Article

Creating your first app intent

Create your first app intent that makes your app available in system experiences like Spotlight or the Shortcuts app.

Overview

To let people leverage your app's features outside of the app itself, system experiences like Spotlight and the Shortcuts app require your help to understand your app's actions and content so the system can expose that functionality. Use App Intents to express your app's capabilities and make your app's actions available to the system. App intents are self-contained types that act as a bridge between your code and system experiences and services. Each app intent encapsulates a single action that's specific to your app. It provides the system with any action that makes sense for your app's audience, such as showing information about a hiking trail from a hiking app, exporting a person's transaction history from a budgeting app, or converting between two specific units of measurement with a converter app.

Every app intent provides descriptive information about itself that experiences and services like Siri can display or announce. When you build an app that contains app intents, the compiler examines your source and generates data about those intents that Xcode stores in the app bundle. After someone installs your app, the system uses that data to discover the intents and makes them available to the system.

Before you get started creating your first app intent, read Making actions and content discoverable and widely available to review App Intents framework features and functionality. Then, identify an action and create your first app intent, and offer an App Shortcut as described below. App Shortcuts make your app intent even more useful. For example, App Shortcuts don't require configuration, and people can place them on the Action button. Additionally, App Shortcuts appear in Spotlight even when a person hasn't launched your app.

Identify an action

Think about actions and tasks people perform in your app, an action's input and output data, and how the system could surface actions in its services and experiences. In general, implement your `AppIntent` to have a narrow focus and do one thing well. People can invoke it individually, or create custom shortcuts by combining it with app intents from other apps in the Shortcuts app.

For your first app intent, choose an action that people are likely to use frequently. Then, add an App Shortcut that includes the app intent.

> **Tip**
>
> To get familiar with the App Intents framework, consider creating your first app intent for functionality that doesn't use a specialized app intent protocol; for example, an app intent that opens your app. When you've successfully created your first app intent, make changes to adopt a specialized app intent or add more app intents for more complex app functionality.

Review when to adopt specialized app intent protocols

For many app intents, the `AppIntent` protocol is the preferred protocol to adopt. However, depending on your app's specific behaviors, you might prefer your code to conform to one of the other intent protocols; for example:

- Create app intents that conform to assistant schemas that make sure your actions and content work well with the enhanced action capabilities of Siri that Apple Intelligence provides.
- If your app plays or records audio and you want to offer that same functionality in an app intent, adopt `AudioPlaybackIntent` instead. This protocol inherits from `AppIntent` and indicates the audio-related behavior to the system so that, where possible, it avoids audio interactions and other potential interruptions.

The App Intents framework provides a number of other specialized app intent protocols. For more information about integrating your app intents with Siri and Apple Intelligence, see Integrating actions with Siri and Apple Intelligence and App intent domains. For more information about other specialized protocols, see App intents.

Create an app intent that opens your app

To define an action, create a type that adopts the `AppIntent` protocol, or a related protocol that provides the specific behavior you need. If possible, start with a simple action that doesn't require parameters. Alternatively, if your action requires a parameter, consider initially hard-coding the parameter to get your first app intent implementation to work. Then make changes to add parameters to your first app intent as described in Adding parameters to an app intent.

For example, the Accelerating app interactions with App Intents sample code project provides an app intent that opens the app and displays a person's favorite hiking trails:

```
struct OpenFavorites: AppIntent {

    static var title: LocalizedStringResource = "Open Favorite Trails"

    static var description = IntentDescription("Opens the app and goes to your favor

    static var openAppWhenRun: Bool = true

    @MainActor
    func perform() async throws -> some IntentResult {
        navigationModel.selectedCollection = trailManager.favoritesCollection

        return .result()
    }

    @Dependency
    private var navigationModel: NavigationModel

    @Dependency
    private var trailManager: TrailDataManager
}
```

In the structure, implement the protocol's `title` requirement to provide the localized text that the Shortcuts app displays in its Action Library and shortcut editor. To include additional context for the intent, implement the optional `description` requirement to provide localized text that describes the app intent's behavior. The Shortcuts app shows the description in its Action Library.

Perform the app intent's action

To provide your intent's functionality, implement the `perform()` protocol requirement. The system invokes this method after it resolves any required parameters, meaning those parameters are safe for your code to access from the function's body.

Your implementation must complete the necessary work and return a result to the system. A result may include, among other things, a value that a shortcut can use in subsequent connected actions, dialogue to display or announce, and a SwiftUI snippet view.

For example, the Accelerating app interactions with App Intents sample code project returns a dialog for the `GetTrailInfo` app intent:

```
func perform() async throws -> some IntentResult & ReturnsValue<TrailEntity> & Provi
    guard let trailData = trailManager.trail(with: trail.id) else {
        throw TrailIntentError.trailNotFound
    }

    /**
     You provide a custom view by conforming the return type of the `perform()` func
     */
    let snippet = TrailInfoView(trail: trailData, includeConditions: true)

    /**
     This intent displays a custom view that includes the trail conditions as part o
     the system can only read the response, but not display it. When the system can
     conditions.
     */
    let dialog = IntentDialog(full: "The latest conditions reported for \(trail.name
                              supporting: "Here's the latest information on trail co

    return .result(value: trail, dialog: dialog, view: snippet)
}
```

If it doesn't make sense for your intent to return a concrete result, return `.result()` to tell the system the intent is complete.

Important

By default, the system launches your app in a limited mode in the background and executes the intent's `perform()` method on an arbitrary queue. To override this behavior and launch the app in the foreground, set the intent's `openAppWhenRun` variable to `true`. If your intent updates the app's user interface, annotate `perform()` with `@MainActor` to make sure the method executes on the main queue.

Verify the behavior of your intent in Simulator or on-device

During development, validate that your intents behave as you expect by testing them in Simulator or on-device. If you're adding intents to a macOS app, build and run the app. For other platforms, select the relevant simulator or connected device and then build and run. After your app launches, follow these steps:

1. Launch the Shortcuts app.

2. Tap or click the New Shortcut (+) button to create a shortcut.

3. Choose Apps in the Action Library's segmented control.

4. Tap or click your app's icon.

5. Select the action to test.

6. For app intents with parameters, use the summary to set the parameter values.

7. Tap or click the Run button.

Set a breakpoint at the top of your `perform()` method to confirm your implementation is working. The debugger pauses execution immediately after you run the shortcut, enabling you to step through the code and inspect the intent's parameters to verify they have the values they require.

Design custom responses

People may interact with your app intent through Siri. For a good user experience, consider communicating the intent's result with a visual response using a custom UI snippet, and as a dialog for Siri to communicate the same information. For more information, see Design custom responses.

Receive input with parameters and return results

Creating an app intent that opens a screen in your app is the first step to becoming familiar with the App Intents framework and to making your app and its content discoverable. Many actions in your app receive input and return data. To describe actions that receive and return data, add parameters to the app intent to tell the system about that data and whether it's required or optional. By exposing parameters, you enable people to configure your intents with values unique to their requirements and enable the App Intents framework to communicate with system experiences to write those values at runtime. For example, the Accelerating app interactions with App Intents sample code project enables people to choose which hiking trail information to view when they invoke an app intent. For more information about using parameters in an app intent, see Adding parameters to an app intent.

Create an App Shortcut

App intents you create appear in the Shortcuts app. People can create custom shortcuts that initiate your app intent and combine app intents to perform custom workflows. To enable people to discover and run your app intents without any configuration, bundle your app's app intents into App Shortcuts to provide workflows of your app's actions.

By offering App Shortcuts, you make your app's functionality instantly available for use in Shortcuts, Spotlight, and Siri from the moment a person installs your app — without any setup in

the Shortcuts app or an Add to Siri button. On devices that support the Action button, people can invoke your App Shortcut with the Action button for quick access to your app's functionality.

To offer an App Shortcut for your first app intent:

1. Create the `AppShortcut` object for your app intent using the `init(intent:phrases:shortTitle:systemImageName:)` initializer and provide phrases that people can use to run the app intent and the metadata that appears in the Shortcuts app.
2. Implement the `AppShortcutsProvider` protocol that provides the App Shortcuts you offer to the Shortcuts app.

For more information about creating App Shortcuts, see App Shortcuts and Create App Shortcuts.

To learn more about supporting the Action button, refer to Action button on iPhone and Apple Watch.

Donate app intents to the system

Make your app intents discoverable by explicitly donating them to the system. When someone performs an action in your app, donate an intent that corresponds to that action. The system uses the information you provide to predict actions someone might take in the future. For example, if someone requests the weather from your app each morning, the system might proactively offer the corresponding app intent at the same time each day.

For more information, see Intent discovery.

See Also

Essentials

App Intents updates

Learn about important changes in App Intents.

Making actions and content discoverable and widely available

Adopt App Intents to make your app discoverable with Spotlight, controls, widgets, and the Action button.

{} Adopting App Intents to support system experiences

Create app intents and entities to incorporate system experiences such as Spotlight, visual intelligence, and Shortcuts.

{} Accelerating app interactions with App Intents

Enable people to use your app's features quickly through Siri, Spotlight, and Shortcuts.