Article

Creating views for widgets, Live Activities, and watch complications

Implement glanceable views with WidgetKit and SwiftUI.

Overview

SwiftUI and WidgetKit power widgets, Live Activities, and watch complications. Because they use the same technology and share design similarities, plan your WidgetKit adoption before you start creating these features. Start simple and add complexity later; for example, start by adding a widget extension as described in Creating a widget extension and support one widget size. Spend time to make sure it offers a focused, glanceable experience. Then, add support for additional widget sizes and features like configurability, animations, and interactivity.

Related session from WWDC23

Session 10027: Bring widgets to new places

If you're new to using WidgetKit, see Developing a WidgetKit strategy.

Use system font styles

Widgets, Live Activities, and watch complications appear adjacent to other widgets or complications. As a result, a consistent look for your content that fits in well with the other elements needs to be a priority. To achieve a consistent look for your widgets and complications, use system fonts, default font parameters, and the following font styles:

- `Font.TextStyle.headline`
- `Font.TextStyle.title`

- `Font.TextStyle.body`
- `Font.TextStyle.caption`

Make sure text fits the available space

Widgets and watch complications offer limited space for content — especially on the Lock Screen or on Apple Watch. Give careful consideration to the amount of text you display. For example, say you support the `WidgetFamily.accessoryInline` widget. It can include an image and text. However, the amount of displayable characters varies depending on the context where the widget appears. On Apple Watch, the size of the inline complication varies depending on the watch face. Include it in a `ViewThatFits` view to make sure text always fits the available space.

> **Note**
>
> Test your widgets with every language you support, especially if you support languages that commonly have words with a lot of characters, such as German.

Use content margins instead of safe areas

watchOS 9, iOS 16, iPadOS 16, macOS 13, and earlier use system-defined safe areas to keep content from getting too close to the edge of the widget, complication, or Live Activity. You likely don't change the safe areas that the system defines. However, you might use the `ignoresSafeArea(_:edges:)` view modifier to extend content farther than the safe area.

WidgetKit complications, and Live Activities use content margins instead of safe areas. As a result, `ignoresSafeArea(_:edges:)` has no effect. Instead, use the `contentMarginsDisabled()` view modifier to define custom content margins.

If you use `ignoresSafeArea(_:edges:)`, follow these steps:

1. Add the `contentMarginsDisabled()` view modifier to your widget configuration.
2. For any content that you intend to remain inside system-defined content margins, make use of `padding(_:)` as needed.

> **Tip**
>
> To access the system's default content margins for an environment, use the `widgetContentMargins` environment variable.

See Also

Presentation

- SwiftUI views for widgets

  Present your app's content in widgets with SwiftUI views.