Sample Code

Handling Workout Requests with SiriKit

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

Download

iOS 14.1+ | iPadOS 14.1+ | Xcode 17.0+

Overview

The app in this sample project allows users to start, pause, resume, cancel, and end custom wall-climbing or bouldering workouts with Siri, as well as view historical workout data inside the main app.

The project consists of three targets:

- `Ascent`, an iOS app that displays a history of climbing and bouldering workouts.
- `AscentIntentsExtension`, an Intent Extension that integrates with SiriKit to control workouts.
- `AscentFramework`, an embedded framework containing shared code needed by both the Intent Extension and the main app.

See Creating an Intents App Extension for more information on the general process of adding an Intent Extension to your app, including how to enable the Siri capability and configure the `NSExtension` keys in `Info.plist`.

Configure the Sample Code Project

This sample app can be run in the iOS Simulator without any special setup, but in order to run it on a device you will need to update the build settings and enable an App Group for the project:

1. Open `Ascent.xcodeproj` with the latest version of Xcode.
2. In the project editor, set a new bundle identifier under Identity on the General pane for each of the three targets in the project.
3. In the Capabilities pane, make sure that App Groups is switched on for the `Ascent` and `Ascent IntentsExtension` targets.
4. Add an App Group identifier with the format `group.com.example`.
5. Reference the new App Group identifier in the source code. Open `WorkoutHistory.swift` and modify the implementation of the `sharedUserDefaults` property to reference the new identifier.

> **Note**
>
> Each target in the project must preface its bundle identifier with the bundle identifier of the app's main target. For example, if you set the bundle identifier for the `Ascent` target to `com.example.Ascent`, set the bundle identifier for the `AscentIntentsExtension` target to `com.example.Ascent.AscentIntentsExtension`.

Recognize and Respond to Workout Requests

In order for the Intent Extension to handle workout requests, the principal class of the extension must implement the `handler(for:)` method of `INIntentHandlerProviding`. Use this method to dispatch the request to a class configured to handle it.

```
override func handler(for intent: INIntent) -> Any {
    for handler in intentHandlers where handler.canHandle(intent) {
        return handler
    }
    preconditionFailure("Unexpected intent type")
}
```

In order for a class to handle a request from SiriKit to use the workout controls, the class must implement one of the workout intent handler protocols. In this sample project:

- `StartWorkoutIntentHandler` implements `INStartWorkoutIntentHandling`.
- `PauseWorkoutIntentHandler` implements `INPauseWorkoutIntentHandling`.
- `ResumeWorkoutIntentHandler` implements `INResumeWorkoutIntentHandling`.
- `CancelWorkoutIntentHandler` implements `INCancelWorkoutIntentHandling`.

- `EndWorkoutIntentHandler` implements `INEndWorkoutIntentHandling`.

Resolve Parameters

The first step in handling a request from Siri with one of these classes is to resolve any parameters from the voice command. For example, when starting a workout the user may specify parameters such as the type of workout, a particular goal, and the location of the workout:

- “Start a 30 minute outdoor wall climb with Ascent” specifies the workout goal, location, and name.
- “Start a wall climb workout with Ascent” specifies only the name of the workout.
- “Start a workout with Ascent” specifies no parameters, only that a workout should be started.

An app must decide how to handle these situations using the available resolution methods for the intent type. To handle parameter resolution for `INStartWorkoutIntentHandling`, implement one or more of the `resolveWorkoutName(for:with:)`, `resolveGoalValue(for:with:)`, `resolveWorkoutGoalUnitType(for:with:)`, `resolveWorkoutLocationType(for:with:)`, or `resolveIsOpenEnded(for:with:)` methods.

In each of these calls, return an appropriate resolution result value by calling the `completion` block. For example, when handling the resolution of the workout name, return an `INSpeakableStringResolutionResult` such as `success(with:)`, `confirmationRequired(with:)`, or `disambiguation(with:)`.

```
func resolveWorkoutName(for intent: INStartWorkoutIntent, with completion: @escaping
    let result: INSpeakableStringResolutionResult
    let workoutHistory = WorkoutHistory.load()

    if let name = intent.workoutName {
        // Try to determine the obstacle (wall or boulder) from the supplied workout
        if Workout.Obstacle(intentWorkoutName: name) != nil {
            result = INSpeakableStringResolutionResult.success(with: name)
        } else {
            result = INSpeakableStringResolutionResult.needsValue()
        }
    } else if let lastWorkout = workoutHistory.last {
        // A name hasn't been supplied so suggest the last obstacle.
        result = INSpeakableStringResolutionResult.confirmationRequired(with: lastWo
    } else {
        result = INSpeakableStringResolutionResult.needsValue()
    }
```

```
    completion(result)
}
```

> **Note**
>
> It is not necessary to implement *all* of the available parameter resolution methods, only those that need additional logic. For example, the extension in this sample doesn't need to do any special processing of the goal value when starting a workout, so there is no implementation of `resolveGoalValue(for:with:)`, although the app still recognizes and processes the goal value.

To aid Siri with recognition of parameter names like "wall climb" and "boulder climb," add an App `IntentVocabulary.plist` to a project. More information on this file and how it can add vocabulary to Siri can be found in Registering Custom Vocabulary with SiriKit.

Once all parameters have been resolved, the system calls `confirm(intent:completion:)`. Validate the resolved parameters and pass back an `INStartWorkoutIntentResponse` with an `INStartWorkoutIntentResponseCode`.

Complete the Request

Once the request to start the workout has been confirmed, the system calls `handle(intent:completion:)`. Some requests from Siri will be better handled by the main app rather than the Intent Extension; for example, the main app can better respond to requests in which long-lived system services are needed to support the workout.

To transfer control to the main app, create a `INStartWorkoutIntentResponse` with the `handleInApp` code and pass it to the completion block.

```
func handle(intent: INStartWorkoutIntent, completion: @escaping (INStartWorkoutInter
    // `handleInApp` will transfer handling this activity to the main app and delive
    let response = INStartWorkoutIntentResponse(code: .handleInApp, userActivity: ni
    completion(response)
}
```

When an Intent Extension handles an intent with the `handleInApp` response code, it delivers the to the intent to the `application(_:handle:completionHandler:)` method in the main app's process in the background.

```
func application(_ application: UIApplication, handle intent: INIntent, completionHa
    if let intent = intent as? INStartWorkoutIntent {
```

```
        completionHandler(handle(intent))
    } else if let intent = intent as? INCancelWorkoutIntent {
        completionHandler(handle(intent))
    } else if let intent = intent as? INPauseWorkoutIntent {
        completionHandler(handle(intent))
    } else if let intent = intent as? INEndWorkoutIntent {
        completionHandler(handle(intent))
    } else if let intent = intent as? INResumeWorkoutIntent {
        completionHandler(handle(intent))
    } else {
        preconditionFailure("Trying to handle unknown intent type")
    }
}
```

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.

{} 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: Accelerating App Interactions with Shortcuts

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

{} Soup Chef with App Intents: Migrating custom intents

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