Square POS SDK with Swift

This document will help you configure a Swift UIKit project for use with the Square POS SDK. The official Square documentation does not reference SwiftUI, so this document does not support it. Most of the configuration steps below will still be applicable to a SwiftUI project, though.

Create an account with Square

Visit https://squareup.com and create a business account. The Square POS system currently does not support a sandbox environment, so you will need to build an account that can actually process cash and credit card transactions. Cash transactions are free but credit card payments are subject to fees. Immediately after making a credit card transaction, you can refund it. Also, you can charge as little as $1.00 to avoid excessive credit holds.

Download the Square POS app

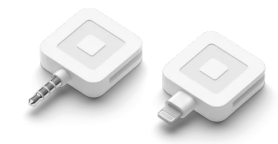
Visit the App Store and install the Square Point of Sale & Payment app (https://apps.apple.com/us/app/square-point-of-sale-payment/id335393788). The app you create will interface with the Square app to make secure payments with their processing system.

Run the app and login in with your Square account. Be sure to allow access to all of the required device hardware resources.

Get the Square credit card reader hardware

There are several hardware options available with differing capabilities. Visit https://squareup.com/us/en/hardware and explore.

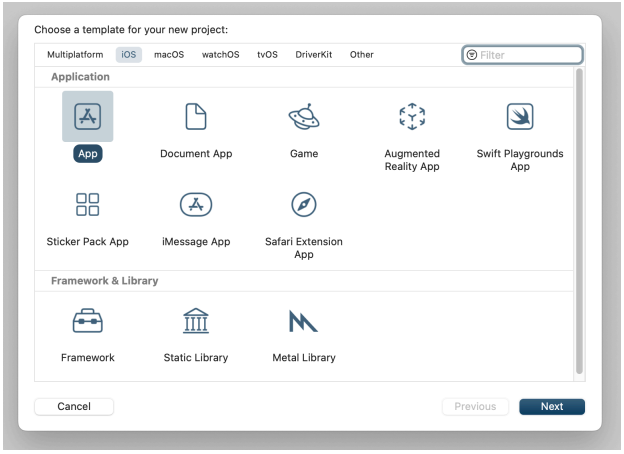
The Square Reader for mag-stripe is the least expensive option (and it's free if you charge $10 or more). Be sure to select the version that best works with your iPhone or iPad.

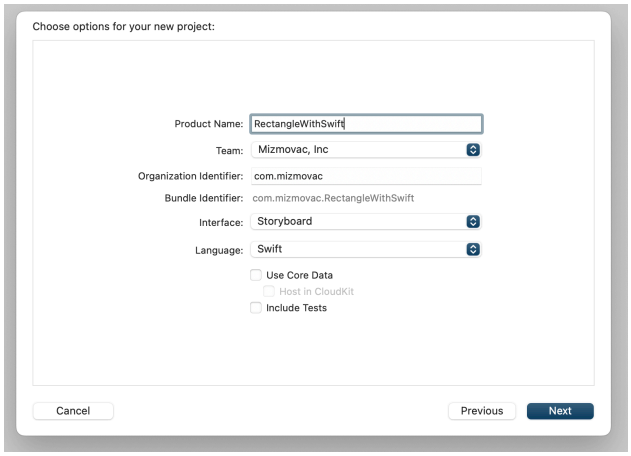


You can also use the reader with the Square POS app to make charges without having to build your own custom app. For example, you can use the reader and app to charge customers at a garage sale. Your investment in the reader will not be limited to apps you create.

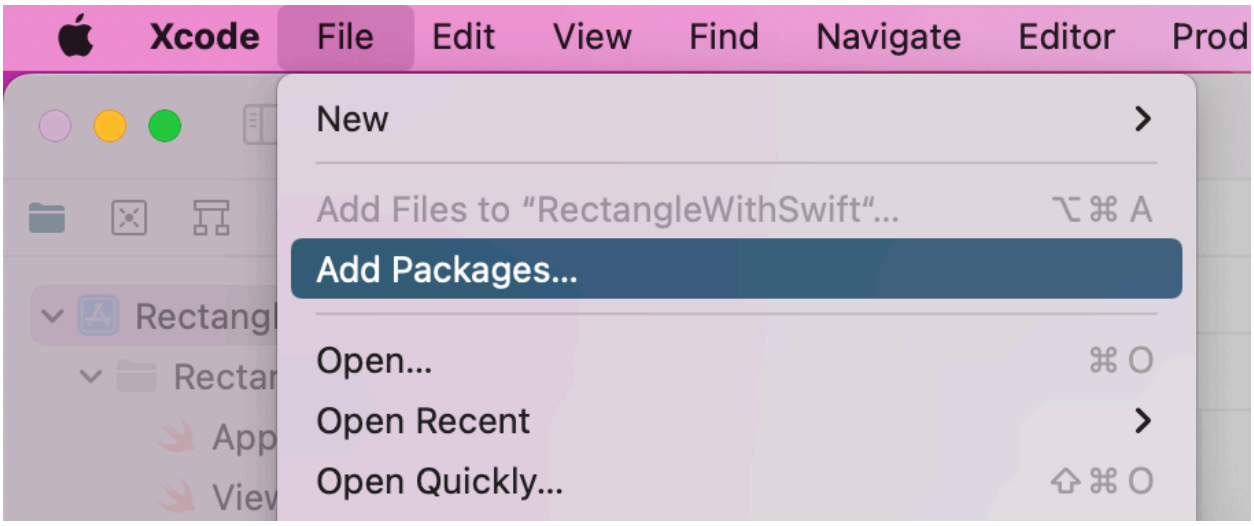
Configure your Xcode project

Build a new project in Xcode for iOS, Storyboard and Swift. Save your project.

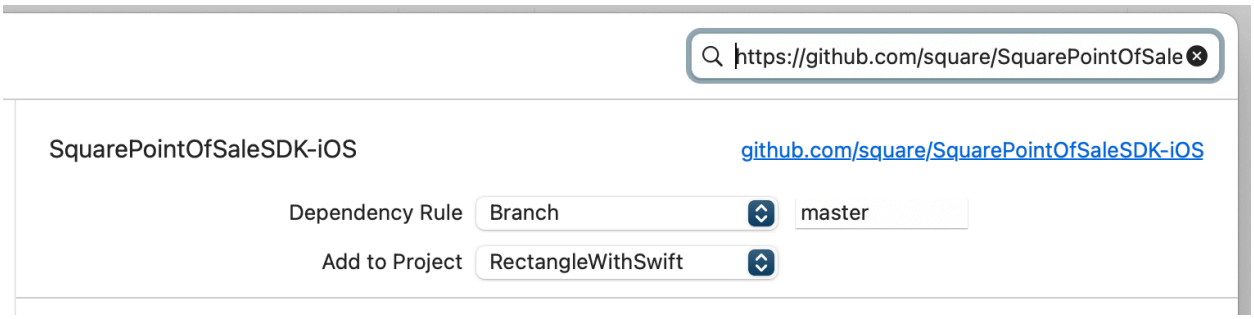




The Square SDK can be found on GitHub at https://github.com/square/SquarePointOfSaleSDK-iOS. Use the Xcode Package Manager to install the SDK into your project.

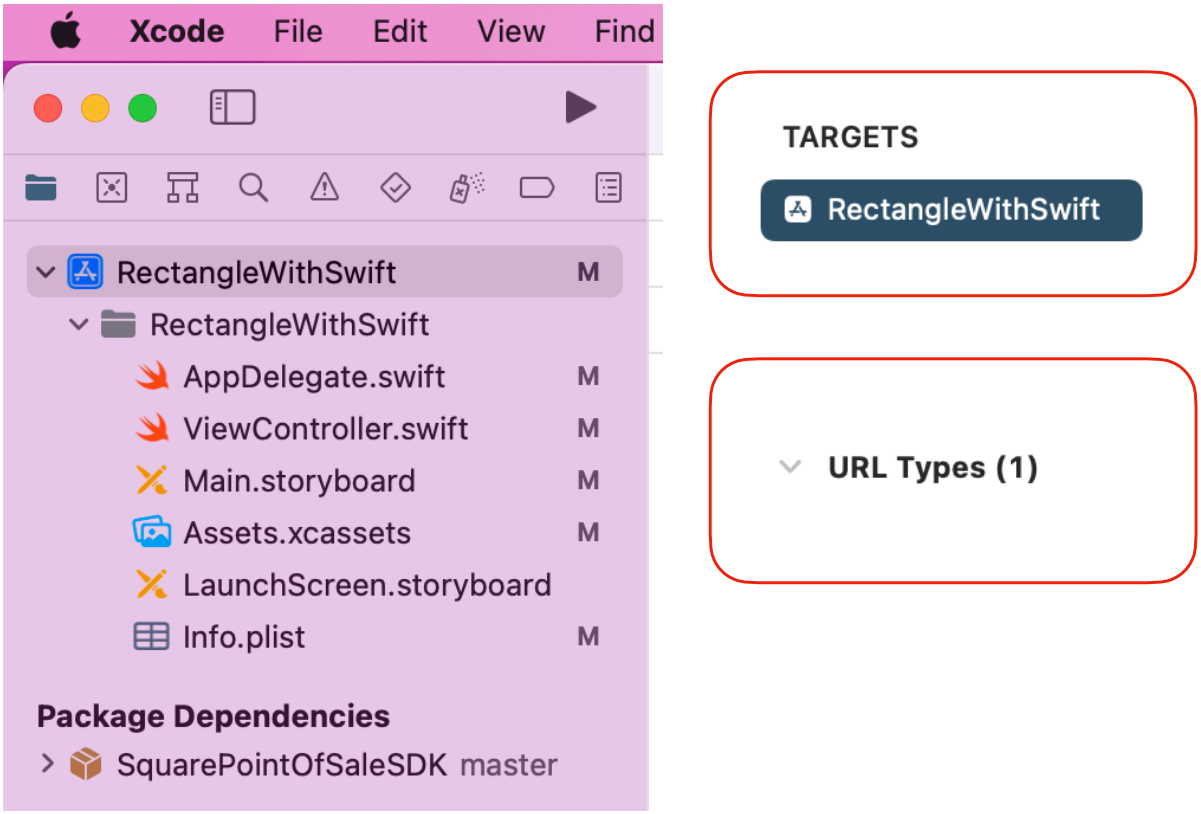


In the Package Manager window, paste the GitHub path Into the Search control and press Enter. Press the Add Package button to pull the latest version of the SDK and merge it into your project. Confirm that you have the kit installed in the Project Navigator under Package Dependancies.



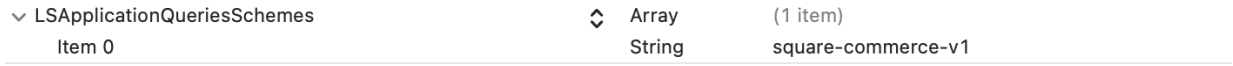
The Square POS app will communicate with your app using a custom URL. You will create a URL Type in your project and register it with Square.

In the Project Navigator, click on the project root. This will open the Project Attributes window. Click on the project target in the window and select the Info tab. Look for the URL Types category in the panels and expand it by clicking the chevron.



In the URL Types panel, you will need to create a URL Type member. Press the + button to open an editor for your new URL. In the Identifier field, enter **Square**. Enter a URL Scheme that is meaningful for your app. In this case, the name of the app is sufficient. Set the Role to **Editor**.

Next you will need to add an entry into the Info.plist file that is required by the Square SDK and Xcode to protect against malicious URLs. In the Custom iOS Target Properties panel, add a new key named **LSApplicationQueriesSchemes** by tapping the ⊕ button that appears when you hover your mouse in any row. In the Type column, select **Array**. Press the chevron to the left of the key and press the ⊕ button to add **Item 0** to the array. In the string field, enter **square-commerce-v1**

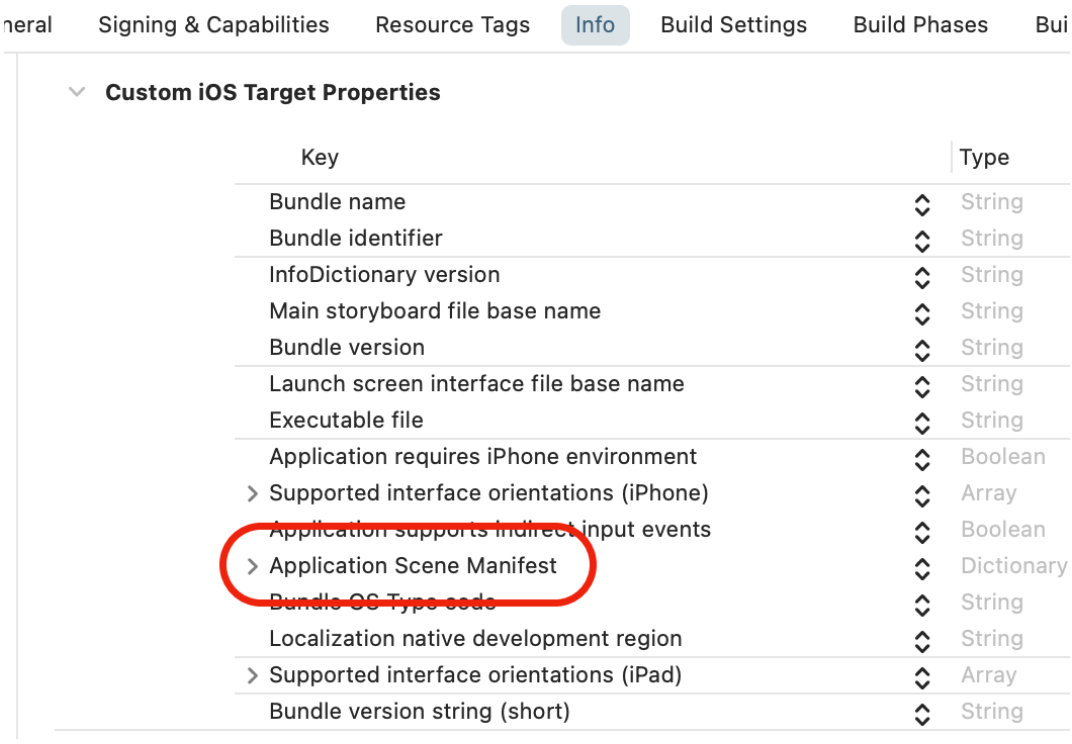


Confirm that your edits match the image above.

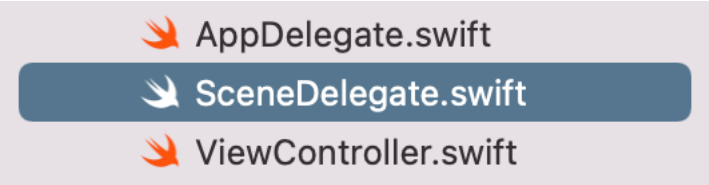
Square documentation parity

The Square documentation is a bit out of date and does not work well with the current application architecture used in iOS projects. The steps below will configure your project to have parity with Square's application design. For purposes of this demonstration project, it will work well enough, but when you build a project for release to the App Store, you should use the latest application architecture. Though the app built here will work reliably, it will give you challenges in designing a user interface that works well with various iOS devices.

Inside of the Info tab and in the Custom iOS Target Properties panel, find the Application Scene Manifest key. Place your mouse cursor in the row and click on the ⊖ button to delete the key/value pair.



In the Project Navigator, select the SeleneDelegate.swift file and delete it.



Select the AppDelegate.swift file and inside the editor, delete these functions:

```
func application(_ application: UIApplication,
   configurationForConnecting connectingSceneSession:
   UISceneSession, options: UIScene.ConnectionOptions) ->
   UISceneConfiguration

func application(_ application: UIApplication,
   didDiscardSceneSessions sceneSessions: Set<UISceneSession>)
```

Add this property to the AppDelegate class in the same file:

```
var window: UIWindow?
```

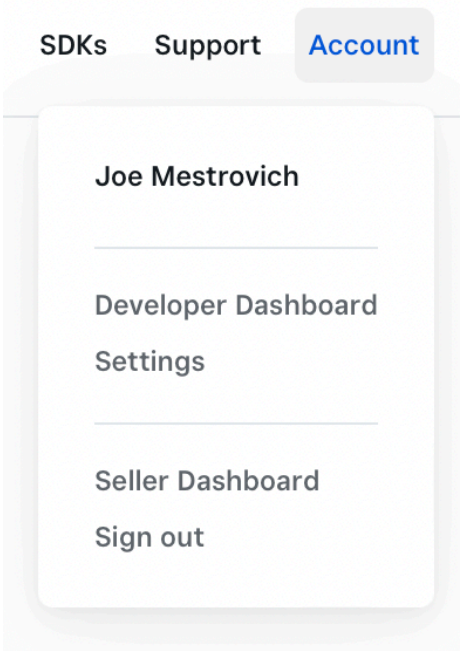
The AppDelegate class should look like this:

```
import UIKit

@main
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?

    func application(_ application: UIApplication,
      didFinishLaunchingWithOptions launchOptions:
      [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
      // Override point for customization after application launch.
      return true
    }

}
```

Create a project ID with Square

The Square payment processing servers need details from your app so that payments are accurately credited to your account. Begin by registering an app project with Square at https://developer.squareup.com. If necessary, create a developer account with Square.

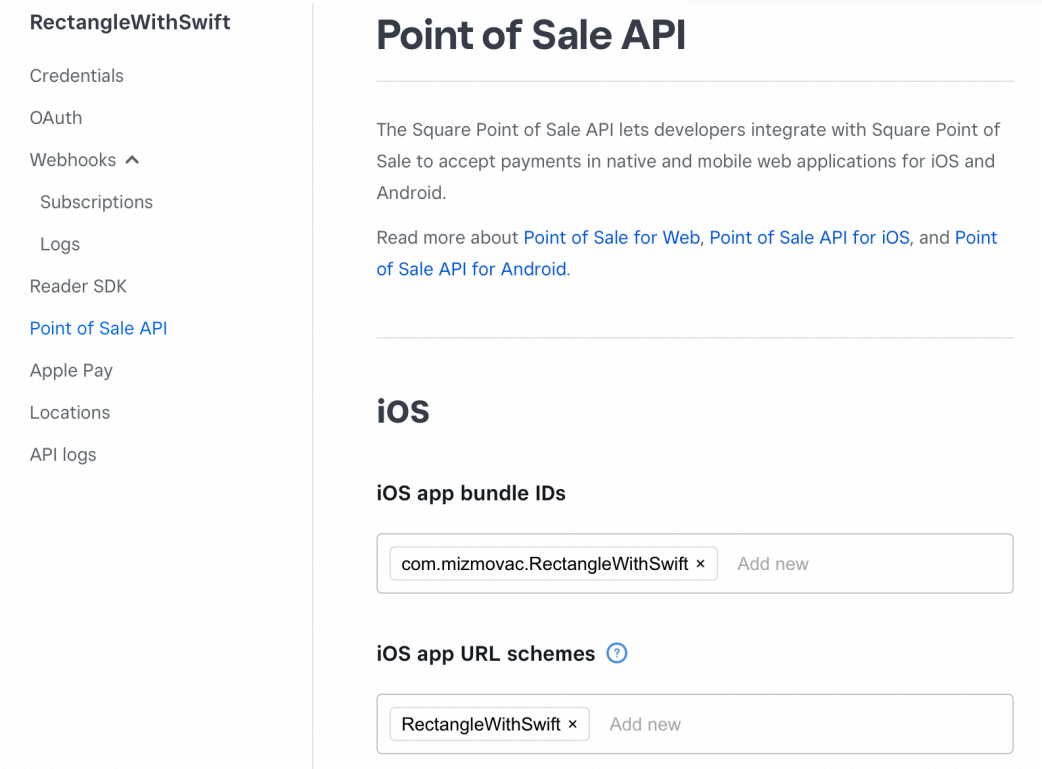
From the Account menu on the website, select Developer Dashboard. Press the gray ⊕ button to register your application with Square.



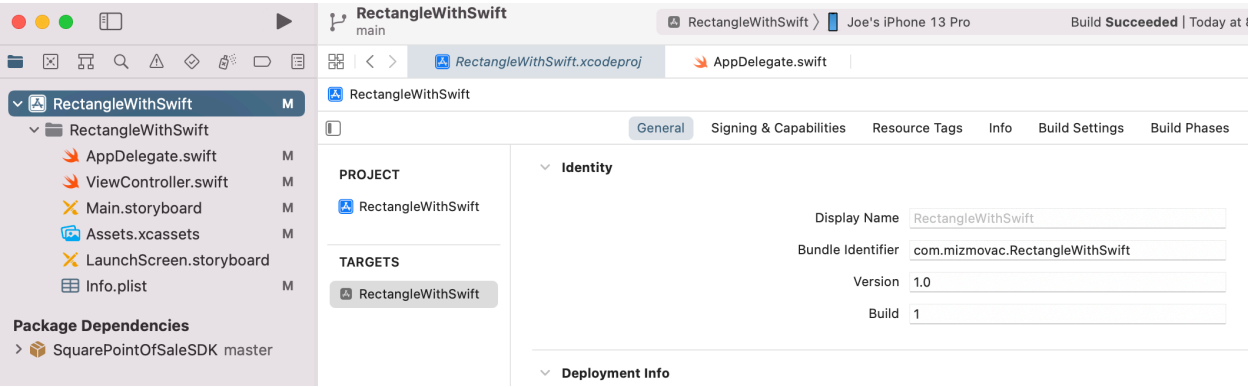
Supply Square with your application name and press Save to request a Production Application ID. Copy this ID for use with your app, but keep it secure. During the hackathon you may share your code with others, so you must be careful to not expose this ID. Consider storing your ID outside of the controller that will use it.

Be especially careful to not display your Production Access token. Though your app will not need it function, it is required for some administrative work and full API access. In either case, it will not be required for most work with the POS SDK.

Next, click on the Point of Sale API menu option on the right hand side of the web page.



You will need to supply Square with the Bundle ID and URL Scheme for your app. In Xcode, select the project root in the Project Navigator. Select the General tab from the Project Attributes window.



Copy the reversed DNS from the Bundle Identifier field. Paste it into the iOS app bundle IDs field on the Square web page.

Next, click on the Info tab in the window. Make a copy of the URL Schemes name you created earlier. Copy that into the iOS app URL schemes field on the Square web page. Save your work.

Square will use these uniquely identifying IDs to communicate with your app. Make sure that the bundle ID is coupled to your app by building a reversed DNS that is unique only to your app and your Apple developer account.

Code a transaction request to the Square POS

You may want to build a class to manage POS transactions. If your application requires more than one URL path to process transactions, it would be especially helpful to organize requests with a manager. This example assumes there is only one path back to the application.

Begin by importing the kit methods:

```
import SquarePointOfSaleSDK
```

The function below contains all that is minimally required to request a transaction:

```
func requestPayment(
	dollarAmount: Double,
	transactionName: String,
	callbackURL: String,
	applicationID: String)
{

	// Square POS transactions must be converted to cents as an integer
	let cents = Int(dollarAmount * 100)

	// The callbackURL must be a type of URL
	// Pass in the URL Schemes name only; the URL initializer will build a
	// URL for you
	let callback = URL(string: callbackURL + "://")!

	// Pass in the Square Production Application ID
	SCCAPIRequest.setApplicationID(applicationID)

	do {
		// Note the option to handle currency types other than dollars
		let transactionAmount = try SCCMoney(
			amountCents: cents,
			currencyCode: "USD")

		// A request with minimal configuration
		let apiRequest = try SCCAPIRequest(
			callbackURL: callback,
			amount: transactionAmount,
			userInfoString: nil,
			locationID: nil,
			notes: transactionName,
			customerID: nil,
			supportedTenderTypes: .all,
			clearsDefaultFees: false,
			returnsAutomaticallyAfterPayment: false,
			disablesKeyedInCardEntry: false,
			skipsReceipt: false)

		// Make the request
		try SCCAPIConnection.perform(apiRequest)

	} catch let error as NSError {
		print(error.localizedDescription)
	}
}
```

Notice several important details in the code above:

- Currencies must be converted to cents and of type integer. This helps avoid real number precision issues.
- The callback URL must be a type of URL. The Swift URL type has an initializer that can build a URL for you.
- You must supply the Square Application ID for each transaction.

There are many other available options that you can use to customize the transaction. Visit https://developer.squareup.com/docs/api/point-of-sale#navsection-point-of-sale to learn more.

Build an application delegate to handle the Square POS response

The Square POS app will attempt to restore control back to your app using the URL you provided in the transaction request. A simple way to manage the callback is to use the existing AppDelegate class that is built with your project.

Select the AppDelegate.swift file from the Project Navigator. Add the import statement for the Square SDK just below the UIKit.

```
import SquarePointOfSaleSDK
```

The UIApplicationDelegate contains the method application(_:open:options:) that you can use to handle the incoming URL from Square. The general form of the function should look like this:

```
func application(
	_ app: UIApplication,
	open url: URL,
	options: [UIApplication.OpenURLOptionsKey : Any] = [:]) -> Bool
{

	// Verify that the response is from the Square app
	guard SCCAPIResponse.isSquareResponse(url) else {
		return false
	}

	do {
		// Build a SCCAPIResponse object for error handling and
		// passing transaction data
		let response = try SCCAPIResponse(responseURL: url)

		if let error = response.error {
			// Handle a failed request.
			print(error.localizedDescription)
		} else {
			// Do something interesting with the response like
			// adjust inventory or track customer purchases
			print("The request was a success!")
		}

	} catch let error as NSError {
		// Handle unexpected errors.
		print(error.localizedDescription)
	}

	return true
}
```

Building your own app

The transaction request code and the application delegate (along with the installation of the Square POS app) are all you need to tender sales. The sample app contains an implementation of this with some minor modifications. Use it to get a sense of how to construct a working POS application.

The app also includes connectivity to the Brother QL-1110NWB printer over WiFi. Read the BrotherLabelsInWiFi tutorial for a primer on how to use the printer with your own app.

Please take note of these key items and issues:

- The callback URL and Square Application ID are constants in the app's view controller. This is not a good practice for a production app and will lead to your bank account getting emptied.
- The Brother label printer uses a WiFi network connection. You must select a networked printer from the list in the app before printing.
- The sample app cannot run inside of a simulator. The Brother and Square SDKs have conflicting requirements for running within a simulator.
- When running the app, wait a few seconds for the list of networked printers to appear. Also, the Square reader may take a couple of seconds to initialize.
- The Square POS does not run in a sandboxed environment. If you make a credit card transaction, visit the Square account management site to refund the charge.
- The receipt is a PDF document. The formatting example is also not production ready and can be much improved. Please use the example as inspiration for something that you make more robust.
- Before printing, notice that the printReceipt() function uses Brother 54mm continuous label paper. You will probably need to change that to the stock you have at hand. Also, be sure to adjust the page width and height for the PDF renderer.
- The example strips Apple's Scene handling in order to make it compatible with Square's documentation. For your own app, consider using Scenes.
- To keep the example app simple it does not use multithreading. For your own apps you will not want to block the main thread.
- For best results, use the Xcode Package Manager to install the Square SDK. The Square documentation does not reference it and suggests using Pods. Ignore that.