Article

Handling an Intent

Fulfill the intent and provide feedback to SiriKit about what you did.

Overview

After resolving and confirming an intent, SiriKit asks you to handle the intent by fulfilling the user's request. When handling an intent, you do the following:

- Perform the task associated with the intent.
- Return a response object with information about what you did.

You handle most intents directly from your Intents app extension, but in some cases you may ask SiriKit to let your app handle the request.

Example: Handling a Start Workout Intent

A workout app may need to configure timers and other health-related information, tasks that cannot be handled easily from the Intents app extension. Therefore, handling a workout intent involves directing SiriKit to launch your app.

The listing below shows an example of how to handle the start of a workout. This method creates a response object with a code that indicates SiriKit should launch the app.

```
func handle(startWorkout intent: INStartWorkoutIntent,
        completion: @escaping (INStartWorkoutIntentResponse) -> Void) {
    // Let the app start the workout.
    let response = INStartWorkoutIntentResponse(code: .continueInApp,
            userActivity: nil)
    completion(response)
}
```

Example: Handling a Request Ride Intent

A ride booking app needs to provide detailed information about a ride booked by the user. Retrieving ride booking information requires communicating with your service and using the returned information to create an `INRideStatus` object.

The listing below shows an example of how to respond to a ride booking request. Most of the method is dedicated to putting the ride details into an `INRideStatus` object. During that process, the method communicates the ride details to the ride-booking service and gets information about the driver and vehicle providing the ride. Finally, the method returns a response that includes the `INRideStatus` object back to SiriKit.

```
func handle(requestRide intent: INRequestRideIntent,
        completion: @escaping (INRequestRideIntentResponse) -> Void) {
    var rideOption : INRideOption? = nil

    // Save a reference to the ride for status updates
    self.bookedRideIntent = intent

    if let phrase = intent.rideOptionName?.spokenPhrase {
        switch phrase {
        case "SUV":
            rideOption = self.createSUVRideOption(pickup:
                        intent.pickupLocation!,
                        dropOff: intent.dropOffLocation!)
            break
        case "Compact":
            rideOption = self.createCompactRideOption(pickup:
                        intent.pickupLocation!,
                        dropOff: intent.dropOffLocation!)
            break
        case "Sedan":
            rideOption = self.createSedanRideOption(pickup:
                        intent.pickupLocation!,
                        dropOff: intent.dropOffLocation!)
            break

        default:    // Default to a sedan.
            rideOption = self.createSedanRideOption(pickup:
                        intent.pickupLocation!,
                        dropOff: intent.dropOffLocation!)
            break
        }
```

```
    }

    // Create the status for the response.
    let rideStatus = INRideStatus()
    rideStatus.rideOption = rideOption
    let (vehicle, driver) = self.getVehicleAndDriver(intent: intent,
            rideOption: rideOption!)

    // Assign the driver and vehicle.
    rideStatus.vehicle = vehicle
    rideStatus.driver = driver
    rideStatus.estimatedPickupDate =
             self.getPickupTimeForVehicle(vehicle : vehicle)
    rideStatus.pickupLocation = intent.pickupLocation
    rideStatus.dropOffLocation = intent.dropOffLocation

    // Book the ride and get its ID.
    rideStatus.rideIdentifier =
             self.bookRide(rideDetails : rideStatus)

    // Get the current ride phase.
    rideStatus.phase =
             self.phaseForRide(identifier : rideStatus.rideIdentifier)

    // Deliver the response to SiriKit
    if rideStatus.phase == .confirmed {
        let response = INRequestRideIntentResponse(code: .success,
               userActivity: nil)
        response.rideStatus = rideStatus
        completion(response)
    } else {
        let activity = NSUserActivity(activityType:
            "com.example.myRideApp.noRideAvailable")
        let response = INRequestRideIntentResponse(code:
               .failureRequiringAppLaunch, userActivity: activity)
        response.rideStatus = rideStatus
        completion(response)
    }
}
```

Tips for Handling Intents

When handling intents, consider the following:

- **Always include as much information as possible in your responses.** Siri and Maps communicate as much information as possible to the user when confirming or handling intents. Filling your response object with detailed information creates a better user experience by clearly communicating what your app did.
- **Configure data objects fully before assigning them to the response object.** For most response objects, properties use copy semantics. If you get the object in a property and modify it, the changes you make happen on the copy, not the original. Therefore, you must always set the value of the property only after making your changes.
- **Return your response objects within a few seconds.** Because Siri and Maps are actively communicating with the user, always return your response as quickly as possible. If it will take more than a few seconds to return your response, return your response with an "in-progress" code to indicate that you are still working on the request.
- **Use custom user activity objects to support deep linking into your app.** Providing a custom `NSUserActivity` object, instead of relying on the default object, lets you supply additional information for configuring your user interface. You can use that information to display specific information about the task that the user performed in Siri or Maps, which yields a better user experience.
- **Always handle user activity objects in your parent app.** Whenever Siri or Maps need to transfer control to your app, they provide an `NSUserActivity` object with information about what happened. Response objects create a default `NSUserActivity` object if you do not provide one yourself. Handling these objects in your app ensures a seamless experience for the user.

SiriKit leverages information from the user's contacts database to fill in parameters when applicable. If your app was denied access to the user's contacts, information related to those contacts may not be included with the intent. For example, any `INPerson` objects in the intent may contain only a value in their `spokenPhrase` property and not contain any other contact information. This lack of access also prevents SiriKit from accessing the addresses of contacts, which causes a phrase like "`I need a ride home using <app>`" to result in an intent without a drop-off location. If your app was denied access to the user's contacts, you might want to inform the user that you can provide a better experience with access restored.

See Also

Articles

Adding User Interactivity with Siri Shortcuts and the Shortcuts App

Add custom intents and parameters to help users interact more quickly and effectively with Siri and the Shortcuts app.

- Defining Relevant Shortcuts for the Siri Watch Face
  Inform Siri when your app's shortcuts may be useful to the user.
- Deleting Donated Shortcuts
  Remove your donations from Siri.
- Dispatching intents to handlers
  Provide SiriKit with an intent handler capable of handling a specific intent.
- Improving Siri Media Interactions and App Selection
  Fine-tune voice controls and improve Siri Suggestions by sharing app capabilities, customized names, and listening habits with the system.
- Improving interactions between Siri and your messaging app
  Donate app-specific content, use Siri's contact suggestions, and adopt the latest platform features to create a more consistent messaging experience.
- Registering Custom Vocabulary with SiriKit
  Register your app's custom terminology, and provide sample phrases for how to use your app with Siri.
- Confirming the Details of an Intent
  Perform final validation of the intent parameters and verify that your services are ready to fulfill the intent.
- Resolving the Parameters of an Intent
  Validate the parameters of an intent and make sure that you have the information you need to continue.
- Generating a List of Ride Options
  Generate ride options for Maps to display to the user.
- Handling the Ride-Booking Intents
  Support the different intent-handling sequences for booking rides with Shortcuts or Maps.
- Donating Reservations
  Inform Siri of reservations made from your app.
- Specifying Synonyms for Your App Name
  Provide alternative names for your app that are more familiar or easier for users to speak.
- Intent Phrases

The keys that you include in your global vocabulary file to show how users engage your app from Siri.

Localizing Your Vocabulary for Chinese Dialects

Apply emphasis markers to your pronunciation tips to assist Siri with Chinese dialects.