration that a person chooses from one presentation to the next.
 - Export types that conform to the Transferable protocol.
- Specify a dialog severity using the dialogSeverity(_:) view modifier.
- Provide a custom icon for a dialog using the dialogIcon(_:) modifier.
- Enable people to suppress dialogs using one of the dialog suppression modifiers, like dialogSuppressionToggle(isSuppressed:).

Toolbars

- Configure the toolbar title display size using the toolbarTitleDisplayMode(_:) modifier.

Search

- Present search programmatically using a binding to a new `isPresented` parameter available in some searchable view modifiers, like searchable(text:isPresented:placement:prompt:).
- Create mutable search tokens by providing a binding to the input of the token closure in the applicable searchable view modifiers, like searchable(text:editableTokens:isPresented:placement:prompt:token:).

Data and storage

- Bridge between SwiftUI environment keys and UIKit traits more easily using the UITraitBridgedEnvironmentKey protocol.
- Get better performance when you share data throughout your app by using the new Observable() macro.
- Access both the old and new values of a value that changes when processing the completion closure of the onChange(of:initial:_:) view modifier.

Views

- Display a standard interface when a resource, like search results or a network connection, isn't available using the ContentUnavailableView view type.
- Display a standard inspector interface with a platform-appropriate appearance by applying the inspector(isPresented:content:) modifier.

Animation

- Perform an action when an animation completes by specifying a completion closure to the withAnimation(_:completionCriteria:_:completion:) view modifier.
- Define custom animation behaviors by creating a type that conforms to the CustomAnimation protocol.
- Perform animations that progress through predefined phases using the PhaseAnimator structure, or according to a set of time-based keyframes by using the Keyframes protocol.
- Specify information about a change in state — for example, to request a particular animation — using custom TransactionKey instances.
- Design custom animation curves using UnitCurve.
- Apply streamlined spring parameters, now standardized across all Apple frameworks, using the new spring(duration:bounce:blendDuration:) animation. You can also use the Spring structure as a convenience to represent a spring's motion.

Text input and output

- Indicate the language that appears in a specific Text view so that SwiftUI can help to avoid clipping and collision of text, and perform proper line breaking and hyphenation using the typesettingLanguage(_:isEnabled:) view modifier.

- Scale text semantically, for example by labeling it as having a secondary text scale, using the `textScale(_ :isEnabled:)` modifier.

Shapes

- Apply more than one `fill(_ :style:)` or `stroke(_ :style:antialiased:)` modifier to a single `Shape`.
- Apply Boolean operations to both shapes and paths, like `intersection(_ :eoFill:)` and `union(_ :eoFill:)`.
- Use predefined shape styles, like `rect`, to simplify your code.
- Create rounded rectangles with uneven corners using `rect(topLeadingRadius:bottomLeadingRadius:bottomTrailingRadius:topTrailingRadius:style:)`.

Drawing and graphics

- Create fully customizable, high-performance graphics by drawing with Metal shaders inside a SwiftUI app using a `Shader` structure.
- Configure an image with a specific dynamic range by applying the `allowedDynamicRange(_ :)` view modifier.
- Compose effects that you apply to a view based on some aspect of the geometry of the view using the `visualEffect(_ :)` modifier. For example, you can apply a blur that varies depending on the view's position in the display.

Layout

- Define custom coordinate spaces using the `CoordinateSpaceProtocol` with new `GeometryProxy` methods, like `bounds(of:)` and `frame(in:)`, to get the dimensions of containers.
- Create a frame for a view that lays out its content based on characteristics of the container view using `containerRelativeFrame(_ :alignment:)`.
- Set the background of a container view using the `containerBackground(_ :for:)` view modifier.

Lists and tables

- Disable selectability of an item in a `List` or `Table` by applying the `selectionDisabled(_ :)` modifier.

- Collapse or expand a Section of a list or table using the `isExpanded` binding in the section's initializer.
- Configure row or section spacing using the `listRowSpacing(_:)` and `listSectionSpacing(_:)` modifiers, respectively.
- Set the prominence of a badge using the `badgeProminence(_:)` view modifier.
- Configure alternating row backgrounds using the `alternatingRowBackgrounds(_:)` modifier.
- Customize table column visibility and reordering using the `TableColumnCustomization` structure.
- Add hierarchical rows to a table using the `DisclosureTableRow` structure, or recursively hierarchical rows using the `OutlineGroup` structure.
- Hide table column headers using the `tableColumnHeaders(_:)` modifier.

Scrolling

- Read the position of a scroll view using one of the scroll position modifiers, like `scrollPosition(id:anchor:)`.
- Flash scroll indicators programmatically using a view modifier, like `scrollIndicatorsFlash(onAppear:)`.
- Clip scroll views in custom ways after disabling default clipping using the `scrollClipDisabled(_:)` modifier.
- Create paged scroll views, aligned to either page or view boundaries, using the `scrollTargetBehavior(_:)` view modifier.
- Create custom scroll behaviors using the `ScrollTargetBehavior` protocol.
- Control the insets of scrollable views using the `safeAreaPadding(_:)` and `contentMargins(_:_:for:)` view modifiers.
- Add effects to views as they scroll on- and offscreen using one of the `scrollTransition(_:axis:transition:)` modifiers.
- Create a `TabView` that supports vertical paging in watchOS by applying the `verticalPage` tab view style.

Gestures

- Make smoother transitions between gestures and animations by using a new `velocity` property on the values associated with certain gestures and a `tracksVelocity` property on

Transaction.

- Gain access to more information, including both velocity and position, by migrating to the new MagnifyGesture and RotateGesture, which replace the now deprecated MagnificationGesture and RotationGesture.

Input events

- Enable a view that's in focus to react directly to keyboard input by applying one of the onKeyPress(_:action:) view modifiers.
- Enable people to choose from a compact collection of items in a Menu by styling a Picker with the palette style.
- Provide haptic or audio feedback in response to an event using one of the sensory feedback modifiers, like sensoryFeedback(_:trigger:).
- Create buttons and toggles that perform an AppIntent in a widget, Live Activity, and other places using new initializers like init(_:intent:) and init(_:isOn:intent:).

Focus


- Distinguish between views for which focus serves different purposes, such as those that have a primary action like a button and those that take input like a text field, using the new focusable(_:interactions:) view modifier.
- Manage the effect that receiving focus has on a view using the focusEffectDisabled(_:) modifier.















Previews in Xcode

- Reduce the amount of boilerplate that you need to create Xcode previews by using the new Preview(_:traits:_:body:) macro.
-

See Also

Technology updates

-  Accelerate updates
Learn about important changes to Accelerate.

-  Accessibility updates
Learn about important changes to Accessibility.
-  ActivityKit updates
Learn about important changes in ActivityKit.
-  AdAttributionKit Updates
Learn about important changes to AdAttributionKit.
-  App Clips updates
Learn about important changes in App Clips.
-  App Intents updates
Learn about important changes in App Intents.
-  AppKit updates
Learn about important changes to AppKit.
-  Apple Intelligence updates
Learn about important changes to Apple Intelligence.
-  AppleMapsServerAPI Updates
Learn about important changes to AppleMapsServerAPI.
-  Apple Pencil updates
Learn about important changes to Apple Pencil.
-  ARKit updates
Learn about important changes to ARKit.
-  Audio Toolbox updates
Learn about important changes to Audio Toolbox.
-  AuthenticationServices updates
Learn about important changes to AuthenticationServices.
-  AVFAudio updates
Learn about important changes to AVFAudio.
-  AVFoundation updates
Learn about important changes to AVFoundation.

