Sample Code

# Soup Chef: Accelerating App Interactions with Shortcuts

Make it easy for people to use Siri with your app by providing shortcuts to your app's actions.

Download

iOS 14.3+ | iPadOS 14.3+ | Mac Catalyst 14.3+ | watchOS 7.0+ | Xcode 17.0+

## Overview

Soup Chef is the fictitious app from this sample code project. The project shows you how to add shortcuts and personalized phrases to Siri. With shortcuts, users can access features of your app from places such as Spotlight search and the Lock screen. With phrases, users can speak to Siri to access your app's features. For example, a user of Soup Chef may order a bowl of clam chowder by saying to Siri, "Order clam chowder."

## Configure the Sample Code Project

Before you can run Soup Chef, you need to:

1. Set the app group name for the SoupChef, SoupChefIntents, SoupChefWatch Extension, and SoupChefIntentsWatch targets to a valid name. For more information on app groups, see Configure app groups.

## Share Code Between the App and App Extension

The Soup Chef project contains targets for an app and an Intents app extension, which the system uses to handle shortcuts that run in the background. To avoid duplicating code that is common

across the two targets, the project includes a shared framework called SoupKit. This framework provides a central location for shared code responsible for tasks such as data management and donating shortcuts. For more information about structuring code, see Structuring Your Code to Support App Extensions.

## Identify Common Actions

Soup Chef has a key feature that people are likely to use frequently: ordering soup. To make it easy for users to place an order for soup, it makes sense for the app to add a shortcut for this feature for use with Siri. By adding the shortcut, Siri can make suggestions to the user based on their previous order history. For example, say the user likes to order tomato soup every Monday around noon. After placing this order a few times using Soup Chef, Siri learns the user's pattern, and can use that information to predicate when the user might place a similar order. On the following Monday, Siri suggests to the user that they might want to order a bowl of tomato soup from Soup Chef.

## Define Custom Intents

To create a shortcut, the app needs an `OrderSoup` intent. However, the system doesn't provide this type of intent, so Soup Chef provides a custom intent.

Define a custom intent by first adding an Intent Definition file to your Xcode project. Then add the custom intent to the definition file. Soup Chef defines its custom intent as `OrderSoup`. The intent has three parameters:

- `soup`: The type of soup.
- `quantity`: The number of bowls of soup.
- `toppings`: Optionals toppings–such as cheese or croutons–to add to the soup.

Soup Chef uses the three parameters to define the different shortcut types. A shortcut type has a title, subtitle, and a set of parameter combinations such as:

- `soup` and `quantity`
- `soup` and `toppings`
- `soup`, `toppings`, and `quantity`

These types define the schema that Siri uses to identify requests the user makes; for example, "Order tomato soup with cheese." The parameter combination for this example is: `soup` and `toppings`.

Soup Chef marks each shortcut type with *Supports background execution*; this way, the system doesn't need to open the app to handle the intent. When marked this way, the system uses Soup

Chef's Intents app extension to handle the order. This provides a better user experience because the user never leaves their current screen.

## Define Custom Responses

The `OrderSoup` intent includes a set of custom responses that Siri shows to the user. Siri selects the response based on a code returned by the Intents app extension when it confirms or handles the intent. For example, if the user orders a cup of chili and chili isn't on the menu that day, the extension returns the `failureOutOfStock` code. Siri then responds to the user with "Sorry, we're all out of chili."

```
if menuItem.attributes.contains(.available) == false {
    //  Here's an example of how to use a custom response for a failure case when a
    completion(OrderSoupIntentResponse.failureOutOfStock(soup: soup))
    return
}
```

## Share Intents Between Targets

To use a custom intent in an app, Xcode needs to generate source code for the items defined in the Intent Definition file. However, for apps that have a shared framework, like Soup Chef, code generation only needs to happen for the framework target. Generating the code for the shared framework and app causes conflicts due to duplicated code.

To specify which target has the generated source code, use the Intent Classes setting in the Target Membership panel. For Soup Chef, only the SoupKit target uses the Intent Classes setting; all other targets use the No Generated Classes setting. And because SoupKit shares its code with other targets, the other targets can use `OrderSoupIntent`.

## Donate Shortcuts

Before Siri can suggest shortcuts to the user, the app must tell Siri about the shortcuts through intent donations. An app makes a donation each time the user performs an action in the app. Soup Chef, for example, donates an `OrderSoupIntent` each time the user places an order through the app. Donating the intent each time the user places the order helps Siri learn about the user's behavior, which helps Siri better predicate when the user may want to order soup again.

```
private func donateInteraction(for order: Order) {
    let interaction = INInteraction(intent: order.intent, response: nil)
```

```
    // Store the order identifier with the interaction so that it is easy
    // to locate an interaction from an order. When a soup is removed from
    // the menu, the app deletes the interaction so the soup isn't suggested
    // to the user.
    interaction.identifier = order.id.uuidString

    interaction.donate { (error) in
        if error != nil {
            if let error = error as NSError? {
                Logger().debug("Interaction donation failed: \(error)")
            }
        } else {
            Logger().debug("Successfully donated interaction")
        }
    }
}
```

## Handle Shortcuts

As mentioned earlier, to handle a order soup intent, Soup Chef provides an Intents app extension, which handles the intent in the background. This makes it possible for the user to remain on the current screen – such as the Lock screen – while Soup Chef places the order in the background.

There are times, however, when the app must launch to handle the intent; for example, when the user taps the shortcut in Search. That's why Soup Chef also handles the order soup intent in its `scene(_ scene: UIScene, continue userActivity: NSUserActivity)` implementation found in the scene delegate.

```
/// The system calls this method when continuing a user activity through the restora
/// in `UISceneDelegate scene(_:continue:)`.
override func restoreUserActivityState(_ activity: NSUserActivity) {
    super.restoreUserActivityState(activity)

    if activity.activityType == NSUserActivity.viewMenuActivityType {

        // This order came from the "View Menu" shortcut that is based on NSUserActi
        prepareForUserActivityRestoration() {
            self.performSegue(withIdentifier: SegueIdentifiers.soupMenu.rawValue, se
        }

    } else if activity.activityType == NSStringFromClass(OrderSoupIntent.self) {

        // This order started as a shortcut, but isn't complete because the user tap
```

```
        // during the order process. Let the user finish customizing their order.
        prepareForUserActivityRestoration() {
            self.performSegue(withIdentifier: SegueIdentifiers.soupMenu.rawValue, se
        }

    } else if activity.activityType == NSUserActivity.orderCompleteActivityType,
              let orderID = activity.userInfo?[NSUserActivity.ActivityKeys.orderID.ra
              let order = soupOrderManager.order(matching: orderID) {

        // This order was just created outside of the main app through an intent.
        // Display the order in the order history.
        prepareForUserActivityRestoration() {
            self.displayOrderDetail(order)
        }
    }
}
```

## Add Phrases to Siri

Ordering soup based on suggestions from Siri is a great start to expediting soup orders, but Soup Chef goes one step further by letting the user set a voice phrase for a particular order, and adding that phrase to Siri. Afterward, the user can ask Siri to place the order by saying the phrase. For example, the user can add to Siri the phrase, “Clam chowder time,” for the shortcut that orders clam chowder with croutons. The next time the user craves clam chowder, they say to Siri, “Clam chowder time,” and Siri tells Soup Chef to place an order for clam chowder with croutons.

Users can set custom phrases in the Shortcuts app. To make the experience better, however, Soup Chef provides the option to add the phrase directly from the app. From the order history, the user can tap a previous order to view its details. At the bottom of the order details is an *Add to Siri* button that, when tapped, lets the user set a new phrase. Soup Chef also provides a suggested phrase to help inspire the user.

```
addShortcutButton = INUIAddVoiceShortcutButton(style: .automaticOutline)
addShortcutButton.shortcut = INShortcut(intent: orderSoupIntent)
addShortcutButton.delegate = configuration.delegate
```

If a phrase already exists for the shortcut, the button displays the current phrase, and allows the user to change the phrase directly from Soup Chef by presenting a view controller to edit or delete the current phrase.

```
func present(_ editVoiceShortcutViewController: INUIEditVoiceShortcutViewController,
    editVoiceShortcutViewController.delegate = self
    present(editVoiceShortcutViewController, animated: true, completion: nil)
}
```

Adding shortcuts and phrases are two examples of how your app can accelerate user interactions with Siri. For more information on how your app can leverage Siri, see the SiriKit developer documentation.

# See Also

## Sample code

{} Adding Shortcuts for Wind Down

Reveal your app's shortcuts inside the Health app.

{} Booking Rides with SiriKit

Add Intents extensions to your app to handle requests to book rides using Siri and Maps.

{} Handling Payment Requests with SiriKit

Add an Intent Extension to your app to handle money transfer requests with Siri.

{} Handling Workout Requests with SiriKit

Add an Intent Extension to your app that handles requests to control workouts with Siri.

{} Integrating Your App with Siri Event Suggestions

Donate reservations and provide quick access to event details throughout the system.

{} Managing Audio with SiriKit

Control audio playback and handle requests to add media using SiriKit Media Intents.

{} Providing Hands-Free App Control with Intents

Resolve, confirm, and handle intents without an extension.

{} Soup Chef with App Intents: Migrating custom intents

Integrating App Intents to provide your app's actions to Siri and Shortcuts.