All Technologies

WidgetKit

Essentials

Developing a WidgetKit strategy

WidgetKit updates

Creating a widget extension

Emoji Rangers: Supporting Live Activ...

WidgetBundle

System experiences

Widgets and watch complications

Widget creation

Creating a widget extension

Developing a WidgetKit strategy

Emoji Rangers: Supporting Live Ac...

Preparing widgets for additional ...

Widget

WidgetFamily

StaticConfiguration

Configurable widgets

Making a configurable widget

Filter /

Widge... / Widgets and watch compl... / Preparing widgets for additional platforms, contexts, an...

Article

Preparing widgets for additional platforms, contexts, and appearances

Create widgets that support additional platforms and adapt to their context.

Overview

Widgets change their appearance to best fit their context. For example, widgets on the Home Screen, Today View, and on Mac use accented colors and a clear glass presentation with a specific tint color, or they appear in a full color rendering. In CarPlay, small widgets appear scaled up and in full color, while Lock Screen widgets and WidgetKit complications use more restrained colors because they're smaller, always visible, and support tinted modes for each platform. Because all widgets appear in more than one context, make sure your widgets and their SwiftUI views support all applicable rendering modes.

WidgetKit uses three different rendering modes:

`accented`
: Divides the widget's view hierarchy into an accent group and a primary group, and then applies a solid color to each group. Use the `widgetAccentable(_:)` view modifier to group views into the accent group. To learn more about using the `accented` rendering mode and making your widget fit the system's look across Apple platforms, refer to Optimizing your widget for accented rendering mode and Liquid Glass.

`vibrant`
: Desaturates text, images, and gauges into monochrome and creates a vibrant effect by coloring your content appropriately for the Lock Screen background or a macOS desktop. Note that people can also color the Lock Screen to a colored tint and WidgetKit uses a red tint for widgets that appear on iPhone in StandBy in low-light conditions.

`fullColor`
: Doesn't change the color of your complication's views in this rendering mode. Use gradients and full-color images, text, and gauges.

The following table shows the rendering modes for each widget you need to support:

Widget size	`fullColor`	`accented`	`vibrant`
`WidgetFamily.systemSmall`	Yes	Yes	Yes
`WidgetFamily.systemMedium`	Yes	Yes	Yes
`WidgetFamily.systemLarge`	Yes	Yes	Yes
`WidgetFamily.systemExtraLarge`	Yes	Yes	Yes
`WidgetFamily.systemExtraLarge Portrait`	Yes	Yes	Yes
`WidgetFamily.accessory Circular`	No	Yes	Yes
`WidgetFamily.accessoryCorner`	No	Yes	Yes
`WidgetFamily.accessory Rectangular`	Yes	Yes	Yes
`WidgetFamily.accessoryInline`	No	Yes	Yes

> Note
> For design guidance, see Human Interface Guidelines > Complications and Human Interface Guidelines > Widgets.

In your code, read the `widgetRenderingMode` environment variable to create SwiftUI views for each applicable rendering mode, as shown in the following example:

```
// ...

@Environment(\.widgetRenderingMode) var renderingMode

var body: some View {
    ZStack {
        switch renderingMode {
        case .fullColor:
            // Create views for full-color widgets and watch complicat
        case .accented:
            // Create views and group applicable views in the accented
            VStack {
                // ...
            }
            .widgetAccentable()
            // Additional views that you don't group in the accented g
        case .vibrant:
            // Create views for Lock Screen widgets on iPhone and iPad
    }
}
```

Make your background views removable

In many contexts, the system removes your widget's background views to match the system appearance. To control which components of your widget the system removes, group them into background containers and mark them as removable. For more information, refer to Displaying the right widget background.

Support Always On

If you create accessory widgets and WidgetKit complications, make sure to use SwiftUI's `isLuminanceReduced` environment variable to detect Always On and color your views to look great with reduced luminance. For design guidance, see Human Interface Guidelines > Always On and Human Interface Guidelines > Widgets.

Update your small widget to support StandBy and CarPlay

On iPhone in StandBy, the Lock Screen shows two widgets side by side on a dark background. In CarPlay, widgets appear on a dedicated Widgets screen. For both appearances, WidgetKit uses your `widgetFamily.systemSmall` widget and scales it to fit available space. For more information about supporting StandBy and CarPlay, refer to Adding StandBy and CarPlay support to your widget.

Indicate that a widget might not fit a specific context

By default, the system suggests widgets in many contexts, and people can choose widgets in the widget gallery to personalize their system experience. However, a widget might not be a good fit for a specific context. For example, a widget that relies on high-resolution photos and background colors for its functionality may not work well on the Lock Screen, where the system applies a vibrant treatment to the widget. To indicate that a widget doesn't work well in a specific context, use `disfavoredLocations(_:for:)` and provide the applicable `WidgetLocation`. As a result, the widget appears in the widget gallery's "Other" section for the disfavored location.

Verify font sizes in macOS

When people place an iPhone widget on a Mac desktop, the system renders it using iOS font metrics. However, a native Mac app's widgets use macOS font sizing. If you're sharing code across your iOS and macOS targets in your Xcode project, make sure to verify font sizing in macOS and adjust it for macOS as needed.

See Also

Widget creation

Creating a widget extension
Display your app's content in a convenient, informative widget on various devices.

Developing a WidgetKit strategy
Explore features, tasks, related frameworks, and constraints as you make a plan to implement widgets, controls, watch complications, and Live Activities.

Emoji Rangers: Supporting Live Activities, interactivity, and animations
Offer Live Activities, controls, animate data updates, and add interactivity to widgets.

`protocol Widget`
The configuration and content of a widget to display on the Home screen or in Notification Center.

`enum WidgetFamily`
Values that define the widget's size and shape.

`struct StaticConfiguration`
An object describing the content of a widget that has no user-configurable options.