Sample Code

Booking Rides with SiriKit

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

Download

iOS 13.0+ | iPadOS 13.0+ | Xcode 17.0+

Overview

The app in this sample project demonstrates how to use app extensions to give users the ability to book rides through Siri and Maps. App extensions implement the necessary intent-handling protocols for the ride-booking domain.

For information about the general process of adding an Intents extension to your app, including how to enable the Siri capability and configure the NSExtension keys in the `Info.plist` file, see Creating an Intents App Extension.

The `GetRideStatusHandler` class, which conforms to the `INGetRideStatusIntentHandling` protocol, contains comments to help you implement the ridesharing domain for your server. It does not include implementation, but has hints about where to perform certain implementation tasks.

Test the Sample Code Project

You can run this sample app in the iOS simulator without special setup. However, a few steps are required to test the app's functionalities.

To test the app for booking a ride with Siri, launch Siri and say one of the possible commands. Here are a few to get you started:

- "Book a ride"

- “Book a ride to San Francisco”
- “Book a ride from Cupertino to San Francisco”

To test the app for booking a ride with Maps, open the Maps app, enter a destination, and tap Directions. Then go to the Ride tab and request a ride.

Make sure your simulator location is set to “Apple” for the sample to function correctly. Ride booking will only be functional if the pickup and drop-off locations are within the geographic area specified in `CoverageArea.geojson`.

Handle Ride Booking with Siri

To enable Siri to recognize and respond to ride-booking requests, you must implement classes that conform to the `INRequestRideIntentHandling` protocol. Within the protocol, different methods are responsible for different components of the ride-booking process.

The first step is to resolve the parameters necessary for booking a ride; most commonly, you need to resolve pickup and drop-off locations. If the parameters you choose to resolve are not provided in the initial command, they show up as prompts from Siri. For example, if the user says “Book a ride” without including a destination, and you choose to resolve the drop-off location, Siri prompts the user to add a destination.

The pickup and drop-off locations are resolved within the `RideRequestHandler` class.

```
func resolvePickupLocation(for intent: INRequestRideIntent, with completion: @escapi
    // Always check if parameter is not nil
    if let pickup = intent.pickupLocation {
        completion(.success(with: pickup))
    } else {
        completion(.confirmationRequired(with: intent.pickupLocation))
    }
}

func resolveDropOffLocation(for intent: INRequestRideIntent, with completion: @escap
    if let dropOff = intent.dropOffLocation {
        completion(.success(with: dropOff))
    } else {
        completion(.confirmationRequired(with: intent.dropOffLocation))
    }
}
```

After the parameters are resolved, the system calls `confirm(intent:completion:)` to let the user validate their request. You need to create an `INRequestRideIntentResponse` object to

be passed into the completion block. This response contains the information Siri uses to construct the ride confirmation card. Build an `INRideStatus` object to provide additional details about the ride, and attach it to the response.

When you are finished building your `INRequestRideIntentResponse`, pass the response back to the completion handler and call the `completion` block so Siri can access your response.

```
func confirm(intent: INRequestRideIntent, completion: @escaping (INRequestRideIntent
    let rideOption = INRideOption(name: "Small car", estimatedPickupDate: Date(timeI
    let rideStatus = INRideStatus()
    rideStatus.rideOption = rideOption
    rideStatus.estimatedPickupDate = Date(timeIntervalSinceNow: 5 * 60)
    rideStatus.rideIdentifier = NSUUID().uuidString // This ride identifier must mat

    // set pickup and dropoff locations
    let response = INRequestRideIntentResponse(code: .success, userActivity: nil)
    response.rideStatus = rideStatus

    completion(response)
}
```

After you confirm the request, handle it using the `handle(intent:completion:)` method. Include any code needed to connect a driver with the passenger, and include any UI or content changes in the app.

In every handle method, you must construct an `INRequestRideIntentResponse`. You pass this response object into the completion handler, which tells Siri what information to show in the Siri interface when a ride request is initialized. To provide more details, add an `INRideStatus` object to your response as well.

Make sure you call the `completion` block with your response passed in at the end of the `handle` function.

```
func handle(intent: INRequestRideIntent, completion: @escaping (INRequestRideIntentR
    // Build response shown to user after tapping "Request."
    let response = INRequestRideIntentResponse(code: .success, userActivity: .none)

    // Populate status with all information you want to display on screen.
    let status = INRideStatus()
    status.rideIdentifier = NSUUID().uuidString
    // Set current ride status
    status.phase = .confirmed
    status.pickupLocation = intent.pickupLocation
```

```
    status.dropOffLocation = intent.dropOffLocation
    status.completionStatus = .completed()

    // Add driver field
    status.driver = INRideDriver(personHandle: INPersonHandle(value: "email@example.
                                 type: .emailAddress),
                                 nameComponents: PersonNameComponents(),
                                 displayName: "John Doe",
                                 image: INImage(named: "handle_blue"),
                                 contactIdentifier: nil,
                                 customIdentifier: nil,
                                 aliases: nil,
                                 suggestionType: .instantMessageAddress)

    // Add ETA field
    status.estimatedPickupDate = Calendar.current.date(byAdding: DateComponents(minu
    let vehicle = INRideVehicle()
    vehicle.model = "1984 Small Car"
    status.vehicle = vehicle

    response.rideStatus = status
    completion(response)
}
```

Show a List of Ride Options in Maps

By conforming to the `INListRideOptionsIntentHandling` protocol, you can display different ride options inside the Maps app under the Ride tab. For the Maps context, you don't need to implement resolve methods. At minimum, you only need to implement the handle method to tell the context what ride options are available, and how to handle ride selection.

To add ride options to Maps, create an `INRideOption` object and populate it with, at minimum, `name`, `estimatedPickupDate`, `availablePartySizeOptions`, and `priceRange`. Populate the response object's `expirationDate`, `paymentMethods`, and `rideOptions` properties and pass the response object into the completion handler.

```
func handle(intent: INListRideOptionsIntent, completion: @escaping (INListRideOptior
    for(index, name) in rideNames.enumerated() {
        let rideOption = INRideOption(name: name, estimatedPickupDate: Date(timeInte
        // Populate ride options
        if index == 0 {
            rideOption.availablePartySizeOptions = poolOptions
```

```
            rideOption.fareLineItems = fareLineItems
            rideOption.priceRange = INPriceRange(price: NSDecimalNumber(string: "6.0
        }
        ...
        rideOptions.append(rideOption)
    }

    let paymentOptions = [
        INPaymentMethod(type: .credit,
                        name: "Card 1",
                        identificationHint: "••••1111",
                        icon: INImage(named: "")),
        ...
    ]

    let response = INListRideOptionsIntentResponse(code: .success, userActivity: nil
    response.expirationDate = Date(timeIntervalSinceNow: 60 * 5)
    response.paymentMethods = paymentOptions
    response.rideOptions = rideOptions

    completion(response)
}
```

To ensure that the ride options show up in Maps, add the `INGetRideStatusIntent`, `INListRideOptionsIntent`, and `INRequestRideIntent` keys to your extension's `Info.plist` file under IntentsSupported.

Customize the Ride Request UI in Siri

You can create a custom UI for Siri or Maps to display when a user requests a ride.

Use Interface Builder to modify the `MainInterface` storyboard in the RideExtensionUI folder. After building your UI, conform to the `INUIHostedViewControlling` protocol in `IntentViewController.swift`. To do so, implement the `configure(with:context:completion:)` method by calling the completion block and passing in a `CGSize` object that fits your custom view.

See Also

Sample code

{} Adding Shortcuts for Wind Down

Reveal your app's shortcuts inside the Health app.

{} 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: 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.