Article

# 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.

## Overview

Use WidgetKit to build widgets, controls, watch complications, and Live Activities. When you offer these system experiences, your app becomes part of the widget ecosystem across platforms and devices, and expands its reach by taking up limited but effective, eye-catching space. System experiences that use WidgetKit as a foundation rely on a set of related frameworks and share design and functional similarities, making them great candidates for code and design component reuse.

To avoid costly changes in your app's development process, plan your WidgetKit adoption before you create designs and write code. As you make your plans, take into account:

- Feature availability for each platform
- Frameworks to use in addition to WidgetKit
- Required appearances and available sizes for widgets, watch complications, and Live Activities
- Technology that powers content updates
- Animation
- Interactivity with your app through deep links, buttons, and toggles
- Configuration options for widgets and watch complications
- Visibility in Smart Stacks
- Functional constraints

Then, approach WidgetKit adoption iteratively. For example, start with a nonconfigurable `WidgetFamily.systemSmall` widget as described in Creating a widget extension because it gives your content broad exposure in the WidgetKit ecosystem on iPhone, iPad, Mac, and Apple Vision Pro. Then, add support for configuration, additional widget sizes, and — depending on your app's features — Live Activities or a watchOS app with watch complications.

## Review system experiences for each platform

Widgets come in different sizes, from circular accessory widgets on the Lock Screen and complications on Apple Watch to extra-large widgets on Apple Vision Pro. You can choose the sizes and complications you want to support, but consider supporting as many sizes and complications as possible.

Live Activities are available on iPhone and iPad and appear on the Lock Screen and in the Dynamic Island on supported devices. Additionally, Live Activities appear on a paired Mac or Apple Watch, and in CarPlay. When you add support for Live Activities, you need to create minimal, compact, and extended presentations that make sure your Live Activities appear correctly for each platform.

In iOS, iPadOS, and macOS, your app can offer controls people place in Control Center. On Mac, people can also place controls on the menu bar as menu bar items. On Apple Watch, controls from your watchOS app or a paired iPhone appear in Control Center and the Smart Stack, and people can place them on the Action button of Apple Watch Ultra.

This table shows the functionality available for each platform:

| Widget size or technology | iPhone | iPad | Apple Watch | Mac | Apple Vision Pro |
|---|---|---|---|---|---|
| Small system widgets | Home Screen, Today View, and StandBy | Home Screen, Today View, and Lock Screen | No | Yes | Yes |
| Medium system widgets | Home Screen and Today View | Home Screen and Today View | No | Yes | Yes |
| Large system widgets | Home Screen and Today View | Home Screen and Today View | No | Yes | Yes |

| Widget size or technology | iPhone | iPad | Apple Watch | Mac | Apple Vision Pro |
|---|---|---|---|---|---|
| Extra large system widgets | No | Home Screen and Today View | No | Yes | Yes |
| Extra large system widgets (portrait) | No | No | No | No | Yes |
| Circular accessory widgets | Lock Screen | Lock Screen | Watch complications and Smart Stack | No | No |
| Corner accessory widgets | No | No | Watch complications | No | No |
| Rectangular accessory widgets | Lock Screen | Lock Screen | Watch complications and Smart Stack | No | No |
| Inline accessory widgets | Lock Screen | Lock Screen | Watch complications | No | No |
| Live Activities | Yes | Yes | From a paired iPhone | From a paired iPhone | No |
| Controls | Yes | Yes | Yes | Yes | No |

## Leverage additional frameworks

Widgets, watch complications, controls, and Live Activities use a widget extension you add to your Xcode project. The role of WidgetKit is to provide the infrastructure and configuration for the features it enables. Based on features and platforms you support, use WidgetKit in combination with other frameworks as follows:

- To create the user interface for each feature, use SwiftUI.
- To add interactivity to widgets and Live Activities, use SwiftUI and the App Intents framework.

- To offer watch complications and watchOS widgets, create a watchOS app.
- To offer configurable widgets and watch complications, use App Intents.
- To provide the contextual clues that the system uses for Smart Stacks and to offer Widget Suggestions, use App Intents and RelevanceKit.
- To start, update, and end Live Activities, use ActivityKit.

## Support different appearances

Depending on the context, a widget or Live Activity changes its appearance to best fit its context. For example, a `WidgetFamily.systemSmall` widget appears as follows:

- On the Home Screen of iPhone and iPad, it uses it uses the `accented` rendering mode for light and dark appearances, and `fullColor` on devices that run iOS and iPad 18 or older.
- On the Lock Screen of iPad and iPhone, it uses the `vibrant` rendering mode that provides a vibrant, blurred appearance. On the Lock Screen of iPhone in StandBy and StandBy in Night Mode, it renders scaled up in size using the `vibrant` rendering mode.
- In CarPlay, it renders scaled-up in size using the `fullColor` rendering mode with the background removed.
- On Mac, it uses the `accented` rendering mode. On older versions of macOS, it uses the `fullColor` or `vibrant` modes.

Similarly, the `WidgetFamily.accessoryRectangular` widget appears as follows:

- On the Lock Screen of iPhone and iPad, it takes on the `vibrant` appearance.
- On Apple Watch, it appears as a watch complication without a background and the `accented` appearance and in a `fullColor` appearance in the Smart Stack.

With each feature you add to your app, make sure your widget, watch complication, or Live Activity supports all applicable contexts and appearances well. For more information, refer to Preparing widgets for additional platforms, contexts, and appearances. For design guidance, refer to Human Interface Guidelines > Widgets.

## Animate content updates

Widgets and Live Activities can use animations to draw a person's attention to data updates, inclusing custom animations. For more information, refer to Animating data updates in widgets and Live Activities.

## Provide up-to-date information

Widgets and watch complications use a different mechanism than your app to update their content. They use a timeline of data updates that you create in your app and hand to WidgetKit. You maintain this timeline as your app receives new data, but, to optimize battery life for a device, each app has a budget to update its widgets or complications. Additionally, the system batches and schedules updates to preserve power. For more information on how timelines work and how you can keep your widgets and watch complications up to date, refer to Keeping a widget up to date and Making network requests in a widget extension. Additionally, widgets can update their data with the Apple Push Notification service (APNs) and WidgetKit push notifications.

Live Activities don't use timelines to update their content. Instead, they use ActivityKit and ActivityKit push notifications you send with APNs. For more information, refer to ActivityKit.

Controls also don't use timelines to update their content. Instead, they update their content when someone uses them, the app reloads them, or the system receives a remote push notification from APNs. For more information, refer to Updating controls locally and remotely.

## Add specific app functionality to your widgets and Live Activities

By default, people tap a widget, watch complication, or Live Activity to launch its corresponding app. To make the experience more intuitive and require fewer interactions, you can use deep linking to launch the scene of your app that matches the widget's visible content. Widgets that offer enough space, such as `WidgetFamily.systemSmall` or larger — and Live Activities in the extended or the Lock Screen appearance — add SwiftUI's `Link` to your views and allow people to open different screens in your app.

> **Note**
>
> In iOS 16 and macOS 13 or earlier versions, only large and extra-large widgets can use `Link`.

Widgets offer direct interaction with your app using the App Intents framework and SwiftUI. Both `Button` and `Toggle` offer dedicated initializers for this purpose. For more information, refer to Adding interactivity to widgets and Live Activities.

## Offer configurable widgets and watch complications

Make it possible for people to select the information they want to view in the widget or a watch complication by offering configurable widgets and complications that provide customizable properties. For example, people might choose to stay informed about a specific stock in a stock market widget, or enter a tracking number for a package delivery widget. Configurable widgets and complications use the App Intents framework and custom intents you define — the same

mechanism you use to support system-level services like Siri and the Shortcuts app. For information about creating configurable widgets and complications, refer to Making a configurable widget.

## Increase visibility in Smart Stacks

On iPhone and iPad, people create stacks of widgets and swipe through them manually. Additionally, people use Smart Stacks with Smart Rotate to view the most relevant widgets and see widget suggestions. To show relevant widgets at the top of Smart Stacks, iOS and iPadOS rely on behavioral clues that apps provide during use. On Apple Watch, people can place and pin widgets in the Smart Stack, but they rely more heavily on the system to automatically display contextually relevant widgets. To determine the most relevant widgets, watchOS queries your widget extension for contextual clues.

For more information, refer to Increasing the visibility of widgets in Smart Stacks.

## Consider user privacy

The Lock Screen and watch faces are always visible, and people can configure widgets and complications to hide sensitive information when the device is locked or supports Always On. Review data that appears on your widget, Live Activity, or complication, and make sure you support redaction of sensitive data. For additional information, refer to Creating a widget extension.

## Store shared data in a group container

To add widgets, watch complications, and Live Activities, you create a widget extension and add it to your app, and the extension target and your app are part of the same app group. As a result, you can store files and data in a shared container that's accessible to the app and the widget extension. For example, your app can download data and store it in a database in the shared container, and then a widget can access the database.

For additional information about app groups and accessing a shared container, refer to Configuring app groups.

## Respect functional constraints

Widgets, watch complications, and Live Activities are always visible. To preserve battery life and user privacy, they follow certain constraints. For example, Live Activities can't access a person's location. The following table shows availability features that impact battery life or user privacy for each feature:

| Functionality | Widgets | Watch complications | Live Activities |
| --- | --- | --- | --- |
| Network access | Yes | Yes | No |
| Location access | Yes | Yes | No |

For additional information, refer to Accessing location information in widgets.

# See Also

## Essentials

WidgetKit updates

Learn about important changes in WidgetKit.

Creating a widget extension

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

{} Emoji Rangers: Supporting Live Activities, interactivity, and animations

Offer Live Activities, controls, animate data updates, and add interactivity to widgets.

`@MainActor @preconcurrency protocol WidgetBundle`

A container used to expose multiple widgets from a single widget extension.