

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 favorite trails")  
    static var openAppWhenRun: Bool = true  
    @MainActor  
    func perform() async throws -> some IntentResult {  
        navigationModel.selectedCollection = trailManager.favoriteCollection  
        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> & ProvidesDial  
    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()` function to  
     * `ProvidesDial`.  
     */  
    let snippet = TrailInfoView(trail: trailData, includeConditions: true)  
  
    /**  
     * This intent displays a custom view that includes the trail conditions as part of the view.  
     * The system can only read the response, but not display it. When the system can display  
     * conditions.  
     */  
    let dialog = IntentDialog(full: "The latest conditions reported for \(trail.name) indicate  
        supporting: \(Here's the latest information on trail condition  
        \(trailData.conditions))")  
    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.