Article

Creating custom views for Live Activities

Create reusable custom views and layouts that support each Live Activity presentation.

Overview

To support Live Activities in your app, you must provide layouts for each Live Activity presentation. Because Live Activities appear automatically across a person's devices, views you use for your Live Activities automatically adapt to provide a default layout. To offer the best user experience and make the most of the available space, create reusable, custom views that automatically adapt for each Live Activity presentation.

Use supplemental activity families

The different presentations are key to creating layouts for your Live Activities. By requiring you to support each presentation, the system can render your Live Activity across devices without you having to add code for each device:

- In StandBy, the system expands the Lock Screen presentation to fill the whole screen.
- On Apple Watch, Live Activities automatically appear in the Smart Stack and as alerts using the compact presentation.
- In CarPlay, Live Activities automatically appear on the Home Screen using the compact presentation.

To offer a great experience in StandBy, on Apple Watch, and in CarPlay, provide flexible custom views to keep people informed about the state of a Live Activity and offer interactivity:

- To tell the system that you provide a custom layout on Apple Watch and in CarPlay, pass `ActivityFamily.small` to the `supplementalActivityFamilies(_:)` view modifier.

- To tell the system that your Live Activity adapts its layout on the Lock Screen of iPhone and iPad, for example in StandBy, pass `ActivityFamily.medium` to the `supplemental ActivityFamilies(_:)` view modifier.

The following example shows how an app might configure a Live Activity that provides different view layouts for small and medium activity families:

```
struct AdventureActivityConfiguration: Widget {
    var body: some WidgetConfiguration {
        ActivityConfiguration(for: AdventureAttributes.self) { context in
            // Create the presentation that appears on the Lock Screen and as a
            // banner on the Home Screen of devices that don't support the
            // Dynamic Island.
            // ...
        } dynamicIsland: { context in
            // Create the presentations that appear in the Dynamic Island.
            DynamicIsland {
                // Create the expanded presentation.
                // ...
            } compactLeading: {
                // Create the compact leading presentation.
                // ...
            } compactTrailing: {
                // Create the compact trailing presentation.
                // ...
            } minimal: {
                // Create the minimal presentation.
                // ...
            }
        }
        ..supplementalActivityFamilies([.small, .medium])
    }
}
```

In your view code, use the `activityFamily` environment value to detect the activity family and change the Live Activity layout accordingly.

The following example shows how a view might respond to different activity families:

```
struct CustomSupplementalActivityView: View {
    @Environment(\.activityFamily) var activityFamily
    var context: ActivityViewContext<EmojiRangersAttributes>
```

```
    var body: some View {
        switch activityFamily {
        case .small:
            SmallSupplementalView(context: context)
        case .medium:
            MediumSupplementalView(context: context)
        }
    }
}
```

> **Note**
>
> Live Activities automatically appear in the Menu bar of a paired Mac, offering interactivity and allowing people to launch your app with iPhone Mirroring. Live Activities that appear in the Menu bar use unchanged compact, minimal, and expanded presentations.

For more information about reusing custom views across widgets and Live Activities, refer to Creating views for widgets, Live Activities, and watch complications.

Set custom content margins

Limiting the content you display is key to offering glanceable, easy-to-read Live Activities. Aim to use the system's default content margins for your Live Activities and only show content that's relevant to the person viewing it. However, you may want to change the system's default content margin to display more content or provide a custom user interface that matches your app. To set custom content margins, use `contentMargins(_:_:for:)`. The following example results in a margin of eight points for the trailing edge of an expanded Live Activity.

```
struct AdventureActivityConfiguration: Widget {
    var body: some WidgetConfiguration {
        ActivityConfiguration(for: AdventureAttributes.self) { context in
            // Create the presentation that appears on the Lock Screen and as a
            // banner on the Home Screen of devices that don't support the
            // Dynamic Island.
            // ...
        } dynamicIsland: { context in
            // Create the presentations that appear in the Dynamic Island.
            DynamicIsland {
                // Create the expanded presentation.
                // ...
            } compactLeading: {
```

```
                // Create the compact leading presentation.
                // ...
            } compactTrailing: {
                // Create the compact trailing presentation.
                // ...
            } minimal: {
                // Create the minimal presentation.
                // ...
            }
            .contentMargins(.trailing, 8, for: .expanded)
            .contentMargins(.all, 20, for: .expanded)
        }
    }
}
```

If you repeatedly use the `contentMargins(_:_:for:)` modifier, the system uses the innermost specified values for a mode.

> **Note**
>
> Avoid placing content too close to the edges of the Dynamic Island.

Use custom colors

Live Activities in the Dynamic Island use a black background color with white text. Use the `keylineTint(_:)` modifier to apply a subtle tint color to the border of the Dynamic Island when the device is in Dark Mode and change text color as needed.

> **Note**
>
> You can't change the background color of Live Activities in the Dynamic Island.

By default, the Lock Screen presentation — including the banner that appears on devices that don't support the Dynamic island — uses a solid white background color in Light Mode and a solid black background color in Dark Mode. To set a custom background color for the Lock Screen presentation, use the `activityBackgroundTint(_:)` modifier. Be sure to choose a color that works well in both Dark or Light Mode or use different background colors that best fit each appearance. To set the translucency of the custom background color, use the `opacity(_:)` view modifier or specify an opaque background color.

If you choose a custom background color for the Lock Screen presentation, use the `activitySystemActionForegroundColor(_:)` view modifier to customize the text color of the auxiliary button that allows people to end the Live Activity on the Lock Screen.

> **Note**
>
> On devices that include an Always-On display, the system dims the screen to preserve battery life and renders Live Activities on the Lock Screen as if in Dark Mode. Use SwiftUI's `isLuminanceReduced` environment value to detect reduced luminance on devices with an Always-On display and use colors and images that look great with reduced luminance.

See Also

User interface

- Adding accessible descriptions to widgets and Live Activities

  Describe the interface elements of your widgets and Live Activities to help people understand what they represent.

- Launching your app from a Live Activity

  Use deep links to enable people to open your app's scene that matches the data of you Live Activity.