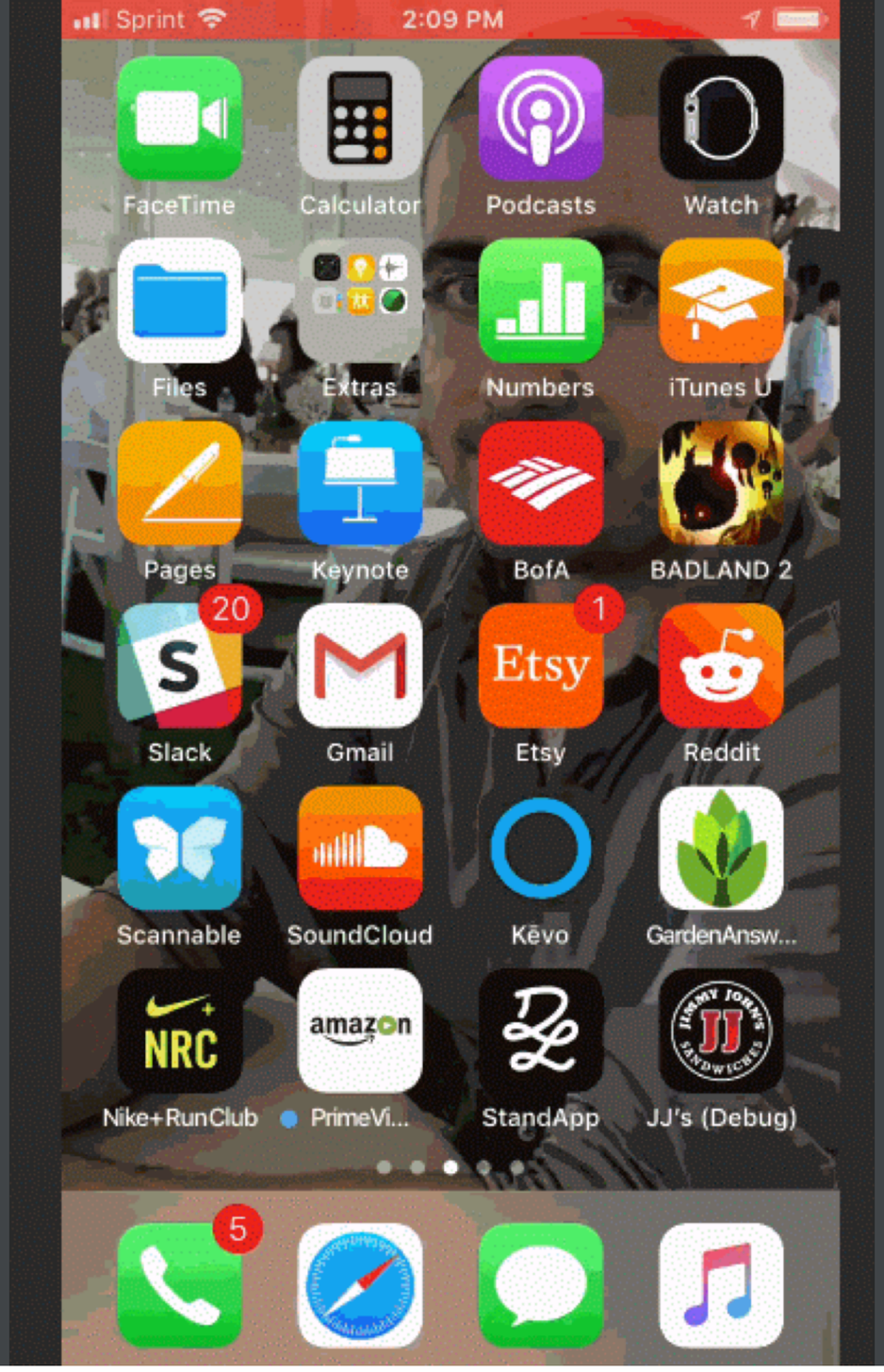
Paying Bills With SiriKit

by Elizabeth Levosinski

Today I wanted to talk about adding a SiriKit extension to an existing app
^ This SiriKit extension allows the user to use Siri voice commands to pay bills
^ For this extension, this is only possible if the user is logged in on the existing app and if the user has set up an automatic payment method
^ Keeping this in mind,
^ I'm going to go over the overall functionality of SiriKit
^ as well as the pitfalls I encountered during my first attempt at building this extension
^ I'm not a SiriKit expert but I wanted to share what I gleaned from my first experience working with it
^ I'll answer all questions at the end
^ Demo



What are Intents?

- SiriKit defines Intents as types of requests that users can make
- Intents that are related are grouped into ‘domains’

Before I explain what SiriKit is though I need to explain what Intents are:

Users could make requests about booking rides, paying bills, transferring money and a few other things.

^ I’ll be focusing on the INPayBillIntent which is used when users want to pay a bill. It’s grouped in the ‘Payments’ domain

INPayBillIntent

- The transaction amount
- The transaction scheduled date
- When the bill is due, bill total, minDue, paymentDate, late fee?
- The “from account” info (User’s credit card or bank account number)
- The “bill payee” (the company the user has an account with that they are paying off)

For instance, if the user had a water bill, the bill payee is defined by the user’s account with the water company (account number, organization name, and account nickname), this comes up again later

What is SiriKit?

SiriKit includes both the Intents and Intents UI App Extensions which allow you to integrate your services with Siri and Maps

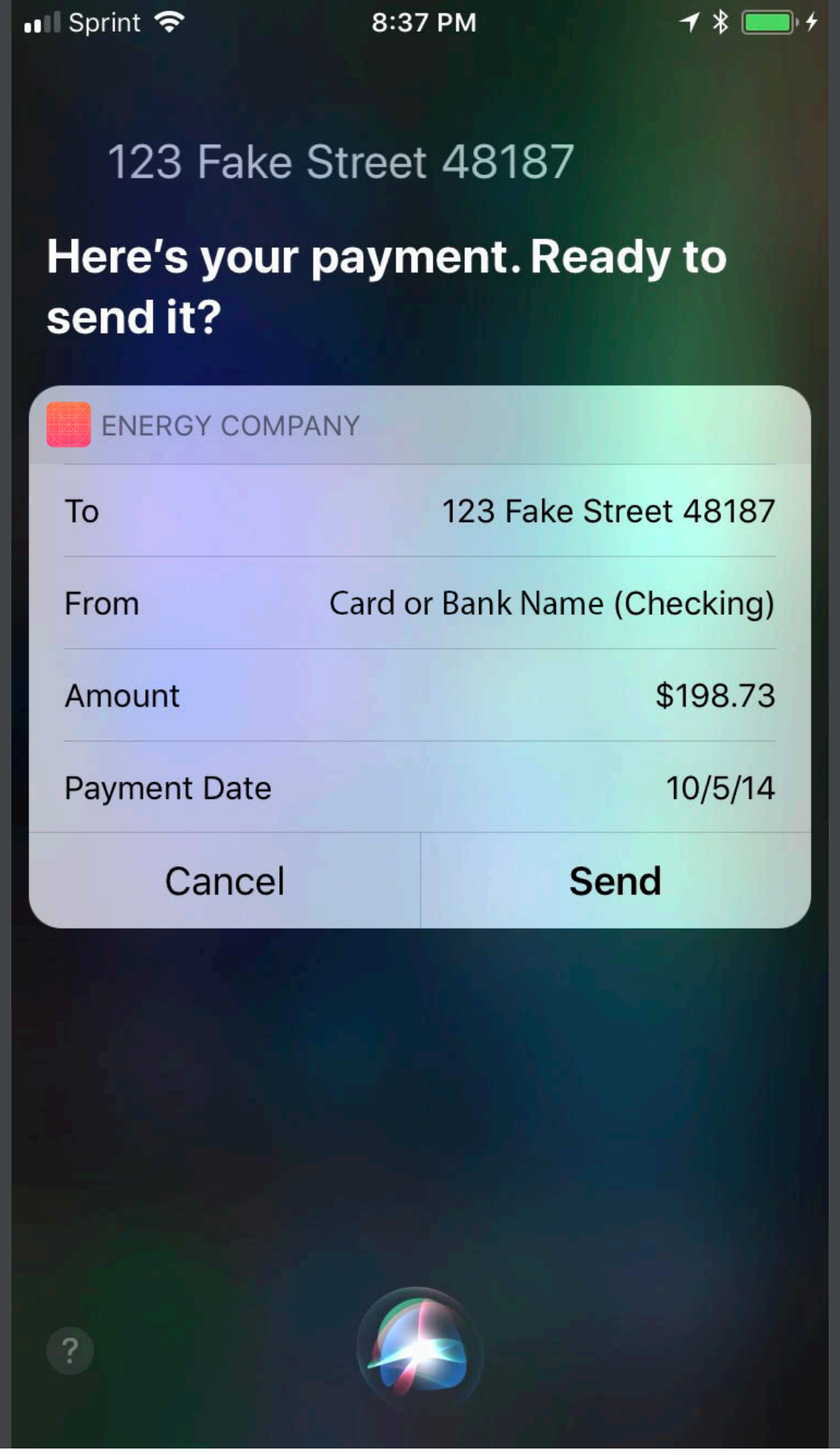
What is an ‘Intents App Extension’?

An Intents app extension receives user requests from SiriKit and turns them into app-specific actions

For example, your user might ask Siri to send a message, book a ride, or pay a bill using your app

What is the ‘Intents UI App Extension’?

An Intents UI app extension is optional and is used to style and customize the content that’s displayed in the Siri or Maps interface after the ‘Intents App Extension’ fulfills a user’s request.



The summary of the steps required to add SiriKit to your app:

- Add an Intents extension
- Declare the supported intents in the Info.plist included in the extension
- Add code to request the user's permission to use Siri
- Add a Siri usage description to the app's Info.plist
- Add the Siri entitlement by flipping the Siri capabilities switch

www.raywenderlich.com/155732/sirikit-tutorial-ios

I'm not going to all the code/ steps to set up sirikit but all the steps you need to get going are listed in great detail in the tutorial link provided at the bottom.

After following those steps and creating the extension, a new folder is generated in your project. It'll contain two files:
 - IntentHandler.swift
 - Info.plist

The IntentHandler.swift contains some default code for a different intent we're not going to use. I replaced that template code with code that sets up the INPayBillIntent:

Providing Handler Object to SiriKit

```swift
// IntentHandler.swift

import Intents

class IntentHandler: INExtension {

  let payBillRequestHandler = PayBillRequestHandler()

  override func handler(for intent: INIntent) -> Any? {
    if intent is INPayBillIntent {
      return payBillRequestHandler
    }
    return .none
  }
}
```


I created a class called “PayBillRequestHandler” which holds all the logic for my SiriKit Extension.

^ It has the INPayBillIntentHandling protocol which gives it access to all of the SiriKit functionality

SiriKit Phases

In my ‘PayBillRequestHandler’ class my code goes through the three SiriKit phases:
- Resolution phase
- Confirmation phase
- Handling the Intent

The parameters of the intent object are validated to make sure you have the info you need to continue
^ Confirming the details on an intent (Perform the final validation of the intent parameters/verify your services are ready)
^ Fulfill the intent and provide feedback to SiriKit about what you did (This is where you do something with all the data you collected for your specific intent, in our case it's posting a payment for our app)

Resoultion Phase:

Names of all the INPayBillIntent resolution functions:

- resolveBillPayee
- resolveFromAccount
- resolveTransactionAmount
- resolveDueDate
- resolveTransactionScheduledDate
- resolveTransactionNote

These are all the possible resolution functions that could be used to prompt the user to populate the INPayBillIntentResponse

^ None of these are required and in fact the less of these you use the better the user experience because the more information you can programmatically populate, the less you will need Siri to ask the user for more info.

All of the resolve functions are used to populate this INPayBillIntentResponse object:

```
func createResponse(with code: INPayBillIntentResponseCode) -> INPayBillIntentResponse {
    let response = INPayBillIntentResponse(code: code, userActivity: nil)
    response.transactionAmount = self.transactionAmount
    response.transactionScheduledDate = self.transactionScheduledDate
    response.billDetails = self.billDetails
    response.fromAccount = self.fromAccount
    response.transactionNote = self.transactionNote
    return response
}
```


Pitfall

^ If any of the information is missing SiriKit won't move on to the confirmation stage, it will just fail

^ Also, ideally should be able to populate all of these properties programmatically just by using the same posts or requests that your main application uses (for instance if a user set up an automatic payment method, we would just make those same calls to populate the INPayBillIntent as much as possible)

```
func finishBillDetails(account: Account, billPayee: INBillPayee) {
   let charges = account.summaryOfCharges // need to pick correct charges from chosen account

   guard let billTotal = charges.billTotal,
     let billDueDate = charges.billDueDate else { return }

   // Company Account Info
   let lateFee = INCurrencyAmount(amount: 0, currencyCode: "USD") // is there a late fee?
   let amountDue = INCurrencyAmount(amount: NSDecimalNumber(value: billTotal), currencyCode: "USD")
   let minDue = INCurrencyAmount(amount: NSDecimalNumber(value: billTotal), currencyCode: "USD")
   self.transactionAmount = INPaymentAmount(amountType: .amountDue, amount: amountDue) // actual amount paid

   let chargesDueDate = self.convertStringToDate(dateString: billDueDate)
   let dueDate: DateComponents = self.convertDate(curDate: chargesDueDate)
   // it doesn't seem to notice the start and end dates have already passed... the data is from 2014
   guard let startDate = charges.startDate,
     let endDate = charges.endDate else { return }

   let start = self.convertStringToDate(dateString: startDate)
   let end = self.convertStringToDate(dateString: endDate)
   let currentDate = Date() // im manually setting the payment date to be on the current day

    self.billDetails = INBillDetails(billType: .electricity, paymentStatus: .unpaid, billPayee: billPayee,
amountDue: amountDue, minimumDue: minDue, lateFee: lateFee, dueDate: dueDate, paymentDate: self.convertDate(curDate: currentDate))

    self.transactionScheduledDate = self.getDateRange(startDate: start, endDate: end)
  }
```


Resolve BillPayee Resolution Function:

```
func resolveBillPayee(for intent: INPayBillIntent,
     with completion: @escaping (INBillPayeeResolutionResult) -> Void) {
  // enter code here..
}
```


I need to use this function because for my app it was possible that a user could have multiple accounts (addresses) and I needed to know which address they were paying for.

resolveBillPayee part 1

```swift
func resolveBillPayee(for intent: INPayBillIntent, with completion: @escaping (INBillPayeeResolutionResult) -> Void) {

  if (!userIsLoggedIn()) { // if the user is not logged in it will fail
    let emptyPayee = INBillPayee(nickname: INSpeakableString(spokenPhrase: ""), number: nil, organizationName: nil)!
    let result = INBillPayeeResolutionResult.success(with: emptyPayee)
    completion(result)
    return
  }

  if (self.billPayeesArray.count > 0) {
    if let selectedPayee = intent.billPayee {
      let result = INBillPayeeResolutionResult.success(with: selectedPayee)
      completion(result)
    }
  }
...
```


The first part of this function checks to see if the user is logged in (you don’t have to do this I just did this because it was a requirement for the app.

^ Whether or not the user is logged in, you must pass SiriKit a payee, there is no failure function, so if they're not logged in pass an empty payee which will fail in the next step

^ This app uses an App Group to tell if the user is logged in or not, I had to do that because the app and the extension are separate and to share a logged in state I needed something that would allow the app and the extension share data between themselves (something I had learned about extensions in general)

^ This function gets called a few times

^ The first time the class property ‘billPayeesArray’ is empty

resolveBillPayee part 2

```
else {
     GetUserAccountAPI.getUserAccountData() { (customerData) in
       guard let customerObj = customerData else {
         return // failure doesn't seem to be a 'result' option
       }

       self.accounts = customerObj.customers[0].accounts

       guard let paymentAccounts = self.accounts else {
         return
       }

       guard let finalResult = self.createBillPayee(paymentAccounts: paymentAccounts) else {
         return
       }

       completion(finalResult) // send data back to SiriKit
     }
   }
 }
```


so I made an api request to get the user’s information
^ Once I have the user’s data for their account info...

createBillPayee Part 1

```
func createBillPayee(paymentAccounts: [Account]) -> INBillPayeeResolutionResult? {
  var result: INBillPayeeResolutionResult?
  var accountNum: String?
  let name = INSpeakableString(spokenPhrase: "Energy Company")
  var accountNickName: INSpeakableString?

  if (paymentAccounts.count > 1) {
    for account in paymentAccounts {
      accountNum = account.accountNumber
      accountNickName = INSpeakableString(spokenPhrase: account.mailingAddress.address + " " + account.mailingAddress.postalCode)
      if let billPayee = INBillPayee(nickname: accountNickName!, number: accountNum, organizationName: name) {
        self.billPayeesArray.append(billPayee)
      }
    }

    result = INBillPayeeResolutionResult.disambiguation(with: self.billPayeesArray)
  }
  . . .
```


I then call my custom function ‘createBillPayee’ to see if they have more than one account

^ I loop through the account information in the data and append each one to the ‘billPayeesArray’

^ if they have more than one account I then call the ‘INBillPayeeResolutionResult.disambiguation’ function and pass in the array of ‘INBillPayee’s I generated

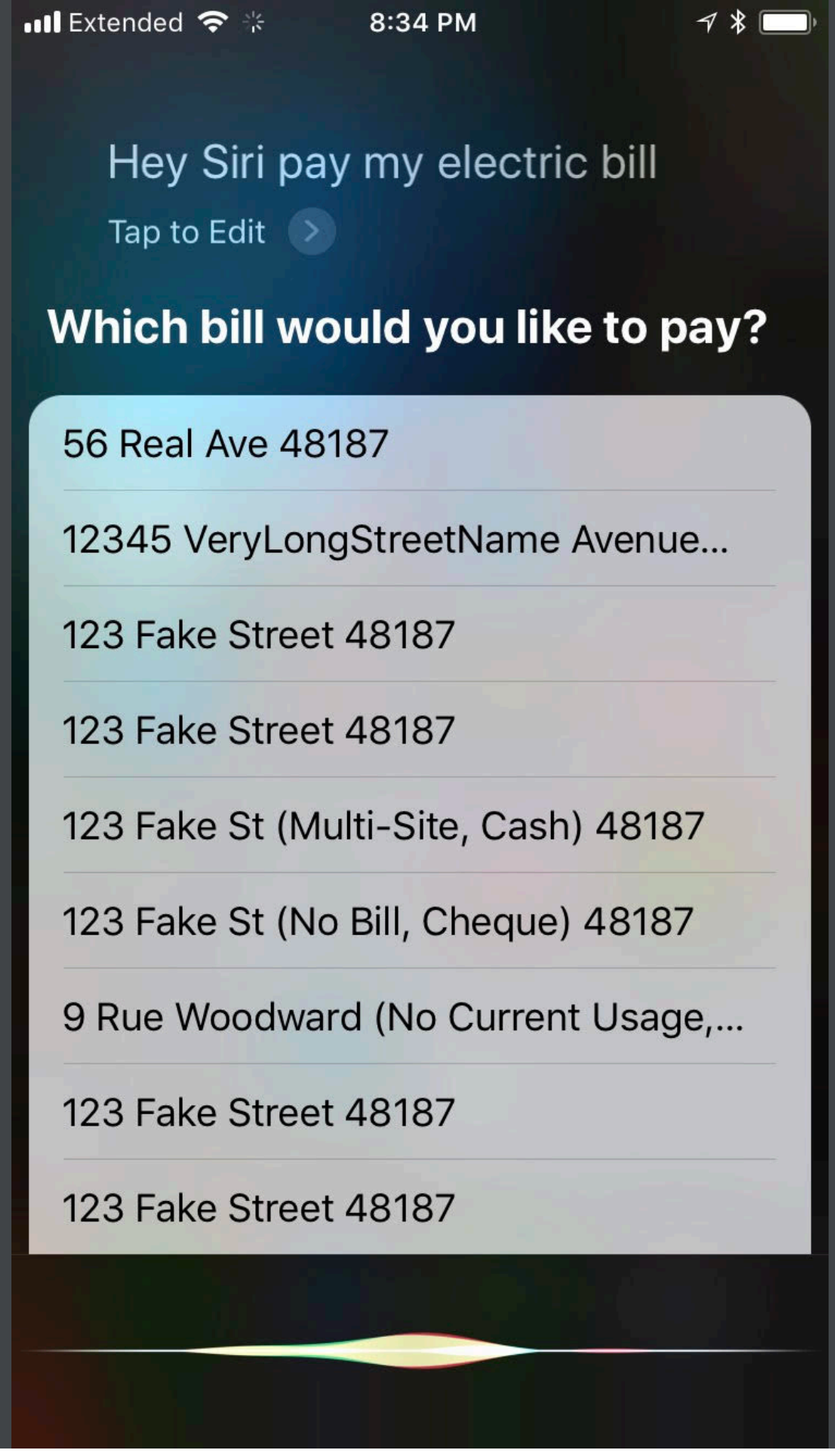
createBillPayee Part 2

```
else {
    accountNickName = INSpeakableString(spokenPhrase: "Energy Company Payment Account")
    accountNum = paymentAccounts[0].accountNumber
    if let billPayee = INBillPayee(nickname: accountNickName!, number: accountNum, organizationName: name) {
      result = INBillPayeeResolutionResult.success(with: billPayee)
    }
  }

  return result
}
```


However if there is only one payment account, the billPayee is resolved and the user will never see the next slide with all the addresses

```
INBillPayeeResolutionResult.disambiguation(with: self.billPayeesArray)
```



If they have only one account (one address) the disambiguation function is never called and they are never prompted for their address

```swift
func confirm(intent: INPayBillIntent, completion: @escaping (INPayBillIntentResponse) -> Void) {
  if (!userIsLoggedIn()) { // if the user is not logged in it will fail
    let failedResponseLoggedOut = INPayBillIntentResponse(code: .failureCredentialsUnverified, userActivity: nil)
    completion(failedResponseLoggedOut)
    return
  }

  guard let currentBillPayee = intent.billPayee else {
    let response = INPayBillIntentResponse(code: .failure, userActivity: nil)
    completion(response)
    return
  }

  guard let accounts = self.accounts else {
    let response = INPayBillIntentResponse(code: .failure, userActivity: nil)
    completion(response)
    return
  }

  for account in accounts {
    if (account.accountNumber == currentBillPayee.accountNumber) {
      finishBillDetails(account: account, billPayee: currentBillPayee)
    }
  }

  createPayBillIntentResponse(codeType: .ready) { (response, _) in
    completion(response)
  }
}
```

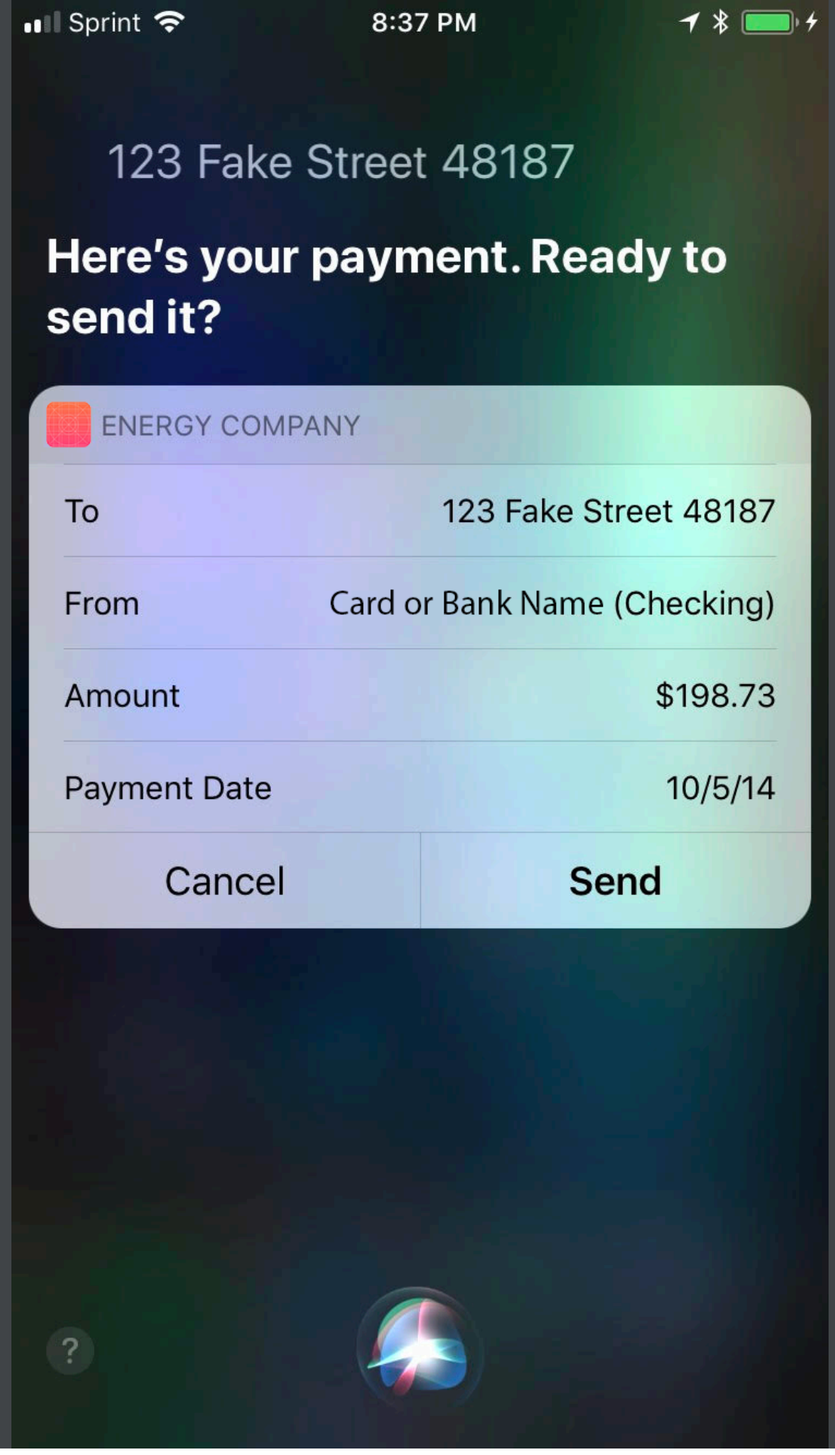

Confirming an Intent

^ This step just ensures that the INPayBillIntentResponse that we are sending to SiriKit has all of its properties filled

^ Why is the userActivity set to nil?

^ It looks like you can use 'userActivity' to pass information to your App Delegate from your SiriKit extension, but it isn't necessary for what we're doing...

Confirming an Intent



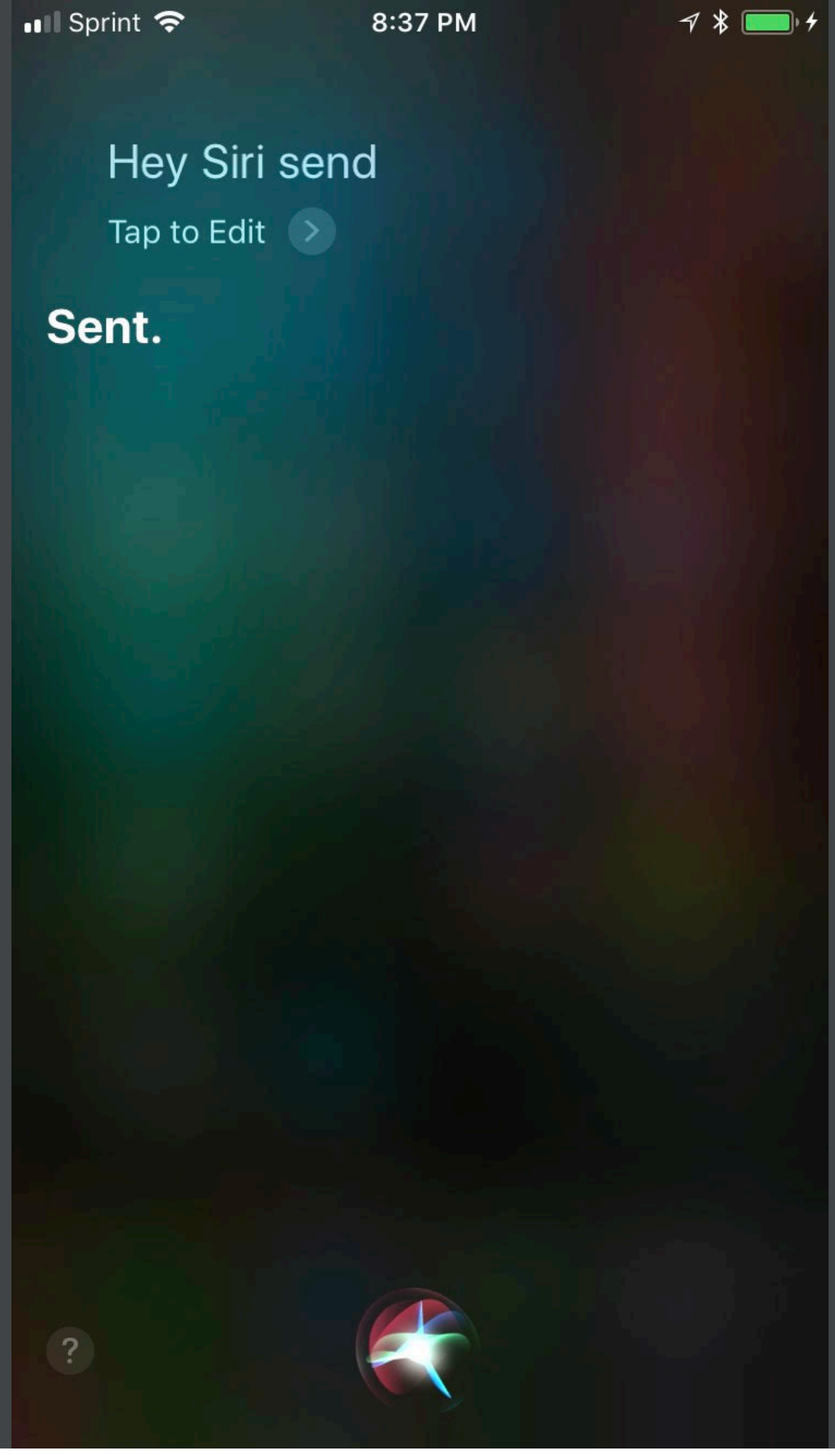
Handling an Intent

```
func handle(intent: INPayBillIntent, completion: @escaping (INPayBillIntentResponse) -> Void) {
  createPayBillIntentResponse(codeType: .success) { (response, savedPaymentMethod) in
    guard let savedPaymentMethod = self.savedPaymentMethod else {
      completion(response)
      return
    }

    PostPaymentAPI.postPayment(savedPaymentResponse: savedPaymentMethod, successfulResponse: response, completion: completion)
  }
}
```


This is just sending the payment, so put in whatever post payment code you have here.

Handling an Intent



Resources

- Apple Documentation: https://developer.apple.com/documentation/sirikit
- Raywenderlich Tutorial for getting started: https://www.raywenderlich.com/155732/sirikit-tutorial-ios
- For userActivity example: http://agostini.tech/2017/05/22/using-sirikit/

