Article

Providing Live Status Updates

Provide regular updates to Maps about the status of a booked ride.

Overview

After the user books a ride, Maps may call the `startSendingUpdates(for:to:)` method of your `INGetRideStatusIntentHandling` handler object to ask for live status updates. Maps calls this method only once. Your implementation must set up a timer or other repeating task to generate status updates for the ride and deliver them to the provided observer object. Maps uses your updates to keep the user informed about the ride's location and estimated arrival time.

The listing below shows a skeletal implementation of the `startSendingUpdates(for:to:)` method that sets up a timer for generating status updates. Because the system can call this method may on any thread, the implementation sets up the timer on the main thread, which ensures that the timer continues to fire.

```
public func startSendingUpdates(forGetRideStatus intent: INGetRideStatusIntent,
            to observer: INGetRideStatusIntentResponseObserver) {
   // Save a reference to the observer object
   self.observer = observer
   // Set up the timer on the main thread and save a reference to it.
   DispatchQueue.main.async {
      self.timer = Timer.scheduledTimer(withTimeInterval: 5,
                     repeats: true, block: { (timer) in
         let rideStatus = INRideStatus()
         // FOR YOU TO DO: Configure the INRideStatus object.

         // Create the response and deliver it.
         let response = INGetRideStatusIntentResponse(code: .success, userActivity:
         response.rideStatus = rideStatus
         self.observer?.didUpdate(getRideStatus: response)
```

```
        })
    }
}
```

When crafting your response, always include a valid `INRideStatus` object. Here are some tips for configuring the status object to ensure that SiriKit processes your updates successfully:

- **Always specify a valid ride phase and completion status.** Maps uses the `phase` property of the `INRideStatus` object to communicate the ride status to the user. Specifically, when the phase changes, Maps updates that information in its interface. Never set the ride phase to `INRidePhase.unknown`.
- **Always provide a completion status for a completed ride.** When setting the ride phase to `INRidePhase.completed`, assign an appropriate value to the `completionStatus` property of the `INRideStatus` object to indicate how to resolve the ride.
- **Always use the same ride identifier that you used during booking.** Updates must include the same value in the `rideIdentifier` property of the `INRideStatus` object that you used when booking the ride. If the identifiers do not match, SiriKit ignores any further updates.
- **Provide a valid ride option object.** The value in the `rideOption` property of the `INRideStatus` object need not match the one you specified during booking, but the ride option name and estimated pickup date must be valid.
- **Provide a map annotation image for your vehicle.** Maps uses the map annotation image from your status object to show the position of your vehicle on the map surface. Specify this image using the `mapAnnotationImage` property of the `INRideVehicle` object you assign to your status object.

Upon the completion of a ride, call your handler object's `stopSendingUpdates(for:)` method when you want to stop the delivery of live updates. You implement this method and use it to cancel the recurring task that you use to generates updates. You can stop updates at any time, but typically you do so after the ride is complete and have settled any charges.

The listing below shows an implementation of the `stopSendingUpdates(for:)` method that invalidates the timer used to deliver live updates and removes references to the observer object. Because both your code and SiriKit can call this method, it is an ideal place to perform all cleanup associated with live updates.

```
public func stopSendingUpdates(forGetRideStatus intent: INGetRideStatusIntent) {
    self.observer = nil
    self.timer?.invalidate()
     // Clean up any other state information.
}
```

The code you use to generate live updates must be fault-tolerant and able to handle errors well. Because getting live updates involves communicating with your server, you should handle cases where your server does not respond or returns an error. For example, you might return a cached response if your server does not respond in a timely manner. If you deliver an invalid response object to the observer, Maps calls the `stopSendingUpdates(for:)` of your handler object and notifies the user that a problem occurred.

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.

- Handling an Intent

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

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.