Documentation

Language: Swift

Adopting Liquid Glass

Find out how to bring the new material to your app.

Overview

If you have an existing app, adopting Liquid Glass doesn't mean reinventing your app from the ground up. Start by building your app in the latest version of Xcode to see the changes. As you review your app, use the following sections to understand the scope of changes and learn how you can adopt these best practices in your interface.

See your app with Liquid Glass

If your app uses standard components from SwiftUI, UIKit, or AppKit, your interface picks up the latest look and feel on the latest platform releases for iOS, iPadOS, macOS, tvOS, and watchOS. In Xcode, build your app with the latest SDKs, and run it on the latest platform releases to see the changes in your interface.

Visual refresh

Interfaces across Apple platforms feature a new dynamic material called Liquid Glass, which combines the optical properties of glass with a sense of fluidity. This material forms a distinct functional layer for controls and navigation elements. It affects how the interface looks, feels, and moves, adapting in response to a variety of factors to help bring focus to the underlying content.

Leverage system frameworks to adopt Liquid Glass automatically. In system frameworks, standard components like bars, sheets, popovers, and controls automatically adopt this material. System frameworks also dynamically adapt these components in response to factors like element overlap and focus state. Take advantage of this material with minimal code by using standard components from SwiftUI, UIKit, and AppKit.

Reduce your use of custom backgrounds in controls and navigation elements. Any custom backgrounds and appearances you use in these elements might overlay or interfere with Liquid Glass or other effects that the system provides, such as the scroll edge effect. Make sure to check any custom backgrounds in elements like split views, tab bars, and toolbars. Prefer to remove custom effects and let the system determine the background appearance, especially for the following elements:

SwiftUI UIKit AppKit

`NavigationStack`

`NavigationSplitView`

`titleBar`

`toolbar(content:)`

Test your interface with accessibility settings. Translucency and fluid morphing

animations contribute to the look and feel of Liquid Glass, but can adapt to people's needs. For example, people might turn on accessibility settings that reduce transparency or motion in the interface, which can remove or modify certain effects. If you use standard components from system frameworks, this experience adapts automatically. Ensure your custom elements and animations provide a good fallback experience when these settings are on as well.

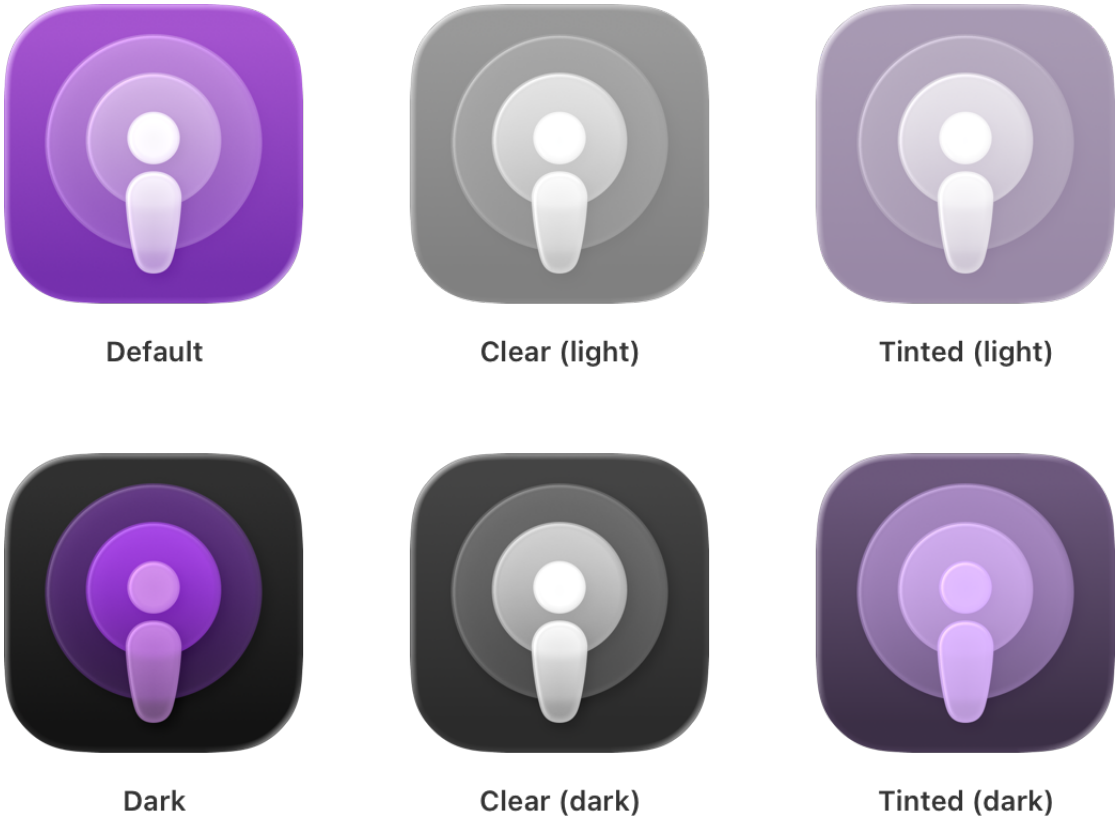
Avoid overusing Liquid Glass effects. If you apply Liquid Glass effects to a custom control, do so sparingly. Liquid Glass seeks to bring attention to the underlying content, and overusing this material in multiple custom controls can provide a subpar user experience by distracting from that content. Limit these effects to the most important functional elements in your app. To learn more, read Applying Liquid Glass to custom views.

SwiftUI UIKit AppKit

`glassEffect(_:in:)`

App icons

App icons take on a design that's dynamic and expressive. Updates to the icon grid result in a standardized iconography that's visually consistent across devices and concentric with hardware and other elements across the system. App icons now contain layers, which dynamically respond to lighting and other visual effects the system provides. iOS, iPadOS, and macOS all now offer default (light), dark, clear, and tinted appearance variants, empowering people to personalize the look and feel of their Home Screen.



Reimagine your app icon for Liquid Glass. Apply key design principles to help your app icon shine:

- Provide a visually consistent, optically balanced design across the platforms your app supports.
- Consider a simplified design comprised of solid, filled, overlapping semi-transparent shapes.
- Let the system handle applying masking, blurring, and other visual effects, rather than factoring them into your design.

Podcasts icon in iOS 18 | Reimagined layered Podcasts icon | Reimagined layered Podcasts icon with system effects applied

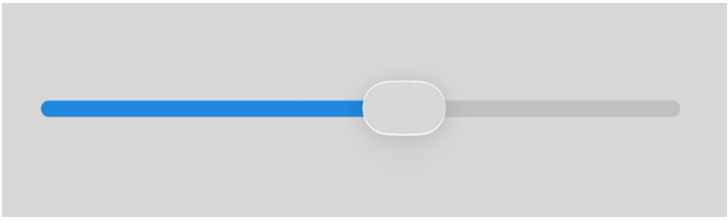
Design using layers. The system automatically applies effects like reflection, refraction, shadow, blur, and highlights to your icon layers. Determine which elements of your design make sense as foreground, middle, and background elements, then define separate layers for them. You can perform this task in the design app of your choice.

Compose and preview in Icon Composer. Drag and drop app icon layers that you export from your design app directly into the Icon Composer app. Icon Composer lets you add a background, create layer groupings, adjust layer attributes like opacity, and preview your design with system effects and appearances. Icon Composer is available in the latest version of Xcode and for download from Apple Design Resources. To learn more, read Creating your app icon using Icon Composer.

Preview against the updated grids. The system applies masking to produce your final icon shape — rounded rectangle for iOS, iPadOS, and macOS, and circular for watchOS. Keep elements centered to avoid clipping. Irregularly shaped icons receive a system-provided background. See how your app icon looks with the updated grids to determine whether you need to make adjustments. Download these grids from Apple Design Resources.

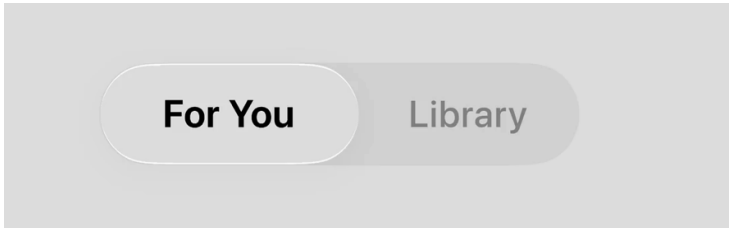
Controls

Controls have a refreshed look across platforms, and come to life when a person interacts with them. For controls like sliders and toggles, the knob transforms into Liquid Glass during interaction, and buttons fluidly morph into menus and popovers. The shape of the hardware informs the curvature of controls, so many controls adopt rounder forms to elegantly nestle into the corners of windows and displays. Controls also feature an option for an extra-large size, allowing more space for labels and accents.



Play

Slider



Play

Segmented control

Review updates to control appearance and dimensions. If you use standard controls from system frameworks and don't hard-code their layout metrics, your app adopts changes to shapes and sizes automatically when you rebuild your app with the latest version of Xcode. Review changes to the following controls and any others and make sure they continue to look at home with the rest of your interface:

SwiftUI UIKit AppKit

`Button`

`Toggle`

`Slider`

`Stepper`

`Picker`

`TextField`

Review your use of color in controls. Be judicious with your use of color in controls and navigation so they stay legible. If you do apply color to these elements, leverage system colors to automatically adapt to light and dark contexts.

Check for crowding or overlapping of controls. Prefer to use standard spacing metrics instead of overriding them, and avoid overcrowding or layering Liquid Glass elements on top of each other.

Optimize for legibility when content scrolls beneath controls. Scroll views offer a scroll edge effect that helps maintain sufficient legibility and contrast for controls by obscuring content that scrolls beneath them. System bars like toolbars adopt this behavior by default. If you use a custom bar with elements like controls, text, or icons that have content scrolling beneath them, you can register those views to use a scroll edge effect with these APIs:

SwiftUI UIKit

`safeAreaBar(edge:alignment:spacing:content:)`

Consider aligning the shape of controls with other rounded elements throughout the interface. Across Apple platforms, the shape of the hardware informs the curvature, size, and shape of nested interface elements, including controls, sheets, popovers, windows, and more. Help maintain a sense of visual continuity in your interface by using rounded shapes that are concentric to their containers using these APIs:

SwiftUI UIKit

`rect(corners:isUniform:)`

`ConcentricRectangle`

Leverage new button styles. Instead of creating buttons with custom Liquid Glass effects, you can adopt the look and feel of the material with minimal code by using one of the following button style APIs:

SwiftUI UIKit AppKit

`glass`

`glassProminent`

`glass(_:)`

Navigation

Liquid Glass applies to the topmost layer of the interface, where you define your navigation. Key navigation elements like tab bars and sidebars float in this Liquid Glass layer to help people focus on the underlying content.

Establish a clear navigation hierarchy. It's more important than ever for your app to have a clear and consistent navigation structure that's distinct from the content you provide. Ensure that you clearly separate your content from navigation elements, like tab bars and sidebars, to establish a distinct functional layer above the content layer.

Consider adapting your tab bar into a sidebar automatically. If your app uses a tab-based navigation, you can allow the tab bar to adapt into a sidebar depending on the context by using the following APIs:

SwiftUI UIKit

`sidebarAdaptable`

Consider using split views to build sidebar layouts with an inspector panel. Split views are optimized to create a consistent and familiar experience for sidebar and inspector layouts across platforms. You can use the following standard system APIs for split views to build these types of layouts with minimal code:

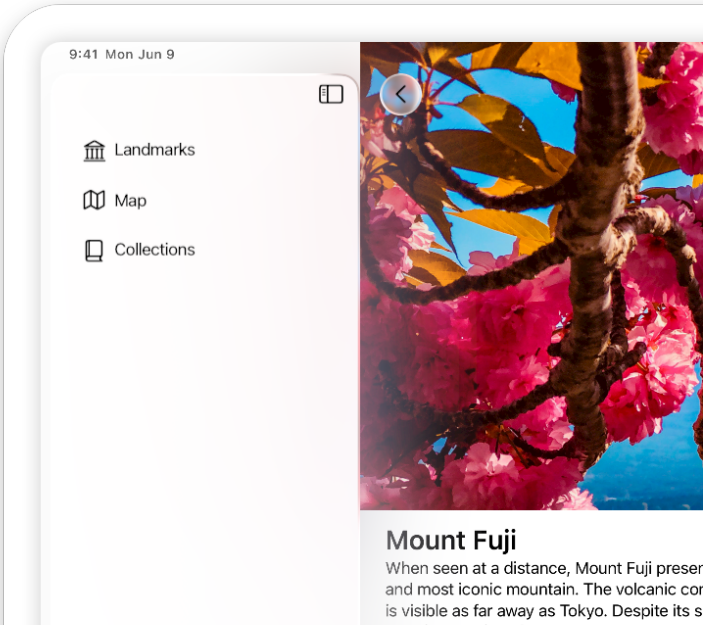
SwiftUI UIKit AppKit

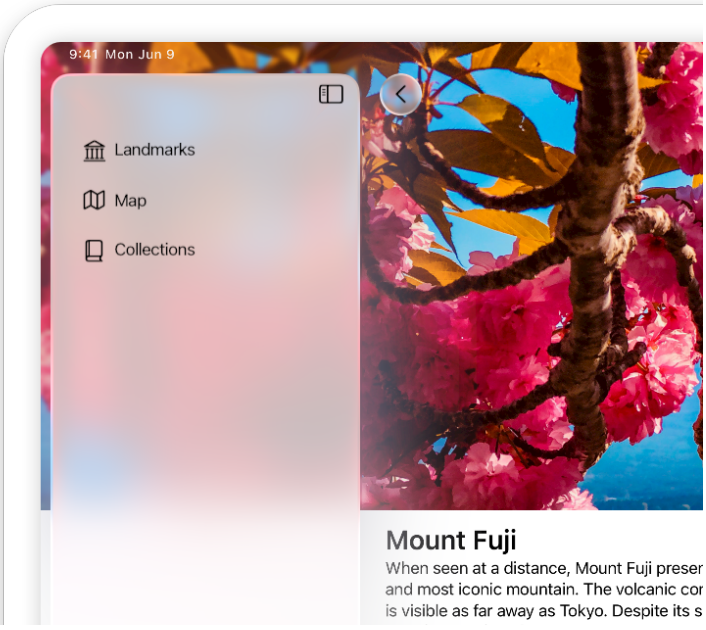
`NavigationSplitView`

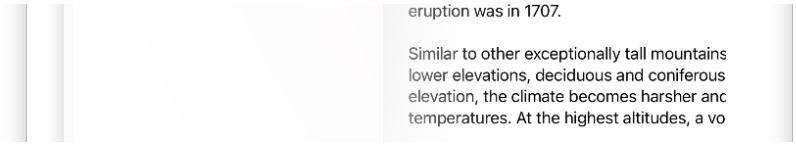
`inspector(isPresented:content:)`

Check content safe areas for sidebars and inspectors. If you have these types of components in your app's navigation structure, audit the safe area compatibility of content next to the sidebar and inspector to help make sure underlying content is peeking through appropriately.

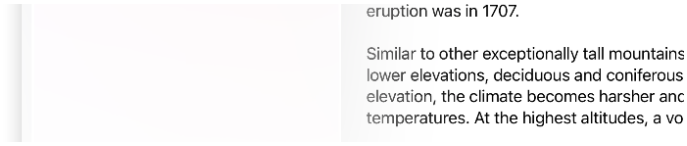
Extend content beneath sidebars and inspectors. A background extension effect creates a sense of extending a background under a sidebar or inspector, without actually scrolling or placing content under it. A background extension effect mirrors the adjacent content to give the impression of stretching it under the sidebar, and applies a blur to maintain legibility of the sidebar or inspector. This effect is perfect for creating a full, edge-to-edge content experience in apps that use split views, such as for hero images on product pages.







Without background extension effect



With background extension effect

SwiftUI UIKit AppKit

`backgroundExtensionEffect()`

Choose whether to automatically minimize your tab bar in iOS. Tab bars can help elevate the underlying content by receding when a person scrolls up or down. You can opt into this behavior and configure the tab bar to minimize when a person scrolls down or up. The tab bar expands when a person scrolls in the opposite direction.

SwiftUI UIKit

```
TabView {
    // ...
}
.tabBarMinimizeBehavior(.onScrollDown)
```

Menus and toolbars

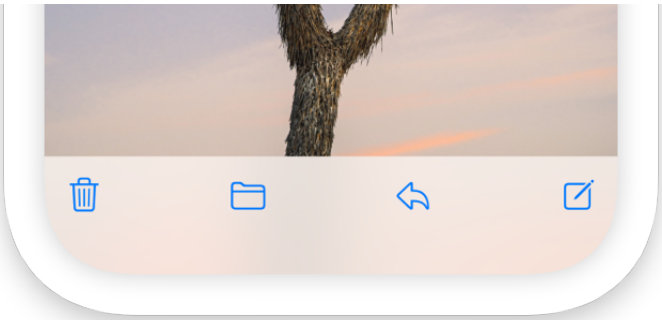
Menus have a refreshed look across platforms. They adopt Liquid Glass, and menu items for common actions use icons to help people quickly scan and identify those actions. New to iPadOS, apps also have a menu bar for faster access to common commands.

Adopt standard icons in menu items. For menu items that perform standard actions like Cut, Copy, and Paste, the system uses the menu item's selector to determine which icon to apply. To adopt icons in those menu items with minimal code, make sure to use standard selectors.

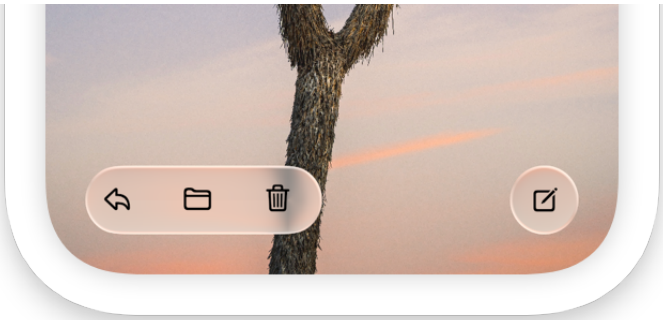
Match top menu actions to swipe actions. For consistency and predictability, make sure

the actions you surface at the top of your contextual menu match the swipe actions you provide for the same item.

Toolbars take on a Liquid Glass appearance, and provide a grouping mechanism for toolbar items, letting you choose which actions to display together.

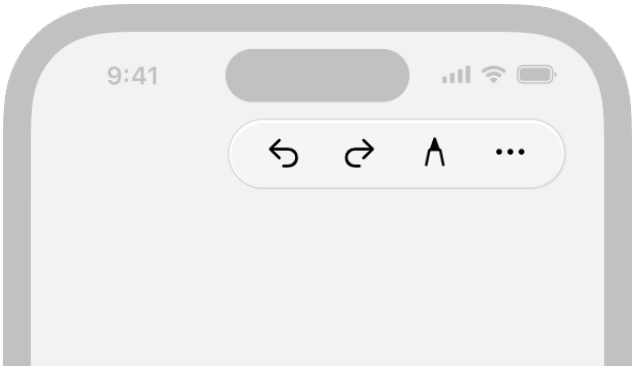


Before

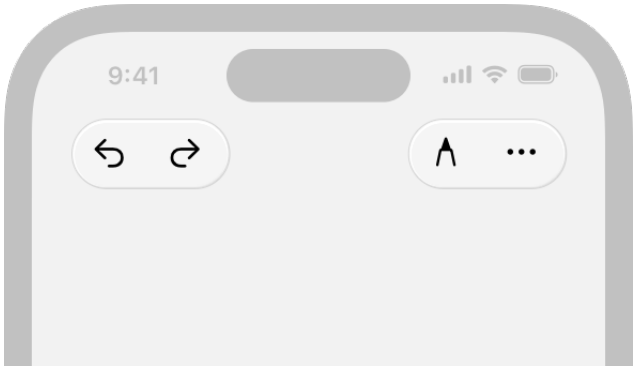


After

Determine which toolbar items to group together. Group items that perform similar actions or affect the same part of the interface, and maintain consistent groupings and placement across platforms.



Incorrect



Correct

You can create a fixed spacer to separate items that share a background using these APIs:

SwiftUI UIKit AppKit

`fixed`

`ToolbarSpacer`

Find icons to represent common actions. Consider representing common actions in toolbars with standard icons instead of text. This approach helps declutter the interface and increase the ease of use for common actions. For consistency, don't mix text and icons across items that share a background.

Provide an accessibility label for every icon. Regardless of what you show in the interface, always specify an accessibility label for each icon. This way, people who prefer a text label can opt into this information by turning on accessibility features like VoiceOver or Voice Control.

Audit toolbar customizations. Review anything custom you do to display items in your toolbars, like your use of fixed spacers or custom items, as these can appear inconsistent with system behavior.

Check how you hide toolbar items. If you see an empty toolbar item without any content, your app might be hiding the view in the toolbar item instead of the item itself. Instead, hide the entire toolbar item, using these APIs:

SwiftUI UIKit AppKit

`hidden(_:)`

Windows and modals

Windows adopt rounder corners to fit controls and navigation elements. In iPadOS, apps show window controls and support continuous window resizing. Instead of transitioning between specific preset sizes, windows resize fluidly down to a minimum size.

Support arbitrary window sizes. Allow people to resize their window to the width and height that works for them, and adjust your content accordingly.

Use split views to allow fluid resizing of columns. To support continuous window resizing, split views automatically reflow content for every size using beautiful, fluid transitions. Make sure to use standard system APIs for split views to get these animations with minimal code:

SwiftUI UIKit AppKit

`NavigationSplitView`

Use layout guides and safe areas. Make sure you specify safe areas for your content so the system can automatically adjust the window controls and title bar in relation to your content.

Modal views like sheets and action sheets adopt Liquid Glass. Sheets feature an increased corner radius, and half sheets are inset from the edge of the display to allow content to peek through from beneath them. When a half sheet expands to full height, it transitions to a more opaque appearance to help maintain focus on the task.

Check the content around the edges of sheets. Inside the sheet, check for content and controls that might appear too close to rounder sheet corners. Outside the sheet, check that any content peeking through between the inset sheet and display edge looks as you expect.

Audit the backgrounds of sheets and popovers. Check whether you add a visual effect view to your popover's content view, and remove those custom background views to provide a consistent experience with other sheets across the system.

An action sheet originates from the element that initiates the action, instead of from the bottom edge of the display. When active, an action sheet also lets people interact with other parts of the interface.

Specify the source of an action sheet. Position an action sheet's anchor next to the control it originates from. Make sure to set the source view or item to indicate where to originate the action sheet and create the inline appearance.

SwiftUI UIKit AppKit

`confirmationDialog(_:isPresented:titleVisibility:presenting:actions:)`

Organization and layout

Style updates to list-based layouts help you organize and showcase your content so it can shine through the Liquid Glass layer. To give content room to breathe, organizational components like lists, tables, and forms have a larger row height and padding. Sections have an increased corner radius to match the curvature of controls across the system.

Before After

Check capitalization in section headers. Lists, tables, and forms optimize for legibility by

adopting title-style capitalization for section headers. This means section headers no longer render entirely in capital letters regardless of the capitalization you provide. Make sure to update your section headers to title-style capitalization to match your app's text to this systemwide convention.

Adopt forms to take advantage of layout metrics across platform. Use SwiftUI forms with the grouped form style to automatically update your form layouts.

Search

Platform conventions for location and behavior of search optimize the experience for each device and use case. To provide an engaging search experience in your app, review these search design conventions.

Search in a toolbar on iPad

Search in a toolbar on iPhone

Check the keyboard layout when activating your search interface. In iOS, when a person taps a search field to give it focus, it slides upwards as the keyboard appears. Test this experience in your app to make sure the search field moves consistently with other apps and system experiences.

Use semantic search tabs. If your app's search appears as part of a tab bar, make sure to use the standard system APIs for indicating which tab is the search tab. The system automatically separates the search tab from other tabs and places it at the trailing end to make your search experience consistent with other apps and help people find content faster.

SwiftUI UIKit

```
Tab(role: .search) {
    // ...
}
```

Platform considerations

Liquid Glass can have a distinct appearance and behavior across different platforms, contexts, and input methods. Test your app across devices to understand how the material looks and feels across platforms.

In watchOS, adopt standard button styles and toolbar APIs. Liquid Glass changes are minimal in watchOS, so they appear automatically when you open your app on the latest release even if you don't build against the latest SDK. However, to make sure your app picks up this appearance, adopt standard toolbar APIs and button styles from watchOS 10.

In tvOS, adopt standard focus APIs. Across apps and system experiences in tvOS, standard buttons and controls take on a Liquid Glass appearance when focus moves to them. For consistency with the system experience, consider applying these effects to custom controls in your app when they gain focus by adopting the standard focus APIs. Apple TV 4K (2nd generation) and newer models support Liquid Glass effects. On older devices, your app maintains its current appearance.

SwiftUI UIKit

`focusable(_:)`

`isFocused`

Combine custom Liquid Glass effects to improve rendering performance. If you apply these effects to custom elements, make sure to combine them using a `GlassEffectContainer`, which helps optimize performance while fluidly morphing Liquid Glass shapes into each other.

Performance test your app across platforms. It's a good idea to regularly assess and improve your app's performance, and building your app with the latest SDKs provides an opportunity to check in. Profile your app to gather information about its current performance and find any opportunities for improving the user experience. To learn more, read Improving your app's performance.

To update and ship your app with the latest SDKs while keeping your app as it looks when

built against previous versions of the SDKs, you can add the `UIDesignRequiresCompatibility` key to your project's Info pane.