Sample Code

Handling Payment Requests with SiriKit

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

Download

iOS 10.0+ | iPadOS 10.0+ | Xcode 17.0+

Overview

The app in this sample project demonstrates how to send payments between users by implementing `INSendPaymentIntentHandling`. Users can make payments with Siri and view a list of previously completed payments inside the main app.

The project consists of three targets:

- `Payments`, an iOS app which shows a history of completed payments.
- `PaymentsIntentsExtension`, an Intent Extension that integrates with SiriKit to send payments.
- `PaymentsFramework`, 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 the `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 on a device you will need to update the build settings and enable an App Group for the project:

1. Open `Payments.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 `Payments` and `PaymentsIntentsExtension` 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 `PaymentProvider.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 `Payments` target to `com.example.Payments`, then you should set the bundle identifier for the `PaymentsIntentsExtension` target to `com.example.Payments.PaymentsIntentsExtension`.

Resolve Payment Parameters

The first step in processing a payment with `INSendPaymentIntentHandling` is to obtain the payee for the transaction and the currency amount to send to the payee. Depending on the command that the user speaks to Siri, your app may have to deal with one of the following scenarios:

- The initial request specifies all required parameters. For example, the user requests, "Send $50 to Jane with Payments."
- The request lacks some required parameters. For example, the user requests, "Send money to Jane with Payments."
- The request supplies incorrect or inaccurate parameters. For example, the user requests, "Send money to Dave with Payments," when no valid contacts match "Dave".
- The initial request supplies no parameters. For example, the user requests, "Send money with Payments."

The extension resolves each parameter separately by responding to methods from the `INSendPaymentIntentHandling` protocol. Depending on the initial request and subsequent conversation with Siri, each of the parameter resolution methods may be called multiple times as Siri clarifies the request from the user.

The extension resolves the payee parameter through calls to `resolvePayee(for:with:)`. Use this method to take input from Siri, match it to a payee, and asynchronously return an `INPerson`

ResolutionResult by calling success(with:), confirmationRequired(with:), or disambiguation(with:).

```
func resolvePayee(for intent: INSendPaymentIntent, with completion: @escaping (INPer
    if let payee = intent.payee {
        // Look up contacts that match the payee.
        contactLookup.lookup(displayName: payee.displayName) { contacts in
            // Build the `INIntentResolutionResult` to pass to the `completion` clos
            let result: INPersonResolutionResult

            if let contact = contacts.first, contacts.count == 1 {
                // An exact single match.
                let resolvedPayee = INPerson(contact: contact)
                result = INPersonResolutionResult.success(with: resolvedPayee)
            } else if contacts.isEmpty {
                // Found no matches.
                result = INPersonResolutionResult.unsupported()
            } else {
                // Found more than one match; user needs to clarify the intended cor
                let people: [INPerson] = contacts.map { contact in
                    return INPerson(contact: contact)
                }
                result = INPersonResolutionResult.disambiguation(with: people)
            }
            completion(result)
        }
    } else if let mostRecentPayee = paymentProvider.mostRecentPayment?.contact {
        // No payee provided; suggest the last payee.
        let result = INPersonResolutionResult.confirmationRequired(with: INPerson(co
        completion(result)
    } else {
        // No payee provided and there was no previous payee.
        let result = INPersonResolutionResult.needsValue()
        completion(result)
    }
}
```

The extension resolves the currency amount for the payment through calls to resolveCurrency Amount(for:with:), and optionally a note to be associated with the transaction is resolved with calls to resolveNote(for:with:).

```
func resolveCurrencyAmount(for intent: INSendPaymentIntent, with completion: @escapi
    let result: INCurrencyAmountResolutionResult

    // Resolve the currency amount.
    if let currencyAmount = intent.currencyAmount, let amount = currencyAmount.amour
        if amount.intValue <= 0 {
            // The amount needs to be a positive value.
            result = INCurrencyAmountResolutionResult.unsupported()
        } else if let currencyCode = paymentProvider.validate(currencyCode) {
            // Make a new `INCurrencyAmount` with the resolved currency code.
            let resolvedAmount = INCurrencyAmount(amount: amount, currencyCode: curr
            result = INCurrencyAmountResolutionResult.success(with: resolvedAmount)
        } else {
            // Unsupported currency.
            result = INCurrencyAmountResolutionResult.unsupported()
        }
    } else if let mostRecentPayment = paymentProvider.mostRecentPayment {
        // No amount provided; suggest the last amount sent.
        let suggestedAmount = INCurrencyAmount(amount: NSDecimalNumber(decimal: most
                                               currencyCode: mostRecentPayment.curre
        result = INCurrencyAmountResolutionResult.confirmationRequired(with: suggest
    } else {
        // No amount provided and there was no previous payment.
        result = INCurrencyAmountResolutionResult.needsValue()
    }
    completion(result)
}
```

Add Domain-Specific Language

Use `INVocabulary` to aid Siri with recognizing any domain-specific vocabulary users are likely to use in their voice commands. For example, in this sample project payee names are not taken from the user's contacts but from a predefined list inside the app, so these names are added as additional vocabulary with `setVocabularyStrings:ofType:`.

```
// Register names of contacts that may not be in the user's address book.
let contactNames = Contact.sampleContacts.map { $0.formattedName }
INVocabulary.shared().setVocabularyStrings(NSOrderedSet(array: contactNames), of: .c
```

More information on the types of custom vocabulary that are appropriate to add to Siri can be found in Registering Custom Vocabulary with SiriKit.

Confirm and Complete the Payment

When all parameters have been resolved, the system calls `confirm(intent:completion:)` to let the app validate the transaction details. Pass back the status of the transaction by calling the `completion` block with an `INSendPaymentIntentResponse` object initialized from a `INSendPaymentIntentResponseCode` and an `INPaymentRecord` in the `paymentRecord`.

```
func confirm(intent: INSendPaymentIntent, completion: @escaping (INSendPaymentIntent
    guard
        let payee = intent.payee,
        let payeeHandle = payee.personHandle,
        let currencyAmount = intent.currencyAmount,
        let amount = currencyAmount.amount,
        let currencyCode = currencyAmount.currencyCode
        else { completion(INSendPaymentIntentResponse(code: .failure, userActivity:

    contactLookup.lookup(emailAddress: payeeHandle.value!) { contact in
        guard let contact = contact else {
            completion(INSendPaymentIntentResponse(code: .failure, userActivity: nil
            return
        }

        let payment = Payment(contact: contact, amount: amount.decimalValue, currenc

        self.paymentProvider.canSend(payment) { success, error in
            guard success else {
                completion(INSendPaymentIntentResponse(code: .failure, userActivity:
                return
            }

            let response = INSendPaymentIntentResponse(code: .success, userActivity:
            response.paymentRecord = self.makePaymentRecord(for: intent)

            completion(response)
        }
    }
}
```

Finally, once the payment has been confirmed by the user, the system calls `handle(intent:completion:)`. Use this method to perform the confirmed transaction and pass back the status of the transaction by calling the `completion` block.

```
func handle(intent: INSendPaymentIntent, completion: @escaping (INSendPaymentIntentR
    guard
        let payee = intent.payee,
        let payeeHandle = payee.personHandle,
        let currencyAmount = intent.currencyAmount,
        let amount = currencyAmount.amount,
        let currencyCode = currencyAmount.currencyCode
        else { completion(INSendPaymentIntentResponse(code: .failure, userActivity:

    contactLookup.lookup(emailAddress: payeeHandle.value!) { contact in
        guard let contact = contact else {
            completion(INSendPaymentIntentResponse(code: .failure, userActivity: nil
            return
        }

        let payment = Payment(contact: contact, amount: amount.decimalValue, currenc

        self.paymentProvider.send(payment) { success, _, _ in
            guard success else {
                completion(INSendPaymentIntentResponse(code: .failure, userActivity:
                return
            }

            let response = INSendPaymentIntentResponse(code: .success, userActivity:
            response.paymentRecord = self.makePaymentRecord(for: intent)

            completion(response)
        }
    }
}
```

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 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.