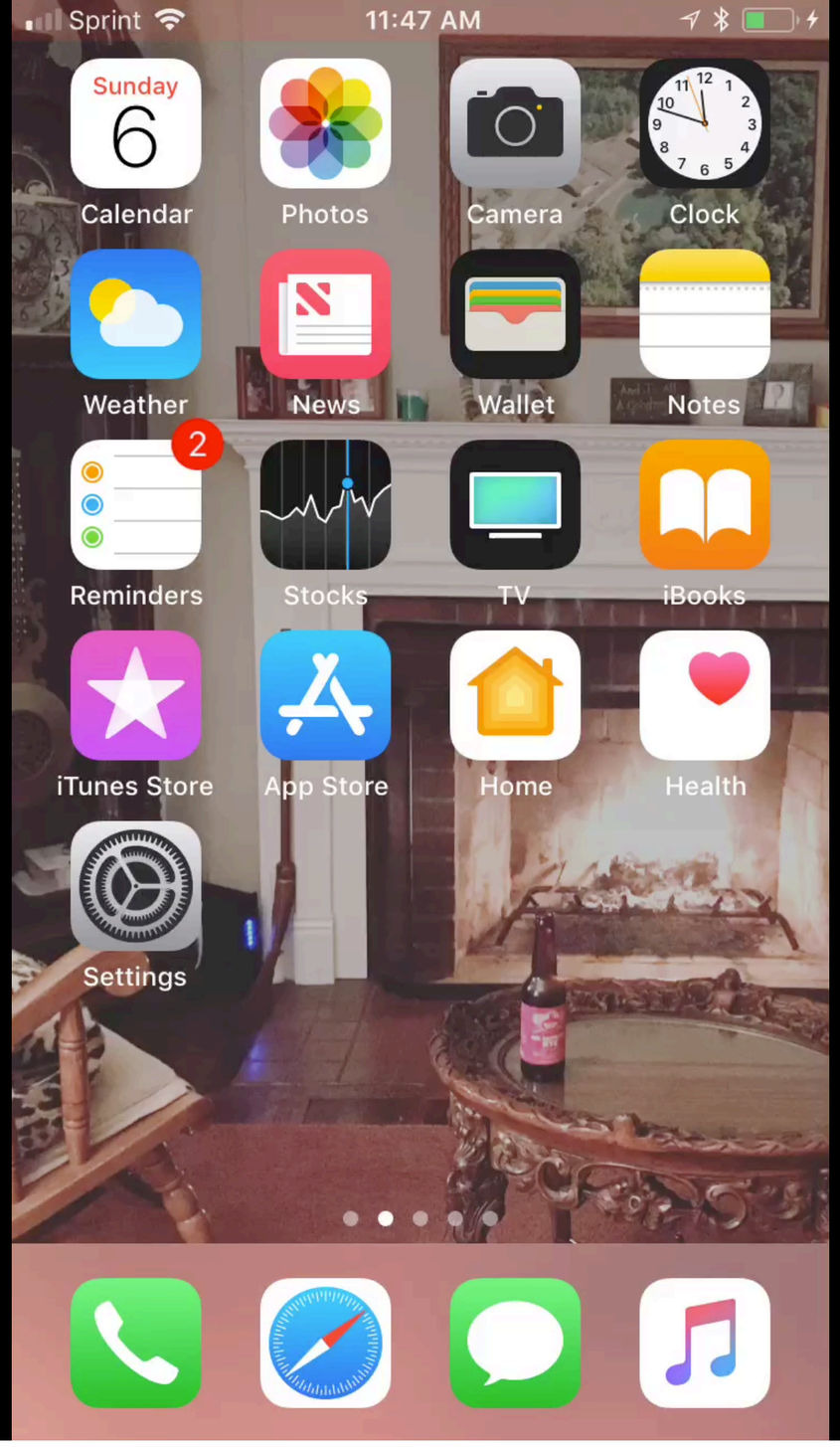
Paying Bills With SiriKit

by Liz Levosinski

- **Download the project here:** https://github.com/levosins/sirikit
- **LinkedIn:** www.linkedin.com/in/elizabethlevosinski
- **Email:** levosins@gmail.com

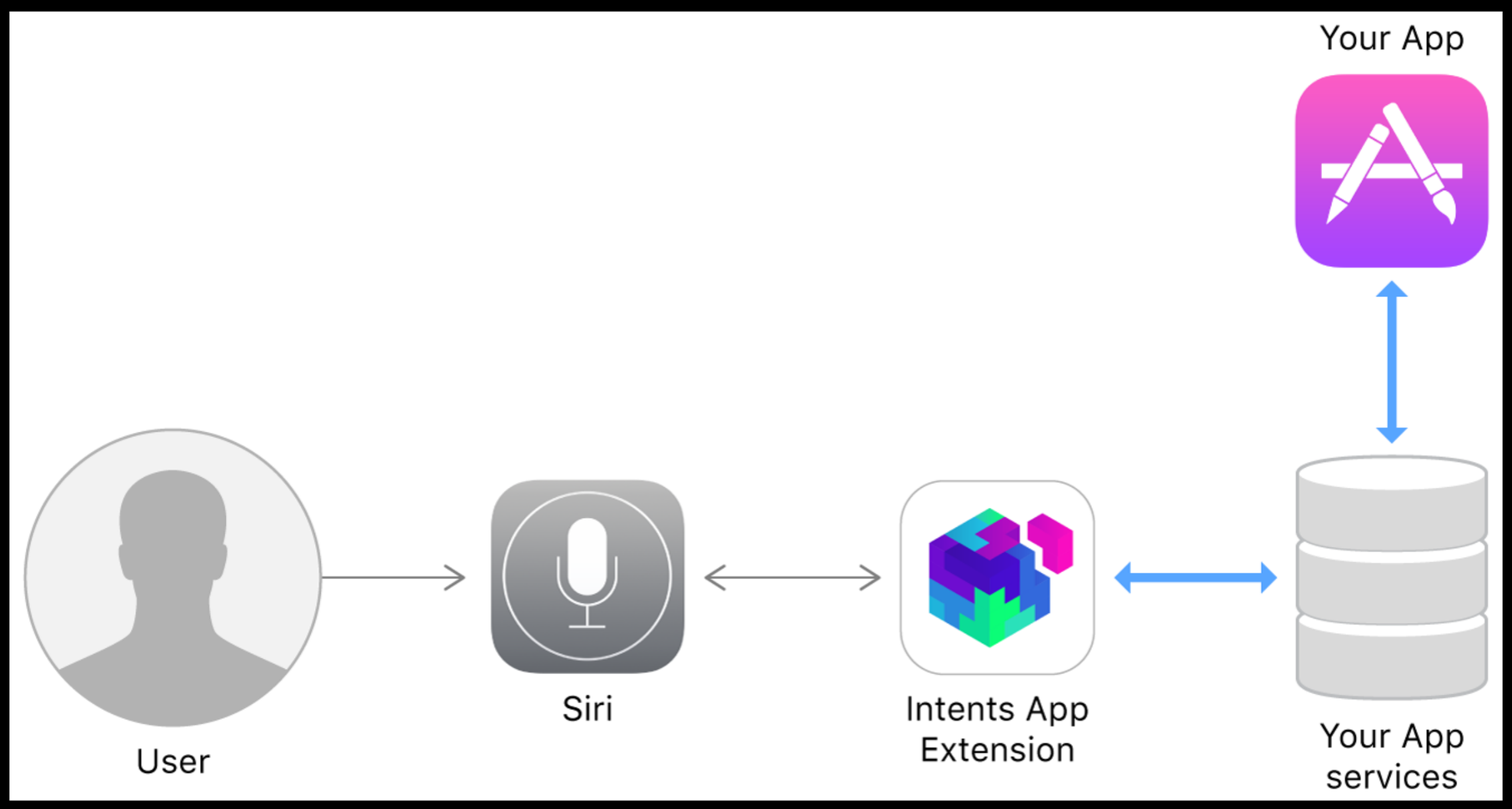
- Video example of the SiriKit Extension
- SiriKit
- Intents (INPayBillIntent)
- How to add SiriKit & App Groups to your app
- Example App Code
- SiriKit custom vocabulary
- Testing
- Security
- SiriKit on HomePod
- Resources

Contents of my talk



Launch the app and give the app authorization to use Siri's services
^ "Sign in" on the app otherwise the extension can't access their account information
^ Background the app
^ Say "Hey Siri" so that Siri is ready to listen to any commands
^ Say "Pay My Bill"
^ Siri will then look in you device for a compatible app that uses the bill pay Intent
^ The app then will make an api call to get the user's data
^ If the user has more than one address that they are paying bills for
^ They are then prompted to pick a the correct address to pay bills for
^ Once the user has chosen an address the app displays a modal with all the payment details
^ Then the user can say or press "Send" to send the payment

What is SiriKit?



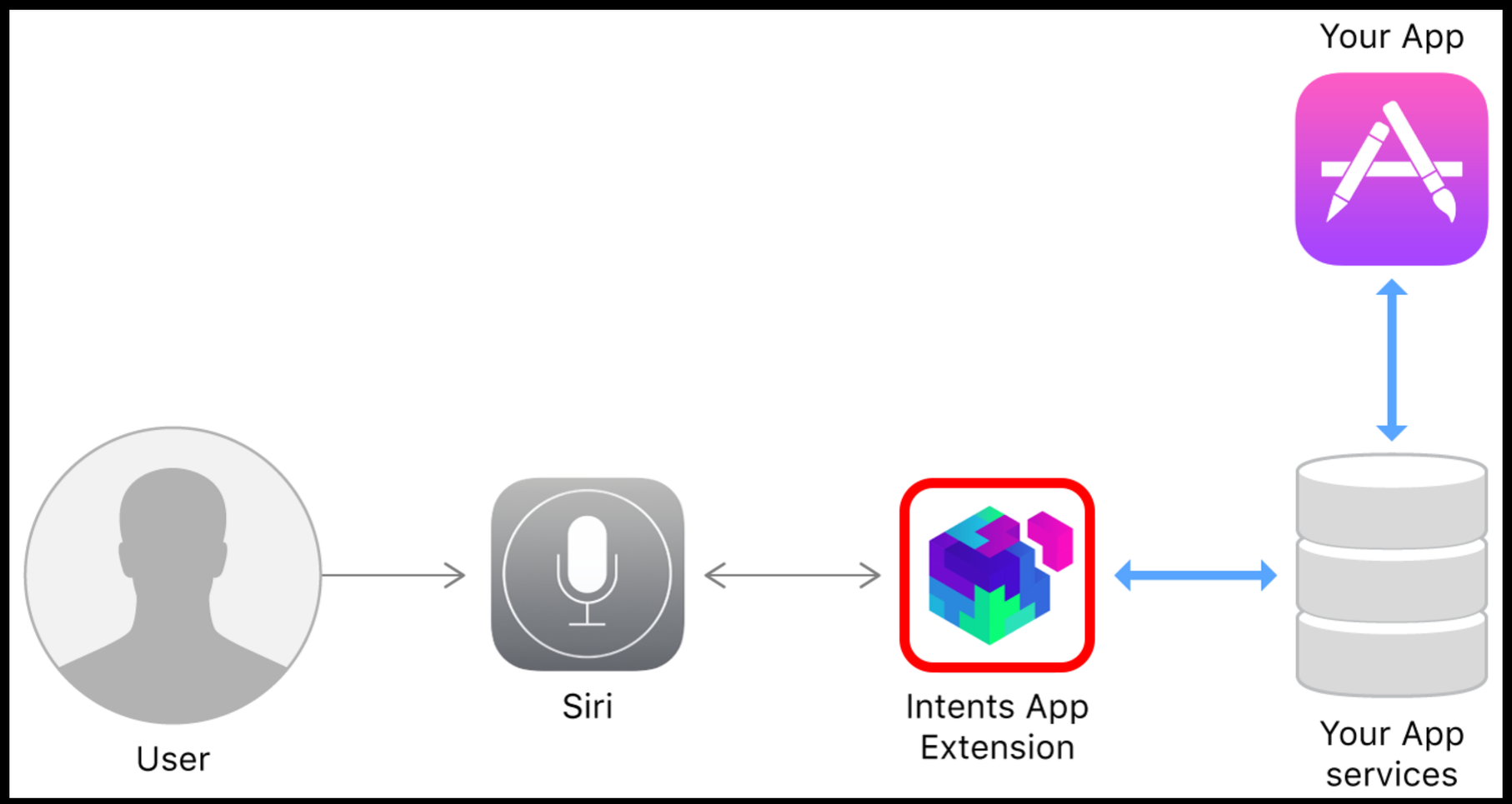
SiriKit encompasses the Intents and Intents UI frameworks, which are used to implement app extensions that integrate your services with Siri and Maps.

^ SiriKit supports two types of app extensions:
^ An Intents app extension
^ An Intents UI app extension

To implement SiriKit your iOS version needs to be at least iOS 10.

Make sure your embedded framework does not contain APIs unavailable to app extensions
^ (Legacy project having to basically re-create the api calls)
^ because the newer extension was unable to work with the legacy code's libraries... which were also in Objective-C)

What is an ‘Intents App Extension’?

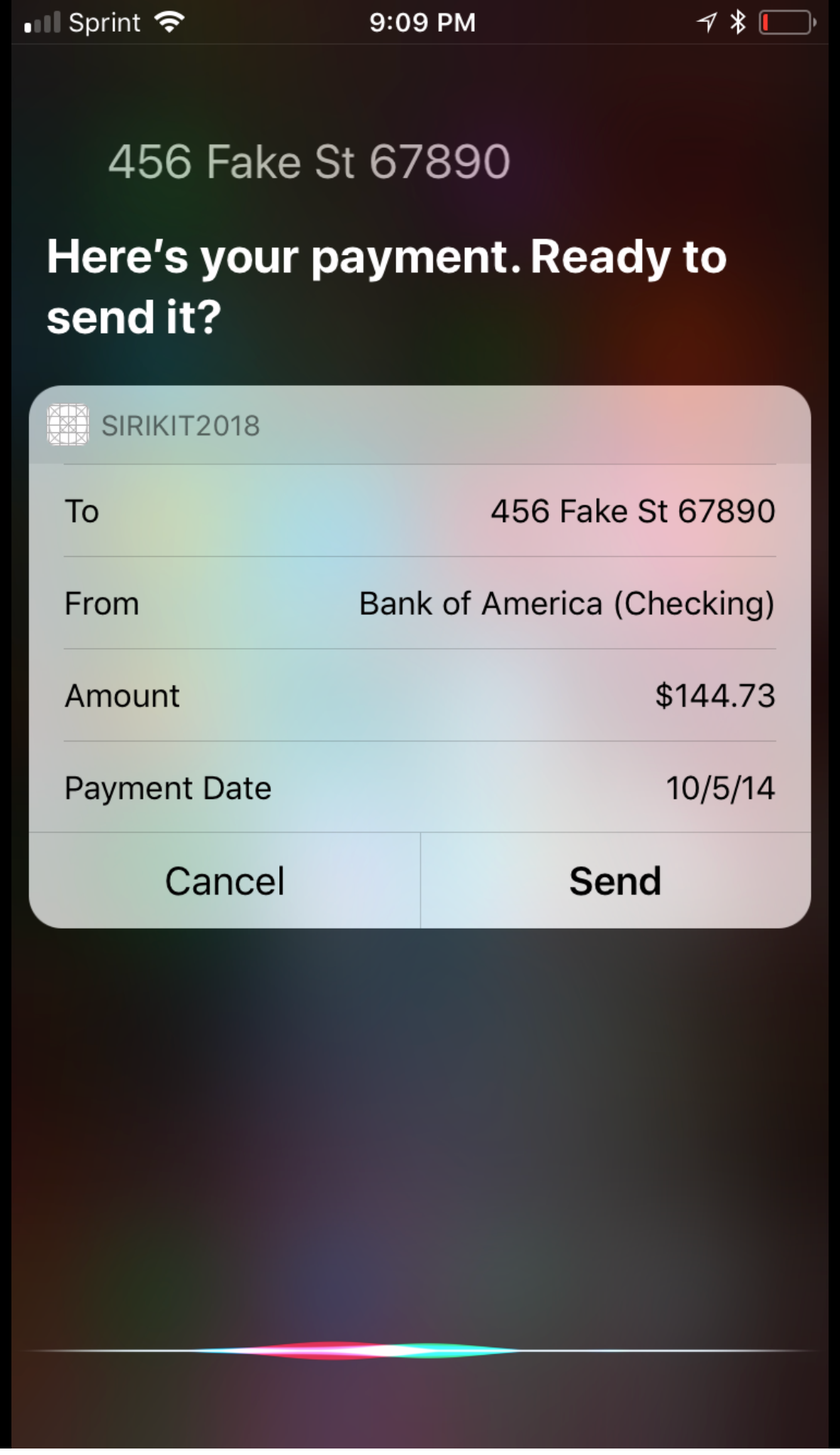


An Intents app extension receives user requests from SiriKit and turns them into app-specific actions

So like the video I showed earlier, your user might ask Siri to send a message, book a ride, or pay a bill using your app

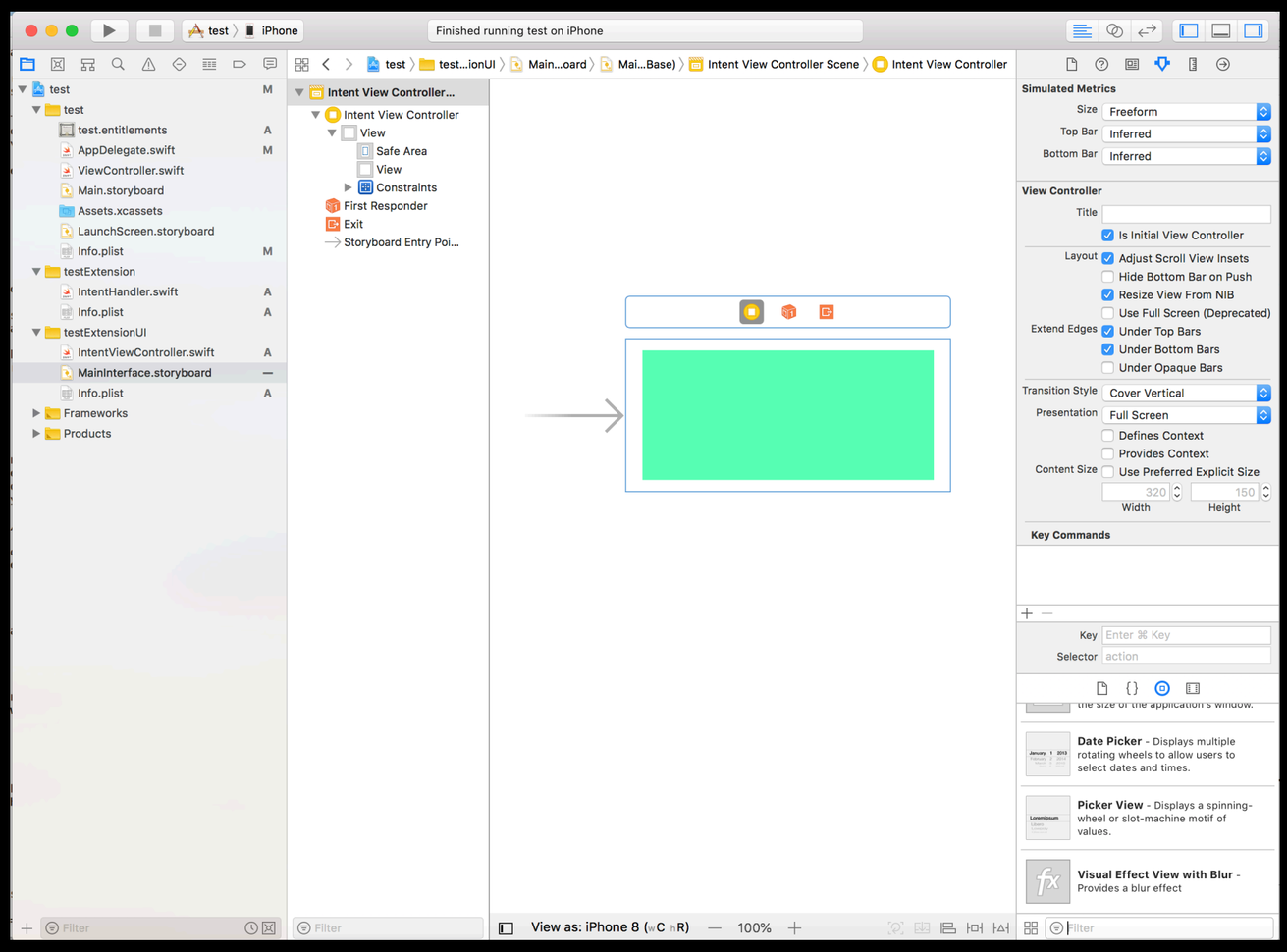
What is an ‘Intents UI App Extension’?

- Optional
- Syle and customize content



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 left is a screenshot of an un-styled SiriKit Extension Interface
^ I chose not add custom styling or change the interface for my SiriKit extension but I will show you a quick and simple example of what you could do.

Intents UI App Extension Storyboard

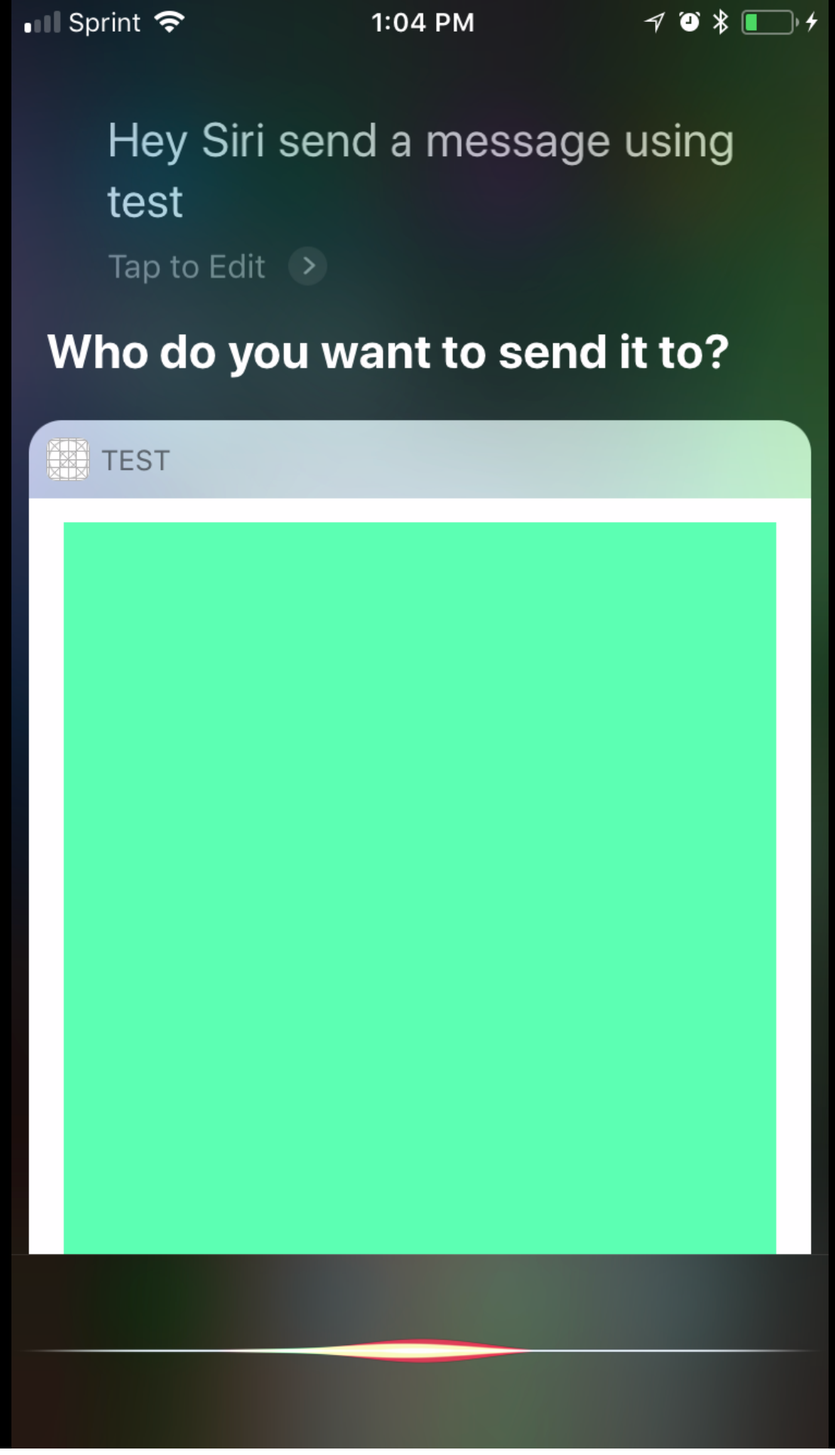


This is an image of application in Xcode that has both an Intents UI App Extension and an Intents App Extension

^ The Intents UI App Extension comes with a ViewController and a Storyboard that you can customize

^ You Can pick colors, animations, font styles etc. I just put in a green UIView, with no text or anything

Example of a Messaging app with a view inserted via an Intents UI App Extension



This is an example with a SiriKit Messaging app
^ It's set up just like my project it's just using the INSendMessageIntent instead of the InPayBillIntent, most of the intents are very similar in how they work though
^ You can also show and hide various parts of the UI

There are design guidelines in Apple's Developer Documentation that say things along the lines of:
^ Make sure that:
^ There's enough padding for the app content
^ That the content doesn’t scroll
^ Avoid making the interface have extra elements that appear interactive, as the only thing the Siri interface can respond to is a tap, so you can’t drag things around etc
^ And lastly don’t include the app name or icon in your custom interface as the system automatically shows this info.

What are Intents?

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

The app I'm explaining today uses the INPayBillIntent from the ‘Payments’ domain. In the Payments domain you can send payments between users or pay bills

The other domains include:
^ VoIP Calling, Messaging,
^ Lists and Notes, Visual Codes, Photos
^ Workouts, Ride Booking, Car Commands
^ CarPlay, Restaurant Reservations

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)

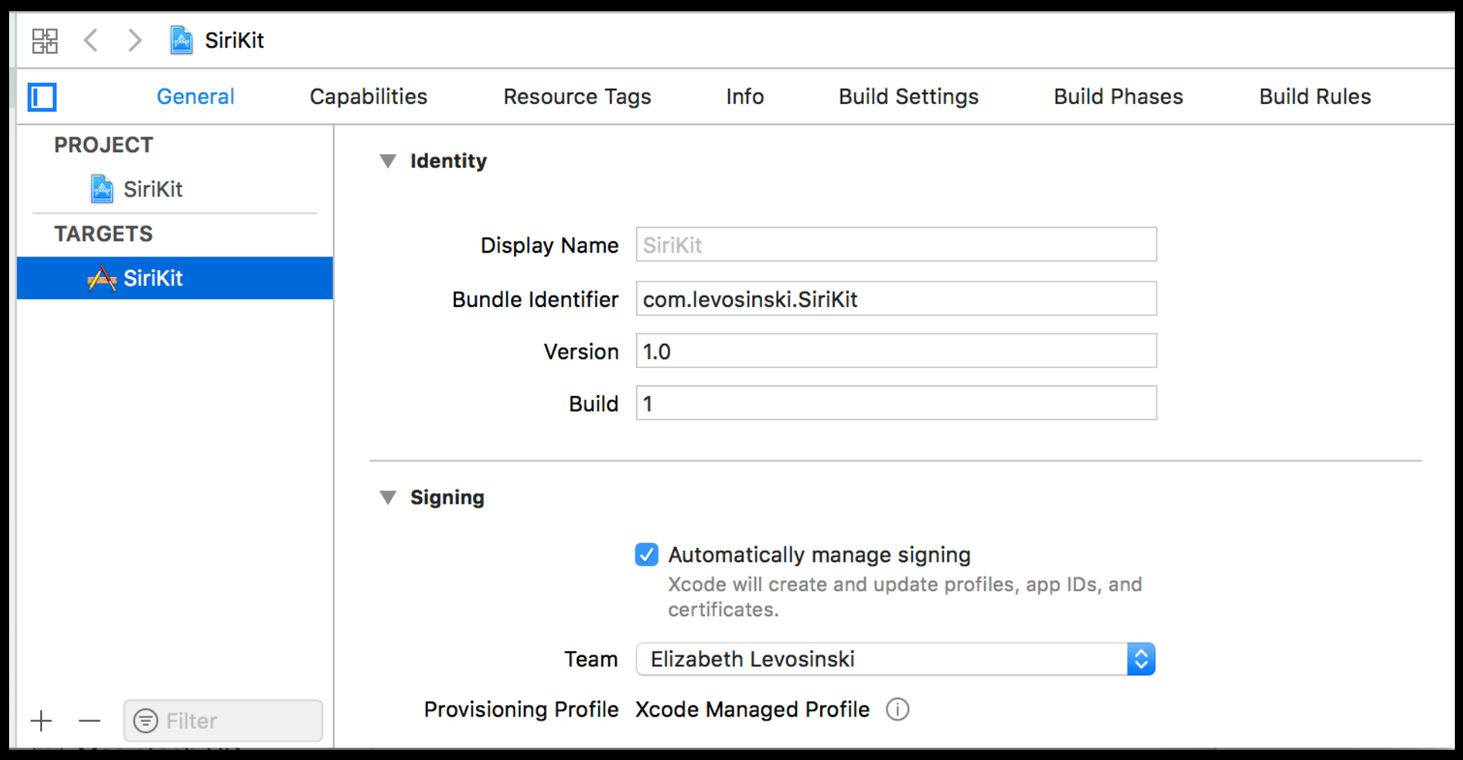
Now I'll go more in depth into the Intent I'm using for this app

^ All of these properties must be filled out for the payment to succeed

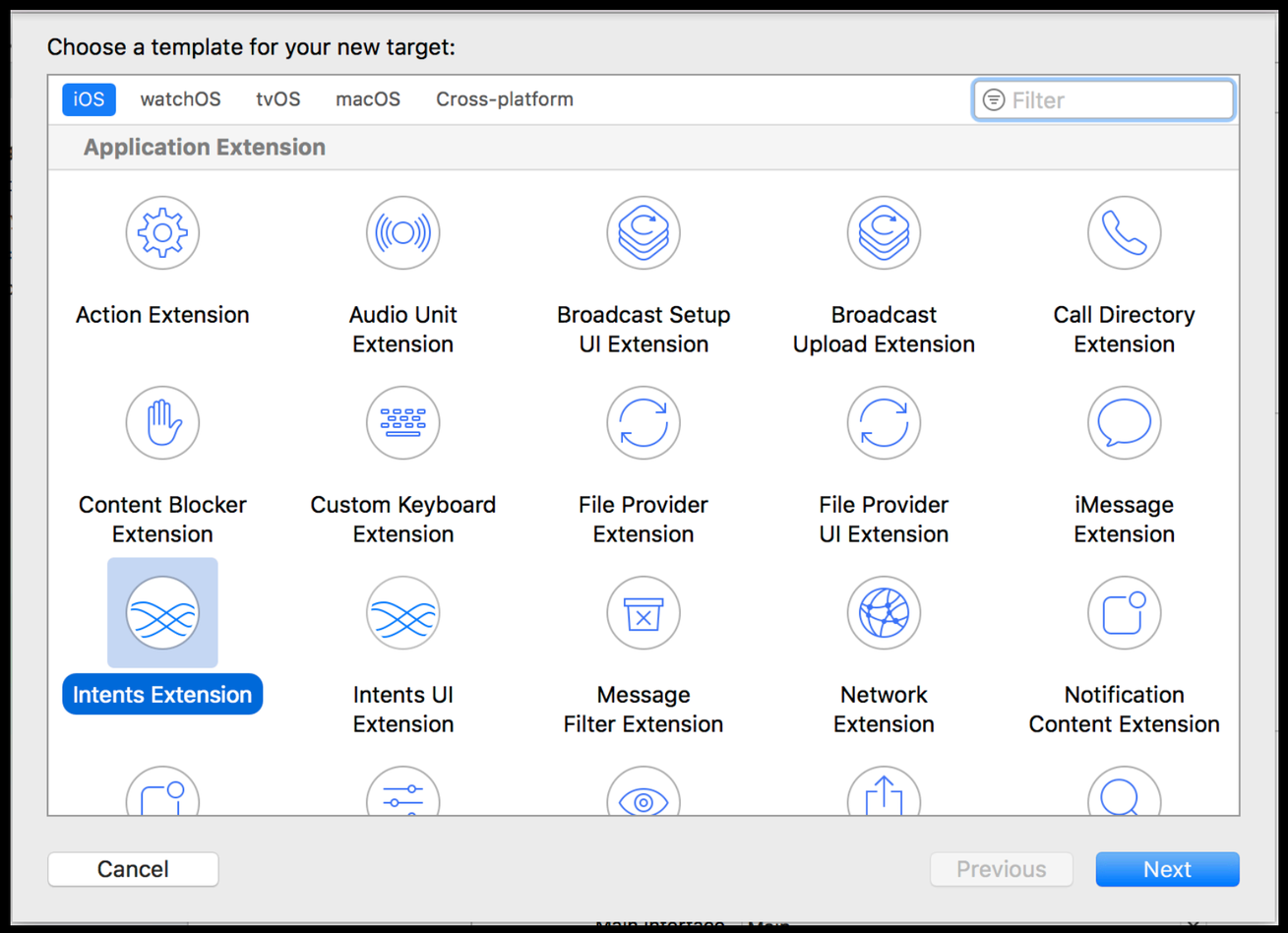
^ For instance, if the user had a water bill, the bill payee is defined by the user’s account with the water company (account number, organization name, and account nickname), this comes up again later

Steps to add SiriKit to your app:

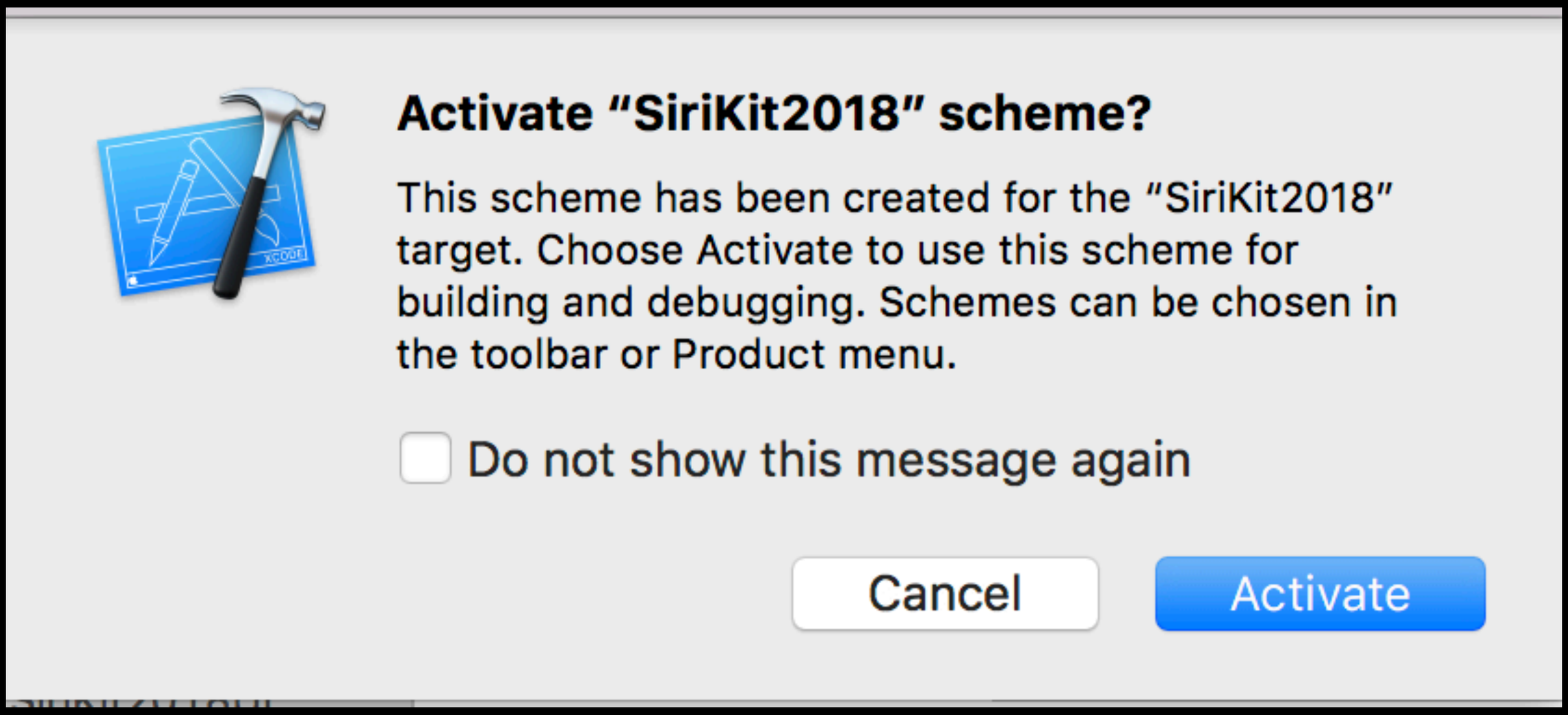
- Add an Intents extension by selecting your project target
- Then pressing the bottom left plus sign



- Select Intents Extension
- Then enter a name for it



- Select Activate scheme



- Go into the Info.plist in the newly generated extension folder
- Under NSExtension > NSExtensionAttributes > IntentsSupported
- Delete the default intents and add the ones you want to use

Key	Type	Value
▼ Information Property List	Dictionary	(10 items)
Localization native development re...	String	$(DEVELOPMENT_LANGUAGE)
Bundle display name	String	testextension
Executable file	String	$(EXECUTABLE_NAME)
Bundle identifier	String	$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	$(PRODUCT_NAME)
Bundle OS Type code	String	XPC!
Bundle versions string, short	String	1.0
Bundle version	String	1
▼ NSExtension	Dictionary	(3 items)
▼ NSExtensionAttributes	Dictionary	(2 items)
▶ IntentsRestrictedWhileLocked	Array	(0 items)
▼ IntentsSupported	Array	(3 items)
Item 0	String	INSendMessageIntent
Item 1	String	INSearchForMessagesIntent
Item 2	String	INSetMessageAttributeIntent
NSExtensionPointIdentifier	String	com.apple.intents-service
NSExtensionPrincipalClass	String	$(PRODUCT_MODULE_NAME).IntentHandler

Under the NSExtension Dictionary inside the IntentsSupported array ^ in our case we enter in the INPayBillIntent

- In the containing application's AppDelegate:

```
import Intents
```

be sure to add import Intents at the top so that you can add the request authorization code

```
// This allows the user to grant app permission to use Siri commands

INPreferences.requestSiriAuthorization { status in
    if status == .authorized {
        print("Siri services have been authorized")
    } else {
        print("Siri services have not been authorized")
    }
}
```

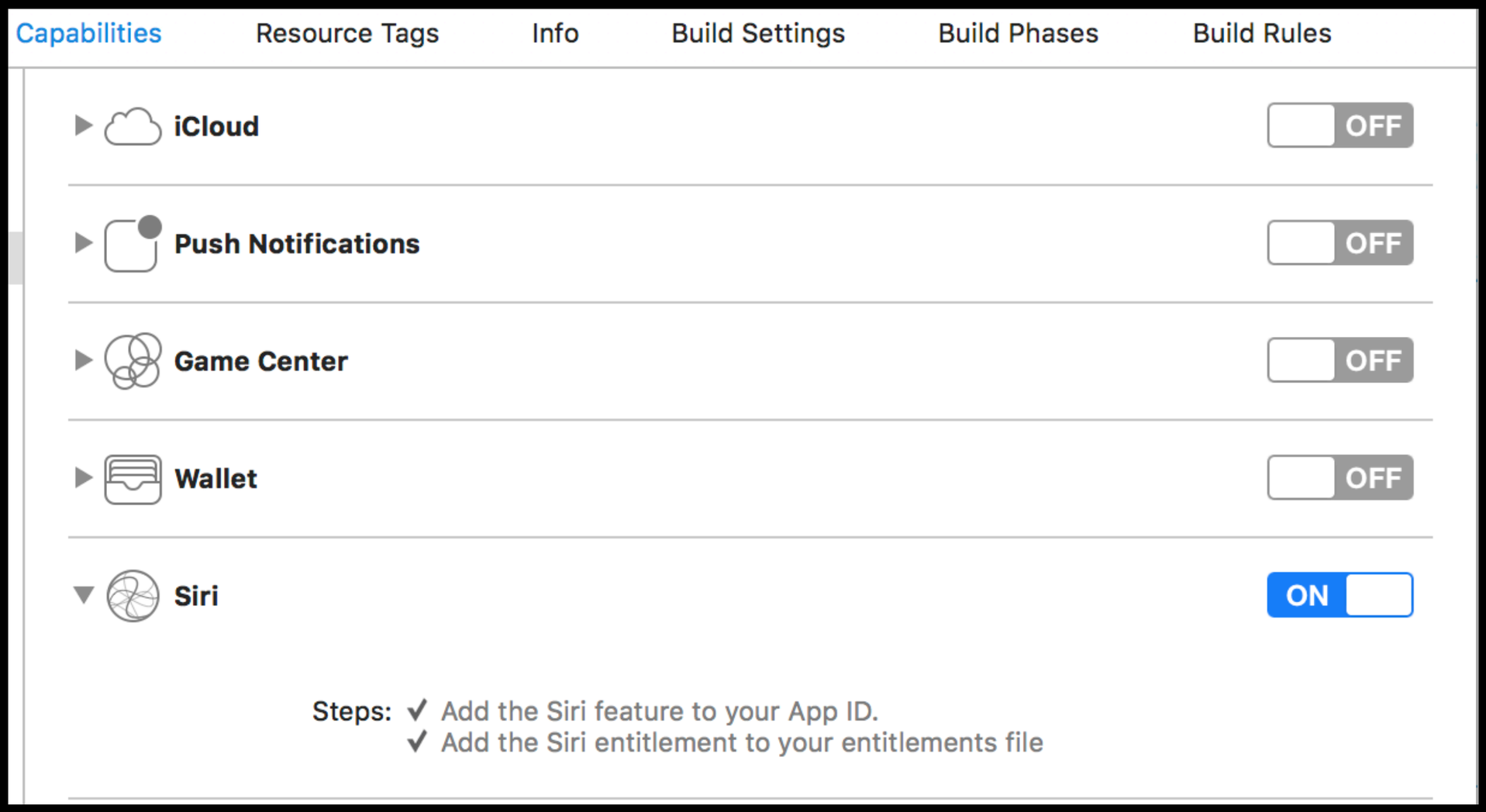
You need to add this so that the user can grant the app permission to use Siri commands in the containing app in the AppDelegate's didFinishLaunchingWithOptions function:

- Add a Siri usage description to the app's Info.plist

SiriKit2018 〉 SiriKit2018 〉 Info.plist 〉 No Selection

Key	Type	Value
▼ Information Property List	Dictionary	(15 items)
Privacy - Siri Usage Descripti...	String	Allows users to pay their bill with Siri
Localization native development re...	String	$(DEVELOPMENT_LANGUAGE)
Executable file	String	$(EXECUTABLE_NAME)
Bundle identifier	String	$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	$(PRODUCT_NAME)
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.0
Bundle version	String	1
Application requires iPhone enviro...	Boolean	YES
Launch screen interface file base...	String	LaunchScreen
Main storyboard file base name	String	Main
▶ Required device capabilities	Array	(1 item)
▶ Supported interface orientations	Array	(3 items)
▶ Supported interface orientations (i...	Array	(4 items)

- Add the Siri entitlement by flipping the Siri capabilities switch



These are all the steps to setup SiriKit with your app and you can check them out in more detail in the resources I've attached to my slides if you wish

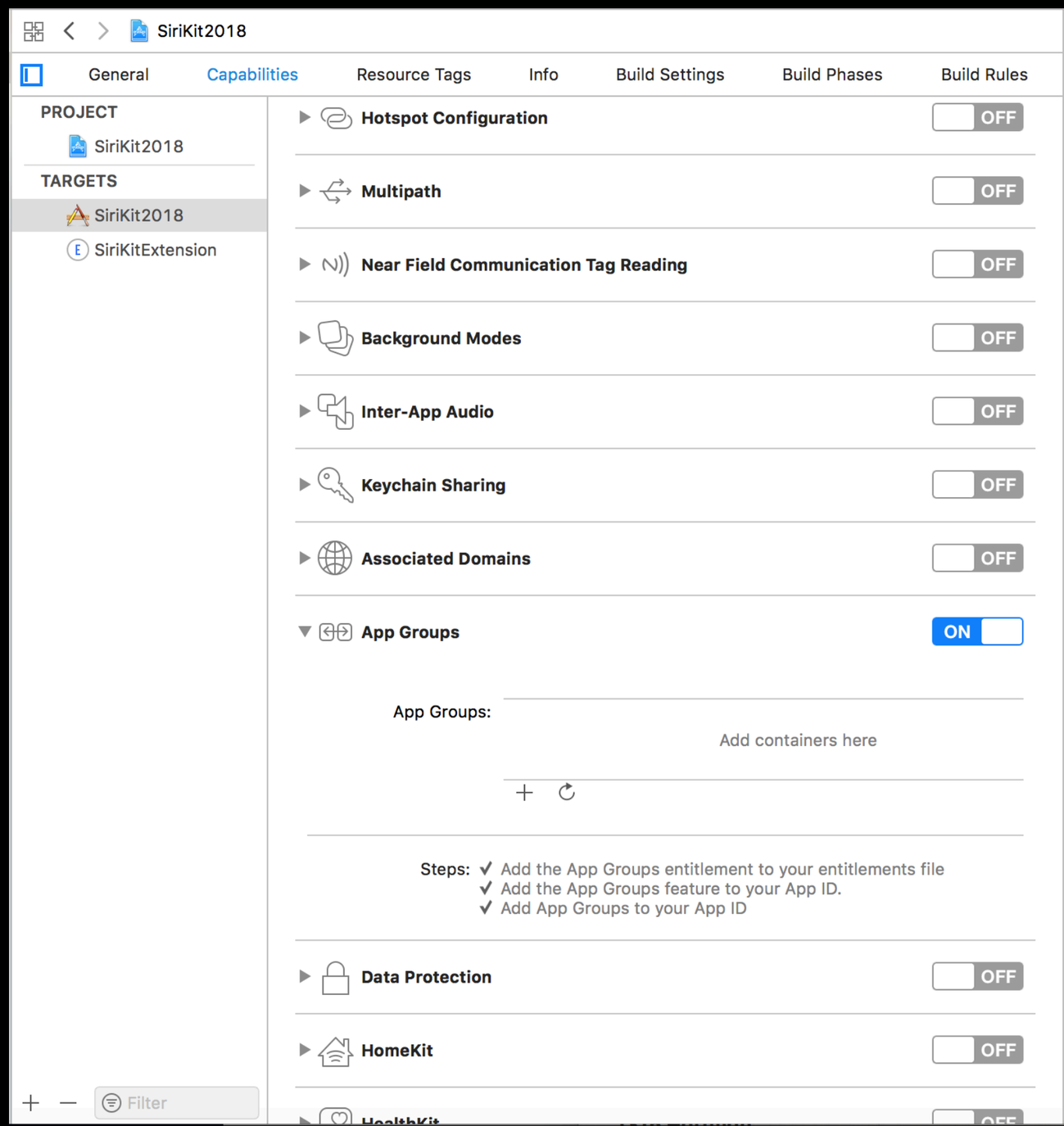
App Groups and Sharing Data with the Containing App

- To enable data sharing, use Xcode or the Developer portal to enable app groups for the containing app and its contained app extensions.
- Next, register the app group in the portal and specify the app group to use in the containing app
  - Go to your developer Account
  - Go to Certificates, Identifiers & Profiles
  - Select Target > Capabilities > App Groups > ON

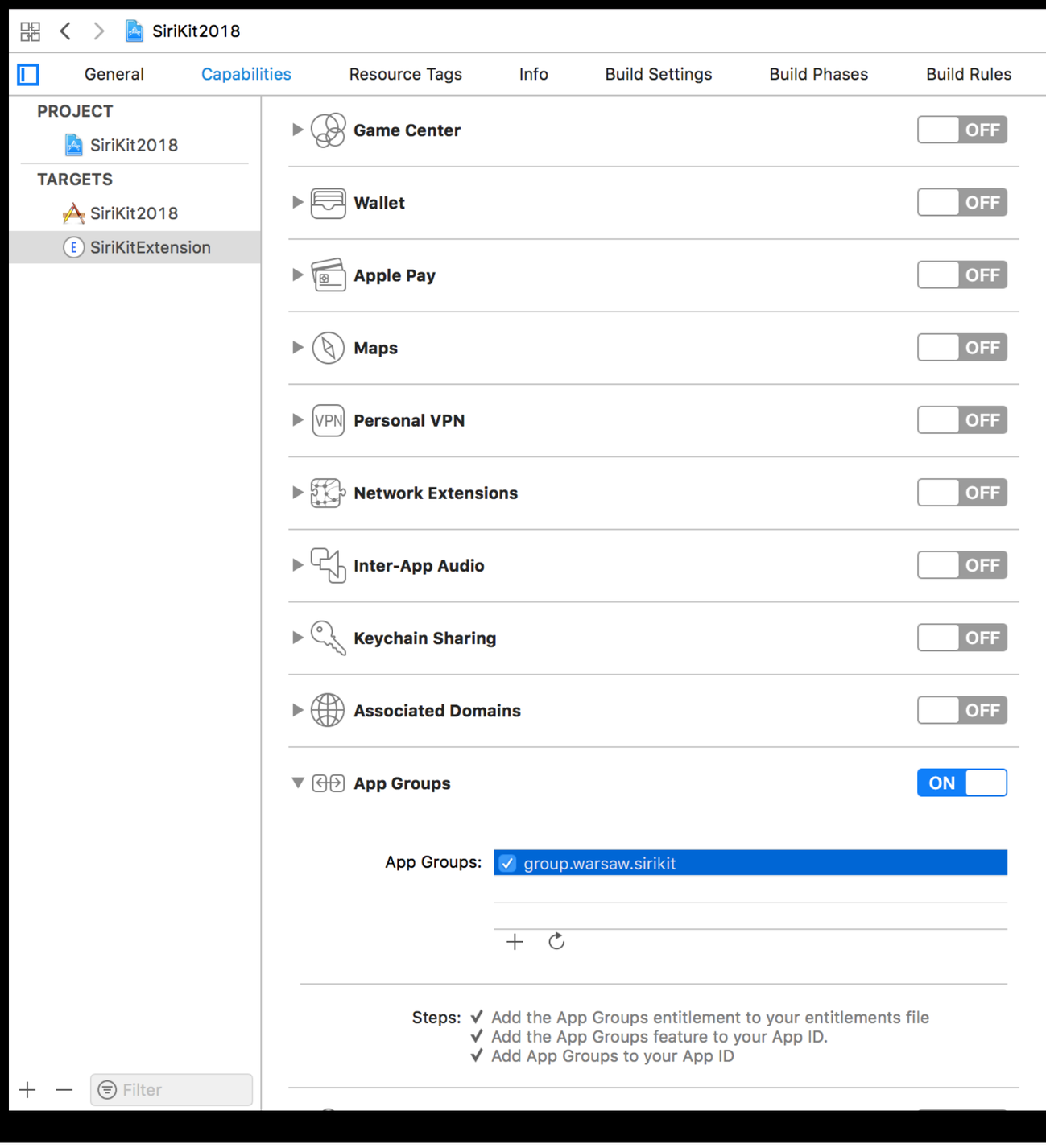
After adding an Intent to the app you'll need to add an App Group so you can tell if the user is logged in or not

^ I use an App Group because the app and the extension are separate and cannot directly share data. (something I had learned about extensions in general)

- Tap on Plus sign under App Groups to add your App Group



- Then just select the extension you want to share this App Group
- Turn App Groups to "ON"
- Check the group you just created



Once you do the these steps, the app group automatically added to your Developer Account under Identifiers

^ It also enables App Groups on the App ID

After enabling App Groups the SiriKit extension and its containing app can both use the NSUserDefaults API to share access to user defaults

```
// If App Group exists, initializes a UserDefaults object
// which when returned will give the Bool value of true or false

public static func userIsLoggedIn() -> Bool {
    guard let defaults = UserDefaults(suiteName: suiteName) else {
        return false
    }
    return defaults.bool(forKey: key)
}
```

Inside the containing App:

When a user presses the "Sign In" button, the app checks to see if the App Group exists

^ If it does it initializes a UserDefaults object using that App Group with the suiteName that it was given earlier

^ The suiteName is the name you gave your App Group in the Developer Portal, in our case it's "group.warsaw.sirikit"

^ With User Defaults you can share any object type, String, Int, Bool, Array etc.

^ In our case a boolean was appropriate because we just want to know if the user is signed in or not

```
//  Updates the User Defaults boolean and
//  synchronizes the defaults

public static func updateSharedData(status: Bool) {
     if let defaults = UserDefaults(suiteName: suiteName) {
          defaults.set(status, forKey: key)
          defaults.synchronize()
     }
}
```

This function updates the User Defaults boolean and synchronizes the defaults so they are updated in both the extension and the containing app

```
@IBAction func signIn(_ sender: UIButton) {
      userSignIn { error in
         statusLabel.text = CustomMessages.loggedIn

         // Update user defaults so we know user is signed in
         Helpers.updateSharedData(status: true)
      }
}
```

The updateSharedData function is called whenever the signIn or signOut IBActions are triggered by the user pressing the buttons in the containing app's View Controller

SiriKit Extension Files:

- IntentHandler.swift
- Info.plist

After creating an App Group and adding the extension to our app, a new folder was generated in the project. This folder contains two files: IntentHandler.swift and 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

During the Resolution phase, the parameters of the intent object are validated to make sure you have the info you need to continue
^ During the Confirmation phase: Confirming the details on an intent (Perform the final validation of the intent parameters/verify your services are ready)
^ Handling the intent step: This ensures that the Intent is fulfilled and provides 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
- resolveBillType

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 = CustomMessages.transactionNote
    return response
}
```

Pitfall

^ If any of the information is missing SiriKit won't move on to the confirmation phase, 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 containing 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)

```swift
func finishBillDetails(account: Account, billPayee: INBillPayee) {
        let charges = account.summaryOfCharges
        let billTotal = charges.billTotal

        guard let chargesDueDate = charges.billDueDate.convertStringToDate(),
            let chargesStart = charges.startDate.convertStringToDate(),
            let chargesEnd = charges.endDate.convertStringToDate() else { return }

        let dueDate: DateComponents = chargesDueDate.convertDate()
        let paymentDate = Date().convertDate()

        // Converting the account data to the correct currency for the INBillDetails object
        let lateFee = INCurrencyAmount(amount: 0, currencyCode: "USD")
        let amountDue = INCurrencyAmount(amount: NSDecimalNumber(value: billTotal), currencyCode: "USD")
        let minDue = INCurrencyAmount(amount: NSDecimalNumber(value: billTotal), currencyCode: "USD")

        // This is setting the actual amount the user is going to pay off, currently this example just
        // automatically has the user paying the bill all off at once
        self.transactionAmount = INPaymentAmount(amountType: .amountDue, amount: amountDue)

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

        self.transactionScheduledDate = Helpers.getDateRange(startDate: chargesStart, endDate: chargesEnd)
    }
```

This function populates the INBillDetails object with the user's account information such as the summary of all the charges, the charges date ranges, late fees, the total due, sets the bill type, the minimum due, the bill's due date and the actual payment date

^ For this project we had the amount the user was going to pay just be the bill's total and set the payment date to be the current date.

^ Most of these properties are things you could customize in your apps if you chose to prompt your user's with resolve functions

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 addresses and I needed to know which address they were paying for.

resolveBillPayee part 1

```
func resolveBillPayee(for intent: INPayBillIntent, with completion: @escaping (INBillPayeeResolutionResult) -> Void) {
      APIManager.getCustomerData { error, customer in
          self.customerData = customer

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

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

The first part of this function checks to see if the user is logged in so we can access their account information

^ 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 confirmation phase

^ This function gets called a few times

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

resolveBillPayee part 2

```
else {
            guard let customer = customer else { return }
            APIManager.getUserAccountData(for: customer) { error, userAccounts  in
                self.accounts = userAccounts

                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 companyAcountNum: String?
      let companyName = INSpeakableString(spokenPhrase: "The Electric Company")

      if paymentAccounts.count > 1 {
          for account in paymentAccounts {
              companyAcountNum = account.number

              // The account nicknames are dynamically populated with a mailing address
              // to make it easy to differentiate between each account
              let companyAccountNickname = INSpeakableString(spokenPhrase: account.address.street + " " + account.address.zipCode)

              if let billPayee = INBillPayee(nickname: companyAccountNickname, number: companyAcountNum, organizationName: companyName) {
                  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’

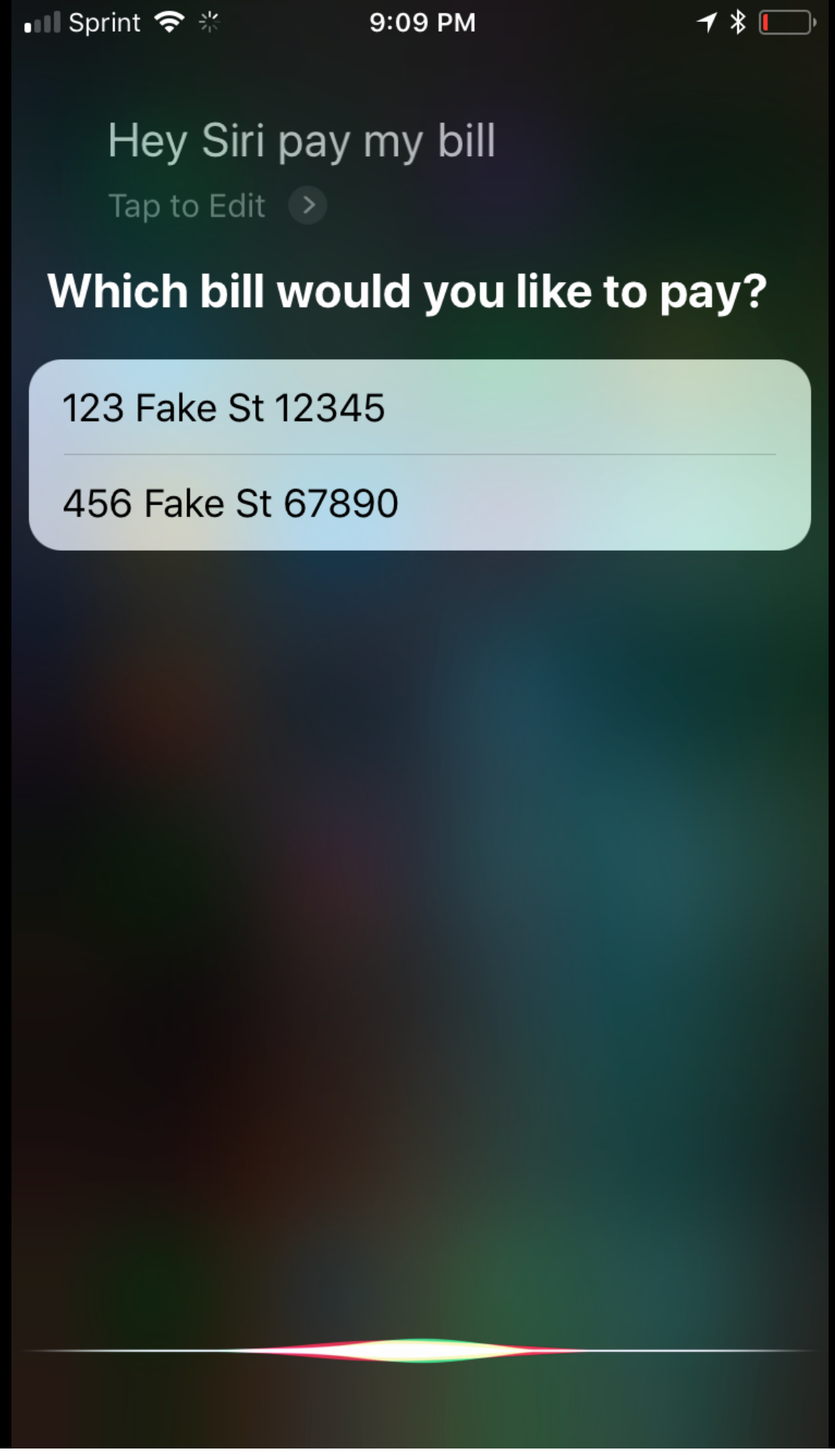
^ if they have more than one account I then call the ‘INBillPayeeResolutionResult.disambiguation’ function and pass in the array of ‘INBillPayee’s I generated

createBillPayee Part 2

```
else {
        let companyAccountNickname = INSpeakableString(spokenPhrase: "The Electric Company Payment Account")
        companyAcountNum = paymentAccounts[0].number

        if let billPayee = INBillPayee(nickname: companyAccountNickname, number: companyAcountNum, organizationName: companyName) {
            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)
```



This is a visual of when the INBillPayeeResolutionResult.disambiguation function is called so the user will choose an address.

```swift
func confirm(intent: INPayBillIntent, completion: @escaping (INPayBillIntentResponse) -> Void) {
  if !Helpers.userIsLoggedIn() {
        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.number == currentBillPayee.accountNumber {
            finishBillDetails(account: account, billPayee: currentBillPayee)
        }
    }

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

Confirming an Intent

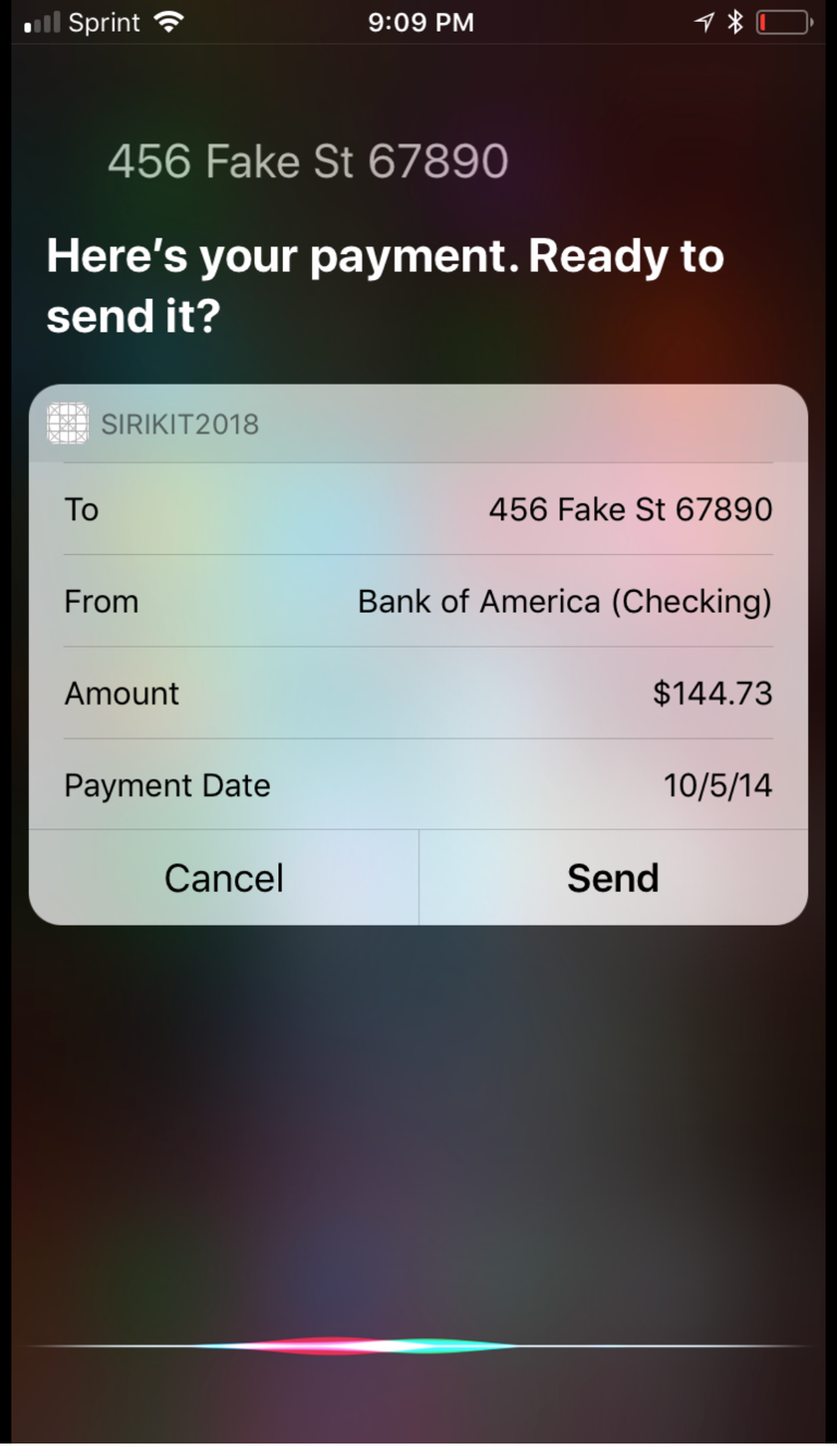
^ This step just ensures that the INPayBillIntentResponse that we are sending to SiriKit has all of its properties filled

^ Also I just want to point out in the INPayBillIntentResponse the userActivity property

^ The userActivity is set to nil because its not needed in our example but

^ It could be used to pass information to your App Delegate from your SiriKit extension

Confirming an Intent

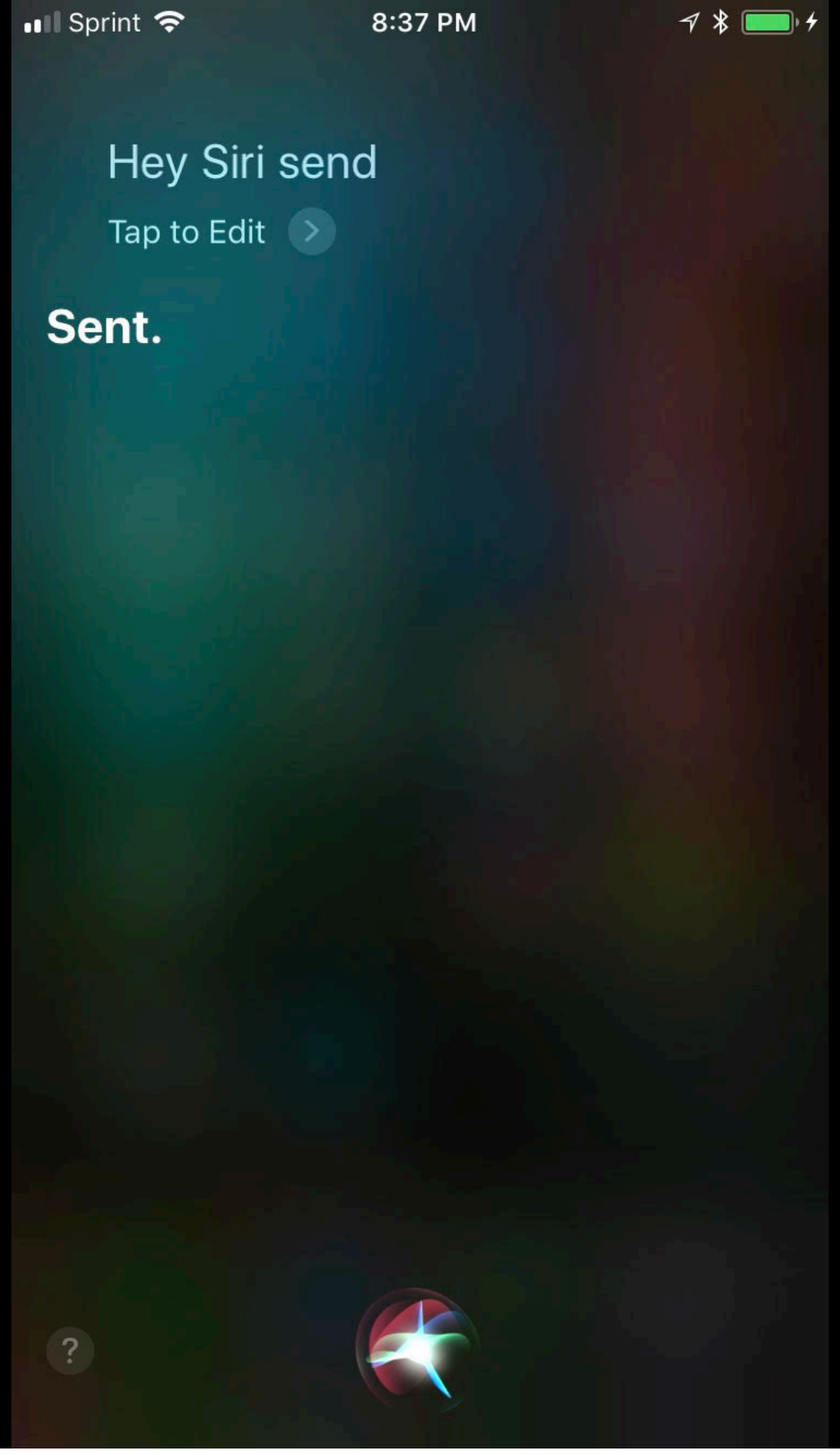


Handling an Intent

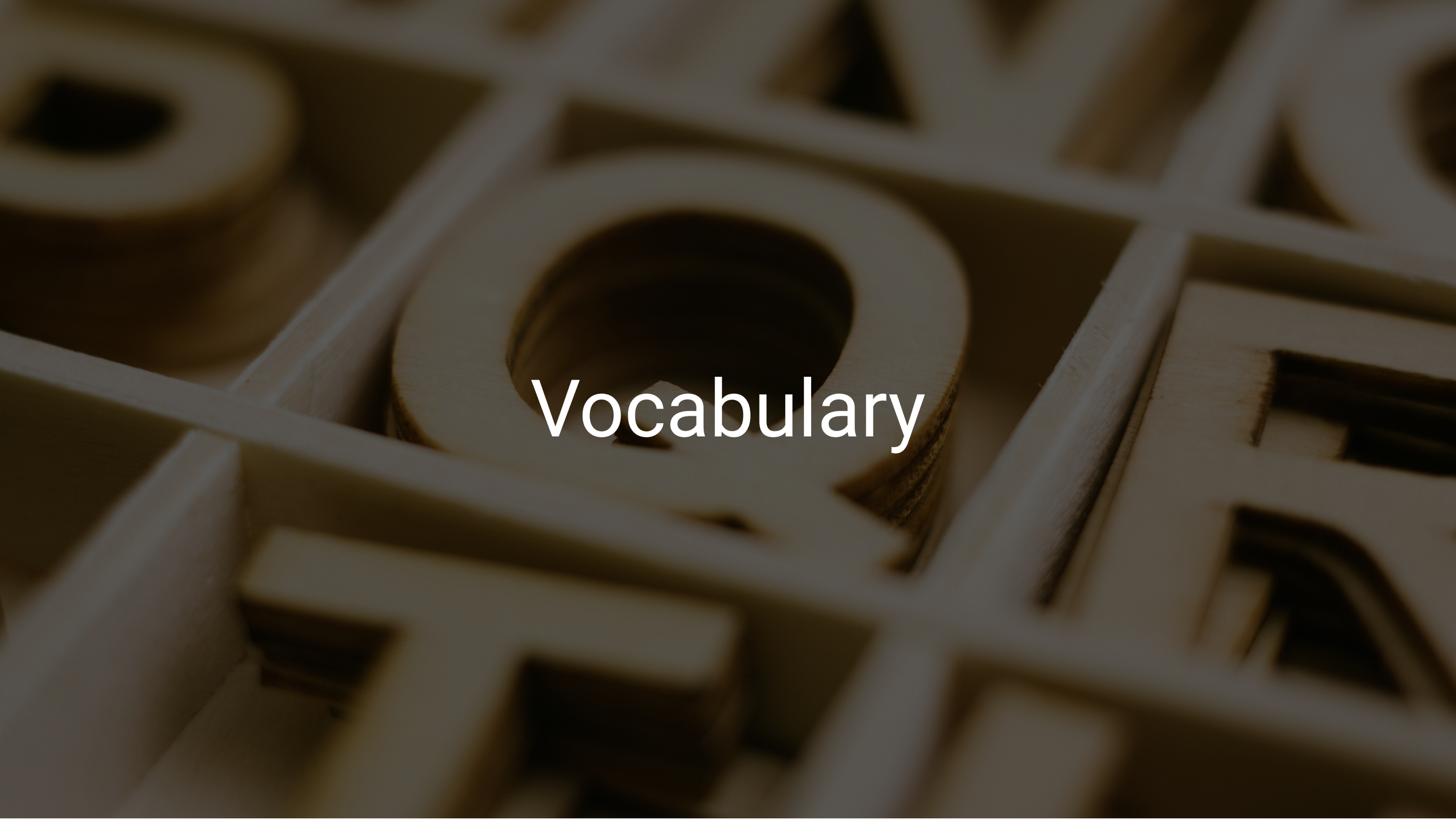
```
func handle(intent: INPayBillIntent, completion: @escaping (INPayBillIntentResponse) -> Void) {
    createPayBillIntentResponse(codeType: .success) { (response, savedPaymentMethod) in
          guard let savedPaymentMethod = self.savedPaymentMethod else {
              completion(response)
     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

          APIManager.submitPayment(savedPaymentMethod: savedPaymentMethod, successfulResponse: response, complete: completion)
    }
}
```

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

Handling an Intent




Extras

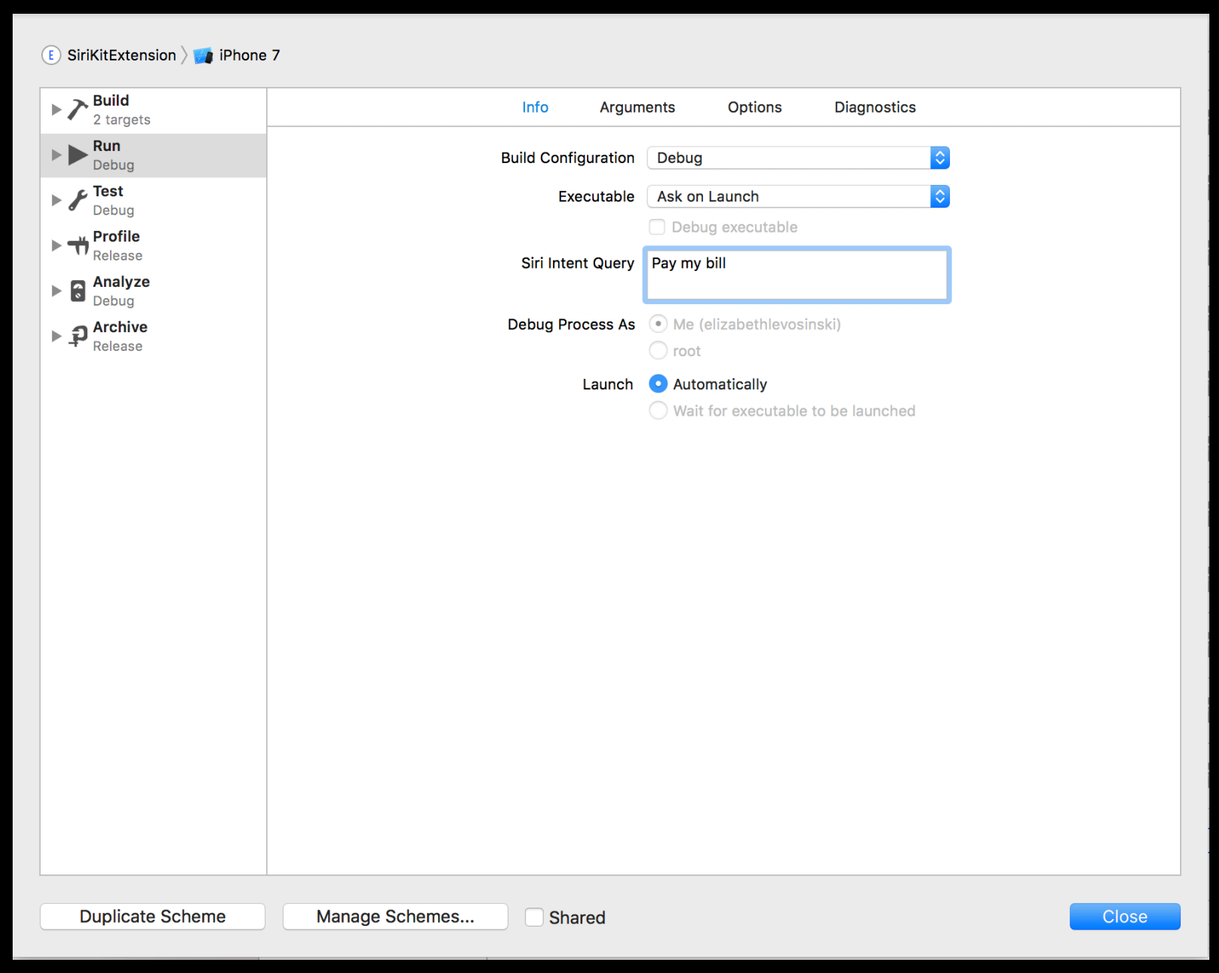


You could define app-specific terms for your users to make requests through Siri

You can declare terms that are:
^ Contact names, Contact groups, Photo tags, Photo album names, Workout names, Vehicle profile names, Car names, Payment organization names or Payment account nicknames

It's suggested that when you register vocabulary don't pick things that are easily understood by Siri already like "My Workout" or "My Photo Album", you want to register vocabulary that is used non conventionally or words that are completely unique to your app

Testing



Since Xcode 9, you can edit the SiriKit Extension scheme to have a default Siri Intent Query so that you don't have to look strange saying the same query out loud:

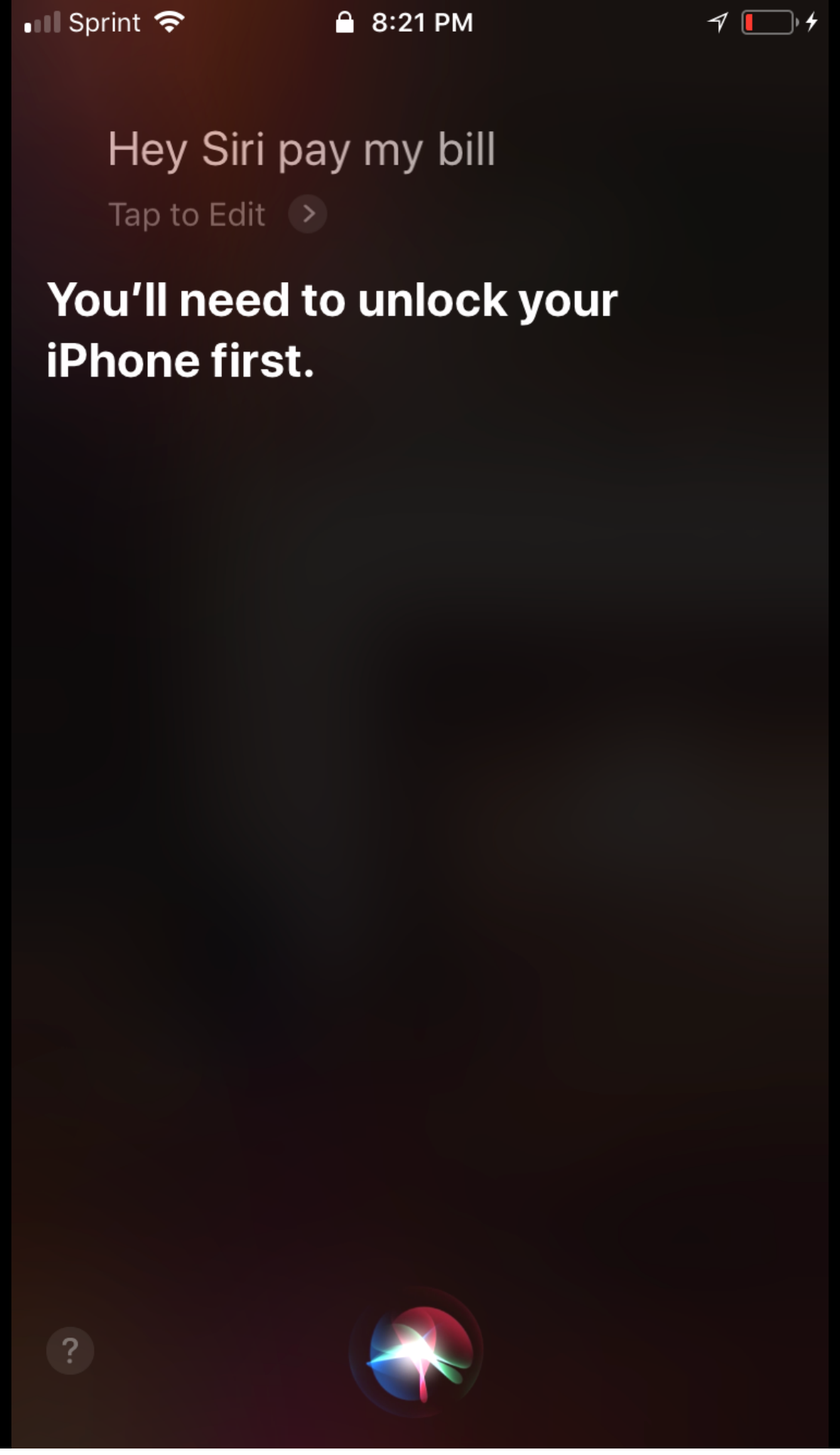
Product > Scheme (make sure the extension is selected) > Edit Scheme

^ Select the Info Tab

^ Inside the Siri Intent Query input type the command (Pay my bill)

Security

- Populate **IntentsRestrictedWhileLocked** with desired **IntentsSupported** values



Depending on what intent you use, you may be able to access Siri from the lock screen. If you have an intent that can be accessed from there you should consider adding an extra security step:

One way you can do that is by populating the 'IntentsRestrictedWhileLocked' array with any IntentsSupported values (which would've been filled out when the app was initially set up)

If you go into your Extension's Info.plist
^ NSExtension (Dictionary) > NSExtensionAttributes > IntentsRestrictedWhileLocked (Array)
^ Add any values from the IntentsSupported array, that you want to restrict (INPayBillIntent)
^ That way if the user invokes that Intent the OS will prompt the user to unlock the device first

SiriKit on HomePod

- HomePod can communicate to apps with the following Domains:
- **Lists and Notes**
- **Messaging**

Currently you can speak to the HomePod, giving it SiriKit requests like Lists and Notes or Messaging and Siri recognizes SiriKit requests made on HomePod and sends those requests to relevant apps on iOS devices you have connected and communicating with the HomePod.
^ To prepare your app to interact with the HomePod in this way, make sure that your SiriKit integration is up to date and that you've adopted all of the appropriate intents.

Resources

- Intents UI Extension Tips: https://developer.apple.com/documentation/sirikit/creating*an*intents*ui*extension/configuring*the*view*controller*for*your*custom_interface
- SiriKit on HomePod: https://developer.apple.com/sirikit/
- 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/
- https://www.prolificinteractive.com/2017/10/06/using-sirikit-ios-apps/
- https://www.bignerdranch.com/blog/sirikit-part-3-finishing-touches/
- https://www.apple.com/homepod/

The Apple Documentation and the Ray Wenderlich tutorial were the most helpful

Questions?

Download Project and Slides here:

https://github.com/levosins/sirikit