

# Apple Pay Quick Start Guide

March 2018

# Contents

<b>Overview .....</b>	<b>3</b>
Target Audience .....	3
Minimum Technical Requirements .....	3
Related Documents and Resources .....	3
<i>Third Party Resources</i> .....	3
<b>Register Your Free Developer Account .....</b>	<b>4</b>
<b>Apple Developer Portal Configuration .....</b>	<b>11</b>
<b>Download The Payeezy Apple Pay SDK .....</b>	<b>13</b>
Integrate The SDK Into Your App .....	13
Add Our Frameworks To Your Project .....	13
About the Example Application .....	17
<i>Create the Storyboard for Our Example</i> .....	17
<i>Authorize / Payment Control</i> .....	18
<i>Amount Label and Input Text Field</i> .....	20
<i>Pay Button</i> .....	26
<i>Create Outlets and an Action for Our View Controller</i> .....	28
<i>Design Considerations</i> .....	30
<i>The FDPaymentAuthorizationViewControllerDelegate Methods</i> .....	40
<i>Managing Entitlements</i> .....	42
<b>Appendix .....</b>	<b>44</b>
Document Edit History .....	44

# First Data Apple Pay Quick Start Guide

## Overview

This guide is designed to get you up and running to enable secure and convenient in-app payments in your iOS app. This API from First Data was created to simplify your integration with Apple Pay™. Payeezy handles all the heavy lifting of the complex cryptography that protects your customers' transactions. It also makes it super simple to create a developer test account and even apply for a merchant account all through our developer portal.

## Target Audience

The target audience for this document is a developer who wants to use Apple Pay in their payment application.

## Minimum Technical Requirements

Developers wishing to use the First Data Apple Pay (In App) sample application will need the following software and hardware:

- Xcode 6.3 and above
- iOS 8.0 and above
- A physical device or an emulator to use for developing and testing. Refer to [Apple Pay](#) for compatible devices.

## Related Documents and Resources

The following First Data documents and resources provide supporting information for this document:

- [Developer Portal](https://developer.payeezy.com/) (<https://developer.payeezy.com/>) - This portal is where developers register and are boarded to First Data's RESTful API. It also contains specifications for all supported APIs, sample requests and responses, and development guides. The following items are especially pertinent to readers of this guide:
  - [Apple Pay Transaction API](#)
  - [Getting Started](#)
  - [FAQs](#)
- [First Data Apple Pay Example](#) – This sample application shows how to integrate the First Data APIs to support the Apple Pay feature.

## Third Party Resources

In addition to the First Data resources, the following third party documents and resources provide supporting information for working with the First Data APIs and sample Apple Pay application.

### Apple Pay Resources

Developers should refer to the Apple Pay documentation, available at the [Apple Pay Developer](#) site, particularly the following:

- Getting Started with Apple Pay
- iOS Human Interface Guidelines: Apple Pay
- Apple Pay Programming Guide
- PKPaymentAuthorizationViewController Class Reference
- Apple Pay Within Apps
- Apple Pay Identity Guidelines
- AppReview Guidelines: Apple Pay
- Apple Pay Button and Resources

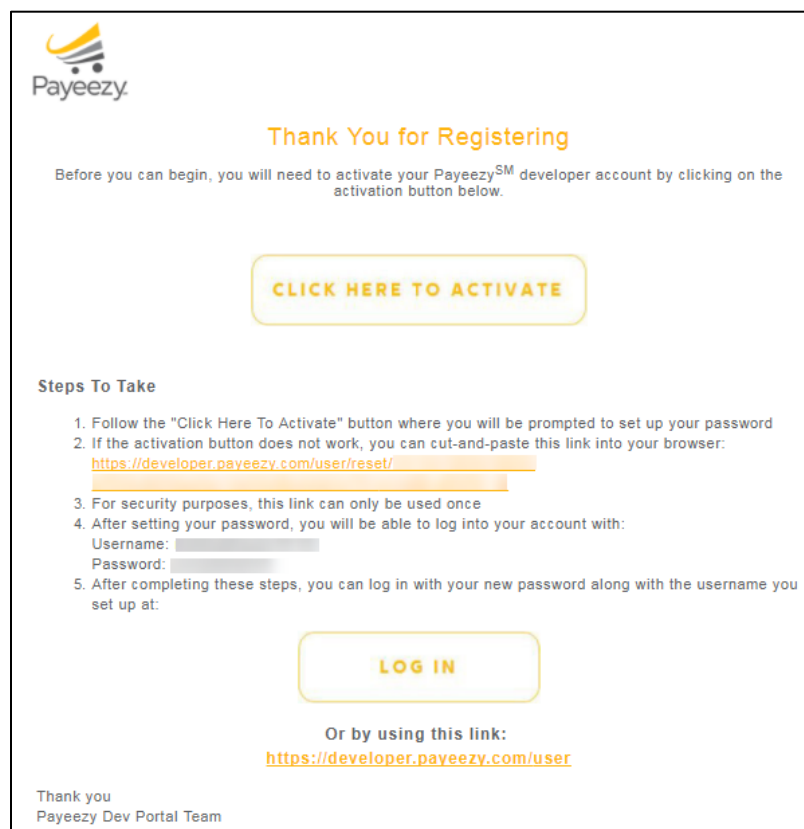
### Download and Build Resources

- [GitHub](https://github.com/) (https://github.com/) – Recommended for First Data sample application code downloads
- [Gradle](http://gradle.org/) (http://gradle.org/) – For builds, the standard Gradle build can be used.

## Register Your Free Developer Account

Once you register for your own free account, First Data sets up your own personal sandbox so you can start integrating and testing Apple Pay transactions in seconds. All you will need to give us is your name and a valid email address.

After you provide registration information an email is sent to you with further instructions. Here's an example.



Be sure to complete your registration from the link, username and password provided in the email. When you process the initial registration you will be forced to select a new, more secure, password for your account. From this point you will be prompted for security questions that can be used if your password is forgotten.

The next step is to have you create an API project on the Developer Portal.

Think of this as a unique app name that is used to identify your application. The name is unrelated to the name you would use on Apple's App Store but it can be the same. Click the **Get Certified** tab and the following screen is shown.

## First Data Apple Pay Quick Start Guide

My APIs

Add more, edit or delete them as you like.

Click here to create an API project - on completion of this step, you will have an API Key and a Secret.

+ ADD A NEW API

Step 2 > Get your Merchant Token

Test

SANDBOX

Click **ADD A NEW API**.

On the next screen you name your application.

CONSUME OUR API

NAME YOUR APPLICATION : \*

My First Payeezy App

Internal name: my-first-payeezy-app Edit

WHAT TYPE OF PRODUCT IS THIS ? : \*

☐ Live

☒ Sandbox

CREATE YOUR APP

Spaces are allowed within the app name. For this example we are using *My First Payeezy App*. The checkbox **Sandbox** must be selected at this time. This will allow you to test your app in a secure environment provided by First Data.

Click **CREATE YOUR APP** to continue.

Once your app is created, click the name to display detailed information associated with the app.

**new My First Payeezy App** SANDBOX

KEYS PRODUCTS DELETE "MY FIRST PAYEEZY APP" EDIT "MY FIRST PAYEEZY APP"

ANALYTICS

**My First Payeezy App's Keys**

Use the API Key (below) and the Token (see the "My Merchants" page ) to execute an API call. The API Secret(below) is your HMAC key.Refer to our API Documentation and FAQ sections for more details.

API Key	8s4NVGcw6WaOrGm9ASF1ukzw5IAaLopD
API Secret	6282947b5e13d10176fa7c08823d4874be8f56003fbc005c68c9930bd3d630c6
Reporting Token	2dc0092ced3b578c

The **KEYS** button is selected by default and shows the API Key and API Secret that you will need later.

Click **PRODUCTS** button to confirm that your app is approved to be tested within the sandbox.

**new My First Payeezy App** SANDBOX

KEYS **PRODUCTS** DELETE "MY FIRST PAYEEZY APP" EDIT "MY FIRST PAYEEZY APP"

ANALYTICS

API Product: Sandbox

Status Approved

Next, you need a merchant token. To get the merchant token you need to complete the certification process as follows.

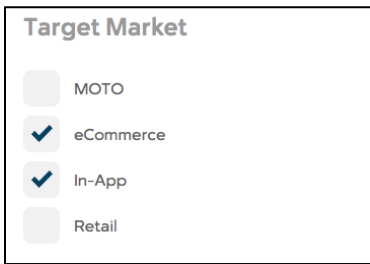
Click on the Get Certified button.



Fill in the fields on the next page that opens up. Some of the items will be pre-filled for you. The phone number you provide needs to be real but does not need to be one that can receive SMS messages.

## First Data Apple Pay Quick Start Guide

When you get to the section Target Market,



**Target Market**

☐ MOTO

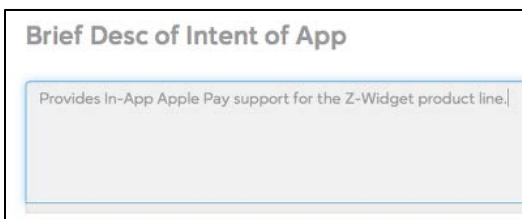
☒ eCommerce

☒ In-App

☐ Retail

Be sure to check the boxes for eCommerce & In-App.

Add a brief description about your app.



**Brief Desc of Intent of App**

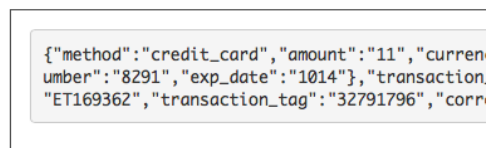
Provides In-App Apple Pay support for the Z-Widget product line.

The web page also advises you to ensure there is properly formatted content filled in for the Certification Test data to ensure that you are not a bot. The example cert will work fine for testing.

Click **CERTIFY**.

Within a few seconds you should see a JSON success response.

**Response : Success**



```
{ "method": "credit_card", "amount": "11", "current_number": "8291", "exp_date": "1014", "transaction_id": "ET169362", "transaction_tag": "32791796", "correlation_id": "1234567890" }
```

You can now select the **I Agree** checkbox for the terms and conditions and press SUBMIT.

You will now be taken to the ADD MERCHANT page.

## Add A Merchant

Choose one:

☒ Sign up for a merchant account

**SUBMIT** Cancel

Click **Submit**. On the next screen, you'll be asked to indicate if you're a merchant or the developer working with the merchant.

## Add Merchant

Are you the Merchant ?

☒ Yes

☐ No, I'm adding other merchants

**SUBMIT** Cancel

Select the appropriate response:

- Yes - You are your own merchant and will be directly accepting payments. If you are a merchant in this scenario, only you will be looking at transactions. You will need to have valid tax information if you select this option.
- No, I'm adding other merchants - You are a developer working at a merchant or are a developer hired to do this work for a merchant.

For this example, we are selecting No. Click **SUBMIT**. The Notify Merchant page is displayed for you to provide the information to notify the merchant of your boarding process.



## First Data Apple Pay Quick Start Guide

### Notify Merchant

Email \*


Company Name \*

Contact Person \*

Phone Number \*

☐ In App Payments

☐ I'm not a robot

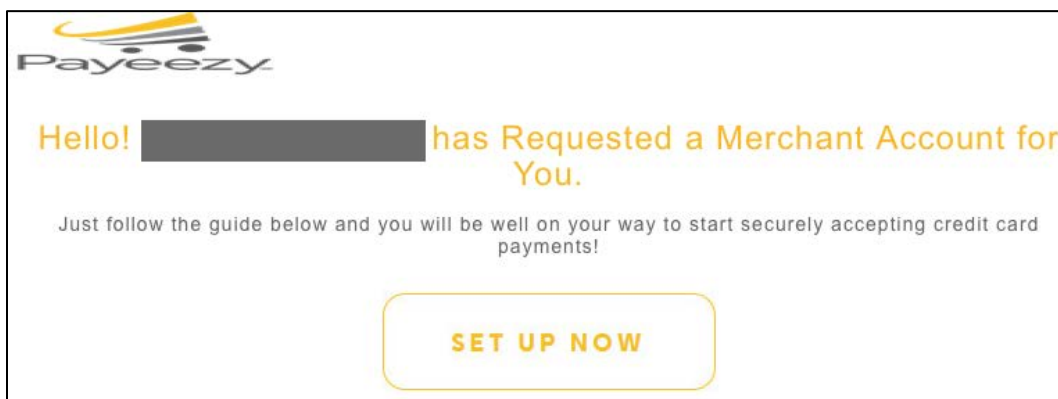
  
reCAPTCHA  
[Privacy](#) - [Terms](#)

NOTIFY MERCHANT

Required fields are marked with an asterisk. In order to enable Apple Pay payments, you must select the In App Payments checkbox. Then, click the ReCaptcha box and click **NOTIFY MERCHANT**.

A message notifying you that an email has been sent to the merchant is presented.

The merchant will receive an email asking them to authorize the next step.



The merchant will be taken through steps where they can specify their tax and banking information as part of the authorization and boarding process.

Until your merchant is validated and boarded, you can work with the test merchant. Click the Merchant tab, and make sure SANDBOX is selected (it is the default.)

## Merchants

**SANDBOX** **LIVE**

+ Add a Demo Merchant

Merchant ID	Merchant Name	Token	JS Security Key	View Details	Status
3176752955	Acme Sock	fdoa-433eb55143320e9d1637c226daf13446433eb55143320e9d	js-8cb386b67c4f80d08f04a419510425e08cb386b67c4f80d0	View Details	Approved

You should provide this merchant token as the token header parameter in your API call.

For integrating with Apple Pay, you must generate a Certificate Signing Request (CSR). To generate this file, click the Certs tab at the top-right side of the screen.

## Certs

**SANDBOX** **LIVE**

Add Certs

APPLE\_PAY

Enter App Label \*

My First Payeezy App

+ ADD

What's this ?

Cert Type	CSR / Public Key	App Label	Public Key Hash	Status
APPLE_PAY	Download	merchant.com.testaow	67S4fZbWcQTILRhQH+R00I1RYJ9ZROZu6OS3zH6CtWk=	ACTIVE

On the Certs page, select APPLE\_PAY from the Add Certs drop-down list. Enter your App Label, and click **ADD**. Once the file is generated, click **Download**.

Note that the CSR is only available for your download when your app status shows it is approved.

### Apple Developer Portal Configuration

You now need to logon to the Apple Developer Portal through your Apple account.

Select **Certificates, Identifiers & Profiles** on the Apple Developer Portal.

Continue to the **Identifiers** section, and then select **Merchant IDs**.

Click **+** to add a new Merchant ID. On the following page, enter the Merchant ID Description and Identifier. The Merchant ID Description should be the same as the App ID on the First Data Developer Portal. The identifier you specify should follow the same reverse name domain scheme used for other app ids. It does not need to agree with your Bundle ID, however it needs to be unique.

#### Merchant ID Description

Description:

You cannot use special characters such as @, &, \*, ', "

#### Identifier

Enter a unique identifier for your Merchant ID, starting with the string 'merchant'.

ID:

We recommend using a reverse-domain name style string (i.e., merchant.com.example.merchantname).

Click **Continue**.

Review and verify the information, and then click **Register**.

Apple processes your registration and presents you with confirmation that your registration of Merchant ID is complete. Click **Done**.

The newly registered Merchant ID is shown in a list with any other Merchant IDs you have defined. You can select your new Merchant ID from the list and see its associated details.

My First Payeezy Merchant

merchant.com.firstpayeezy

ID

Name: My First Payeezy Merchant

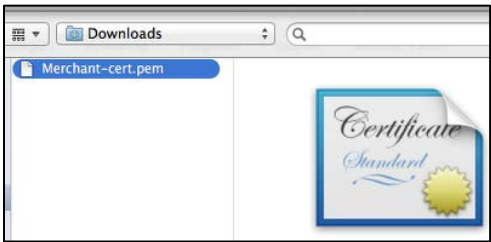
ID: merchant.com.firstpayeezy

Edit

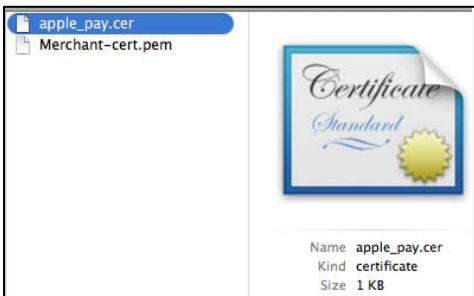
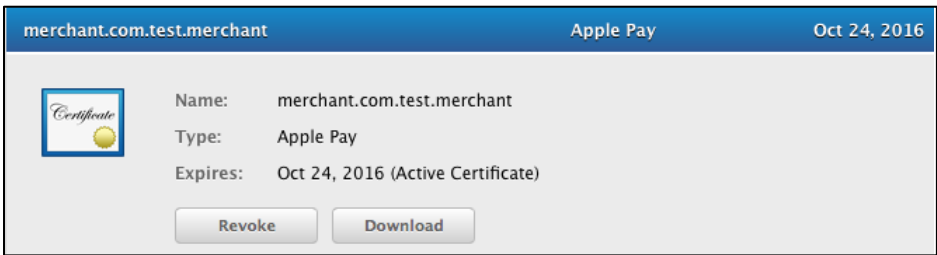
Delete

Click **Edit** to begin the the Certificate Creation process. On the next screen click **Create Certificate**. At this point, you will be uploading the CSR you generated on the First Data Developer Portal.

Navigate to your Merchant-cert.pem file's location and select it.



The Generate button will become active. Click it to generate your Apple Pay certificate. Once it has been generated, click **Download** to save it.



## First Data Apple Pay Quick Start Guide

### Download The Payeezy Apple Pay SDK

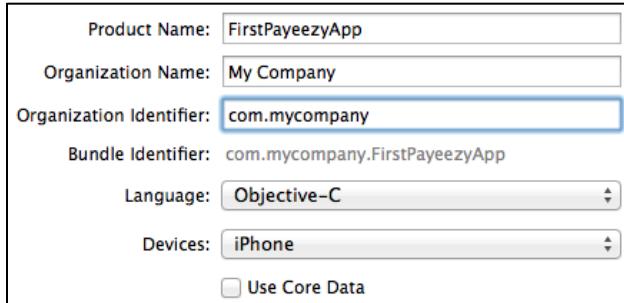
Download our Apple Pay SDK from GitHub. Decompress the downloaded .zip file and store the included frameworks in a folder that you can reference from your app.

### Integrate The SDK Into Your App

First make sure that you've installed Xcode version 6.0 or greater so you can access the new Apple Pay API's. If you're not sure which version you have, download the latest version from the Apple Developer site or directly from the App Store.

Create a new project in Xcode. Use Single View Application as your template. We will be using Objective-C for our programming language.

For this example, the following screen capture shows how we filled out the project options sheet

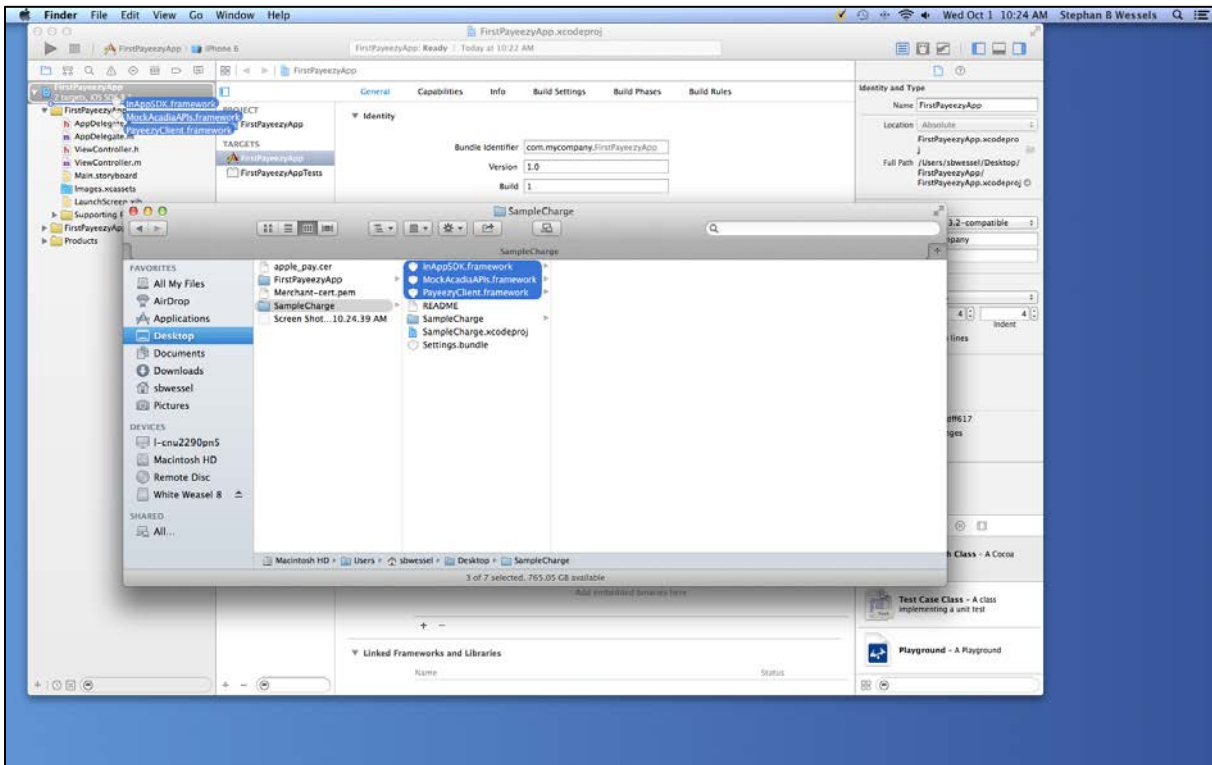


The screenshot shows the 'New Project' dialog in Xcode, specifically the 'Options' tab for a 'Single View Application'. The fields are filled as follows:

- Product Name: FirstPayeezyApp
- Organization Name: My Company
- Organization Identifier: com.mycompany (highlighted with a blue border)
- Bundle Identifier: com.mycompany.FirstPayeezyApp
- Language: Objective-C (selected from a dropdown menu)
- Devices: iPhone (selected from a dropdown menu)
- Use Core Data: ☐ (unchecked)

### Add Our Frameworks To Your Project

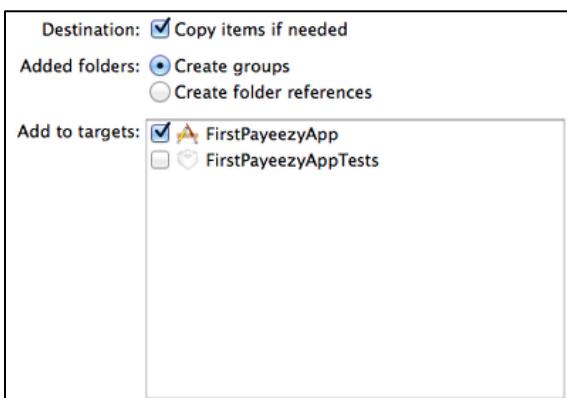
With your project opened in Xcode and a Finder window opened to the folder containing the frameworks you downloaded in Step 2, drag and drop the frameworks onto your project.



There are 3 frameworks included here:

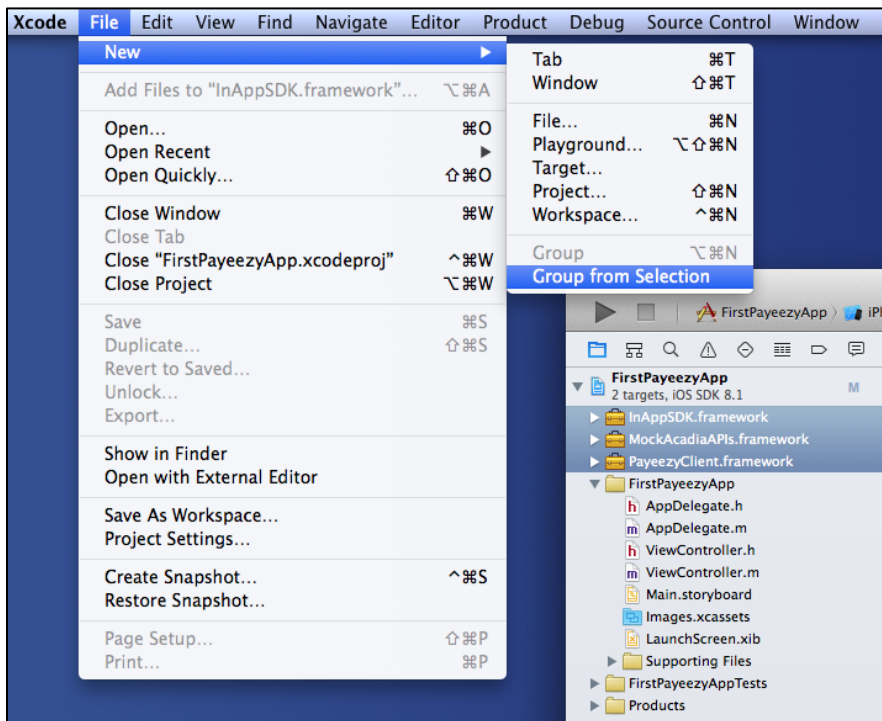
- **InAppSDK.framework** This is the main framework that supports ApplePay transactions.
- **PayeezyClient.framework** This is the iOS client for the Payeezy (RESTful) API backend services. The InAppSDK framework relies on this client for making secure service calls to the services. Visit <http://developer.payeezy.com> for documentation and an interactive sandbox for experimenting with the Payeezy API's.
- **MockAcadiaAPIs.framework** This framework contains mock versions of the ApplePay API's. Its purpose is to allow developers to start integrating In-App payments without having access to the ApplePay API's and/or an iPhone 6. When we are ready to deploy on a real Apple Pay™ enabled iPhone we will no longer include this framework.

When prompted be sure to select the **Copy items if needed** option.

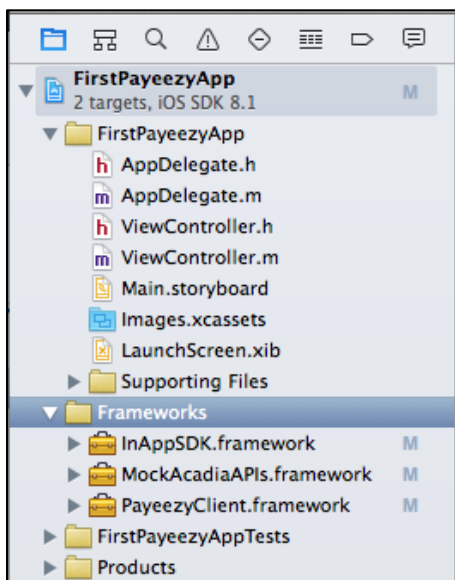


## First Data Apple Pay Quick Start Guide

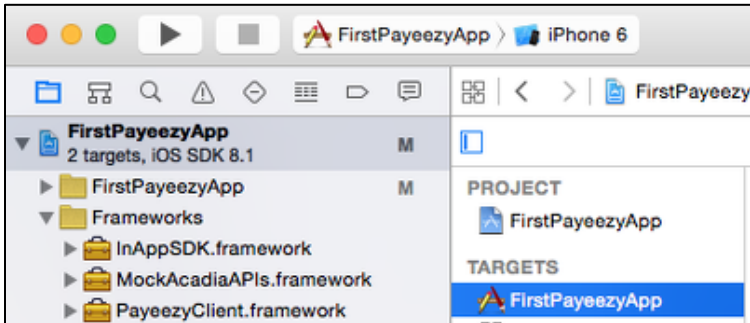
If you prefer to use a frameworks group when working with Xcode projects, it is a simple matter to select the frameworks and create a new Group.



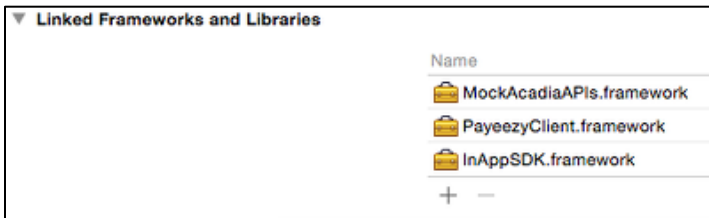
Name the new group Frameworks and drag it to its new location within the Project window.



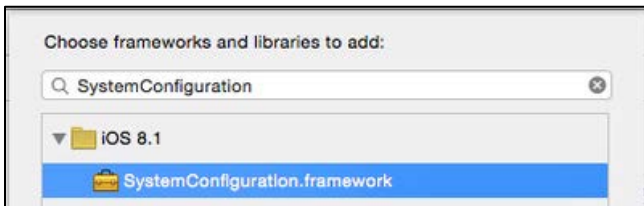
We will also need to add the SystemConfiguration.framework from Apple's available frameworks. The simplest way to do this is to first select the target.



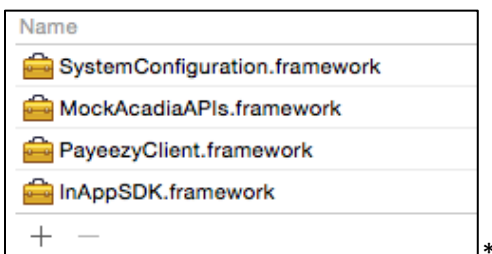
Under the General tab find the Linked Frameworks and Libraries section.



When you click on the + button Xcode will prompt you with a list of frameworks to choose. You can narrow down your search by typing in SystemConfiguration as a filter.



Click the **Add** button and Xcode will include the new framework.





## First Data Apple Pay Quick Start Guide

### About the Example Application

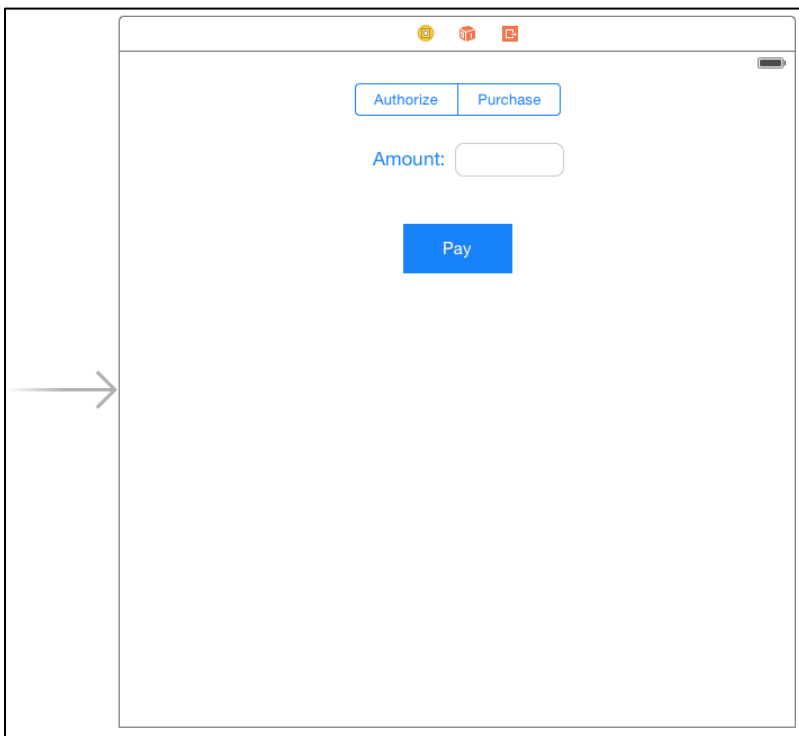
We will create an Application that demonstrates how easy it is to enable secure In-App payments via Apple's Apple Pay technology. Apple collaborated with First Data to be one of the first payment platforms that is certified to support this revolutionary payment method.

The application itself contains a single view controller that allows the user to specify a transaction amount and select whether to perform a pre-authorization for that amount or an instantaneous purchase. That transaction is routed through the First Data In-App payments SDK to perform an Apple Pay transaction.

We will create the Storyboard contents next, then discuss and add the model objects you will need. Finally we will wire up the models and messages to your Storyboard.

### Create the Storyboard for Our Example

The Storyboard we want to create will look like this.



There are 4 GUI elements to add to the Storyboard. We'll also use the default **Any/Any** size so we can design an app that will work well on the support iPhone 6 & 6 Plus (as well as any potential other future supported devices).

## Authorize / Payment Control

Drag a Segmented Control onto the Storyboard and locate it near the top of the view and roughly centered. First we'll set the properties we need and then define the location constraints.

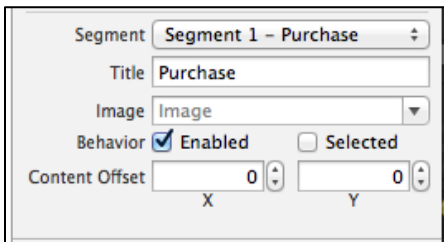
Set the Properties view for the control as follows.

The screenshot shows the Xcode Properties view for a Segmented Control. The view is divided into three main sections: Segmented Control, Control, and View.

- Segmented Control**
  - Style: Plain
  - State: ☐ Momentary
  - Segments: 2
  - Segment: Segment 0 - Authorize
  - Title: Authorize
  - Image: Image
  - Behavior: ☒ Enabled, ☒ Selected
  - Content Offset: X: 0, Y: 0
- Control**
  - Alignment: Horizontal (selected), Vertical
  - Content: ☐ Selected, ☒ Enabled, ☐ Highlighted
- View**
  - Mode: Scale To Fill
  - Tag: 0
  - Interaction: ☒ User Interaction Enabled, ☐ Multiple Touch
  - Alpha: 1
  - Background: Default
  - Tint: Default
  - Drawing: ☐ Opaque, ☐ Hidden, ☒ Clears Graphics Context, ☐ Clip Subviews, ☒ Autoresize Subviews
  - Stretching: X: 0, Y: 0, Width: 1, Height: 1
  - ☒ Installed

## First Data Apple Pay Quick Start Guide

The details for segment–0, **Authorize**, are as shown above. Select the settings for segment–1, **Purchase**, as follows.



Segment: Segment 1 – Purchase

Title: Purchase

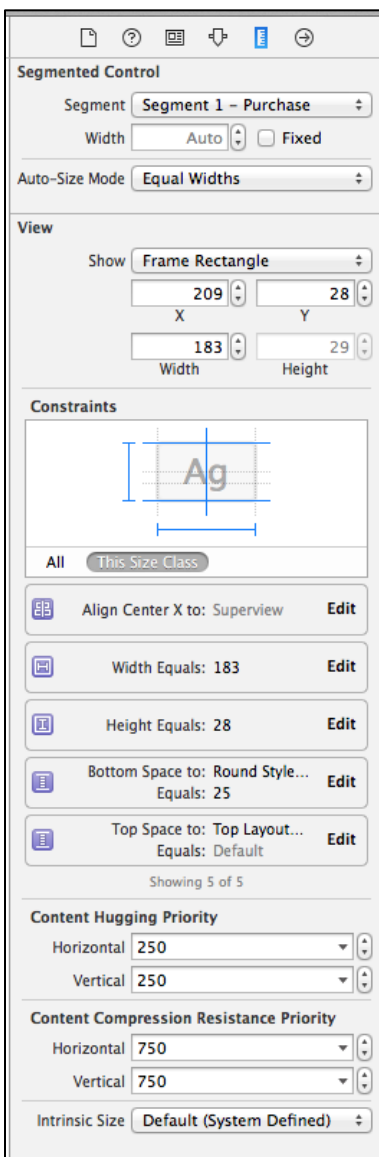
Image: Image

Behavior: ☒ Enabled ☐ Selected

Content Offset: X: 0 Y: 0

Next we must set the constraints for our segment control. The concept we are aiming for is to have the segment control to be centered horizontally inside the view and constrained against the top edge of the view. This will make it work regardless of orientation or size of iPhone.

The constraints look as follows.



Segmented Control

Segment: Segment 1 – Purchase

Width: Auto ☐ Fixed

Auto-Size Mode: Equal Widths

View

Show: Frame Rectangle

X: 209 Y: 28

Width: 183 Height: 29

Constraints

All This Size Class

- Align Center X to: Superview Edit
- Width Equals: 183 Edit
- Height Equals: 28 Edit
- Bottom Space to: Round Style... Equals: 25 Edit
- Top Space to: Top Layout... Equals: Default Edit

Showing 5 of 5

Content Hugging Priority

Horizontal: 250

Vertical: 250

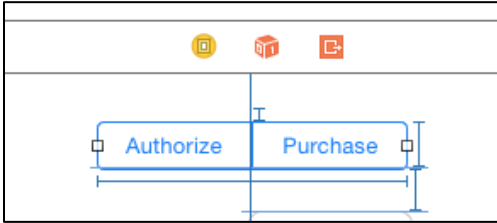
Content Compression Resistance Priority

Horizontal: 750

Vertical: 750

Intrinsic Size: Default (System Defined)

Note that we have fixed the width and height of the control and aligned it to the center of the super-view. The Top Layout constraint defines how close we get to the top of the view. The Bottom Space constraint has no meaning until we add an object below it. If you have only defined the segmented control so far you will not see that constraint.



We will go back and deal with state and events, if needed, for this control later.

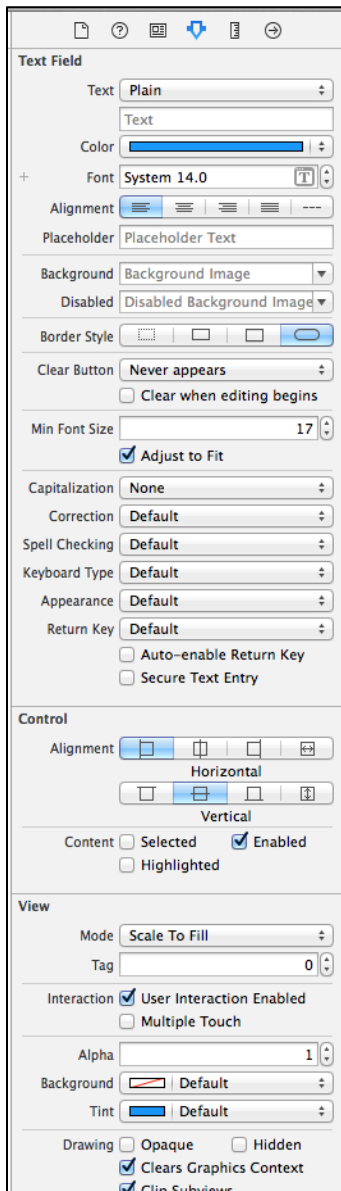
### Amount Label and Input Text Field

Our next two visual objects will be the input text field and a companion label. We'll add the input field first.

Drag a Text Field object onto the view and locate it below the segmented control and slightly right-of-center in the view. We'll get the properties right for it first then fix the constraints.

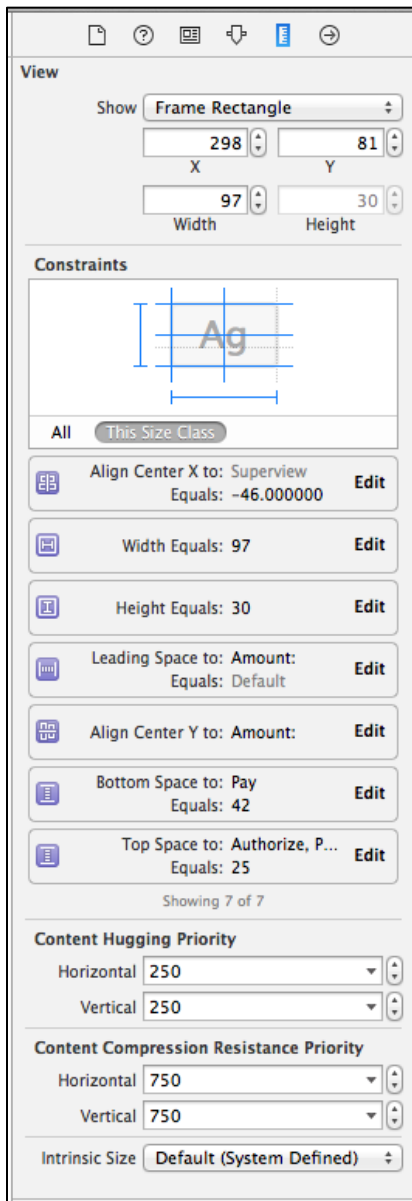
## First Data Apple Pay Quick Start Guide

Here is the properties view.



When defining the constraints we seek to align the object just below the segment and will use the horizontal center of the view as a reference and specify an X-offset.

Here is the constraints & sizing information.



The constraints for Leading Space to Amount and Align Center Y to Amount as well as the Bottom Space constraints are not relevant until we add the label object.

Drag a label to the left of the input text. The properties are as follows.

## First Data Apple Pay Quick Start Guide

The image shows the Xcode Interface Builder settings for a UILabel and a UIView. The UILabel section includes options for Text (Plain), Color (blue), Font (System 17.0), Alignment (left), Lines (1), Behavior (Enabled), Baseline (Align Baselines), Line Breaks (Truncate Tail), Autoshrink (Fixed Font Size), Highlighted (black), Shadow (red), and Shadow Offset (0 horizontal, -1 vertical). The UIView section includes options for Mode (Left), Tag (0), Interaction (User Interaction Enabled, Multiple Touch), Alpha (1), Background (red), Tint (blue), Drawing (Opaque, Hidden, Clears Graphics Context, Clip Subviews, Autorelease Subviews), Stretching (0 X, 0 Y, 1 Width, 1 Height), and Installed (checked).

**Label**

Text: Plain

Amount:

Color: [Blue]

Font: System 17.0

Alignment: [Left]

Lines: 1

Behavior: ☒ Enabled  
☐ Highlighted

Baseline: Align Baselines

Line Breaks: Truncate Tail

Autoshrink: Fixed Font Size  
☐ Tighten Letter Spacing

Highlighted: [Black] Default

Shadow: [Red] Default

Shadow Offset: 0 Horizontal, -1 Vertical

**View**

Mode: Left

Tag: 0

Interaction: ☐ User Interaction Enabled  
☐ Multiple Touch

Alpha: 1

Background: [Red] Default

Tint: [Blue] Default

Drawing: ☐ Opaque ☐ Hidden  
☒ Clears Graphics Context  
☐ Clip Subviews  
☒ Autorelease Subviews

Stretching: 0 X, 0 Y, 1 Width, 1 Height

☒ Installed

For the constraints we will locate this object relative to the input text field. Here are the sizing and constraint values.

The screenshot shows the Xcode Interface Builder inspector for a UILabel widget. The 'Label' section has 'Preferred Width' set to 'Automatic' and 'Explicit' unchecked. The 'View' section has 'Show' set to 'Frame Rectangle'. The 'Constraints' section shows a diagram of the label with dimensions 225x85 and 65x21. Below the diagram are four constraints: 'Width Equals: 65', 'Height Equals: 21', 'Trailing Space to: Round Style... Equals: Default', and 'Align Center Y to: Round Style...'. The 'Content Hugging Priority' section has 'Horizontal' and 'Vertical' both set to 251. The 'Content Compression Resistance Priority' section has 'Horizontal' and 'Vertical' both set to 750. The 'Intrinsic Size' is set to 'Default (System Defined)'.

**Label**  
Preferred Width: Automatic ☐ Explicit

**View**  
Show: Frame Rectangle  
X: 225 Y: 85  
Width: 65 Height: 21

**Constraints**  
All This Size Class

- Width Equals: 65 Edit
- Height Equals: 21 Edit
- Trailing Space to: Round Style... Equals: Default Edit
- Align Center Y to: Round Style... Edit

Showing 4 of 4

**Content Hugging Priority**  
Horizontal: 251  
Vertical: 251

**Content Compression Resistance Priority**  
Horizontal: 750  
Vertical: 750

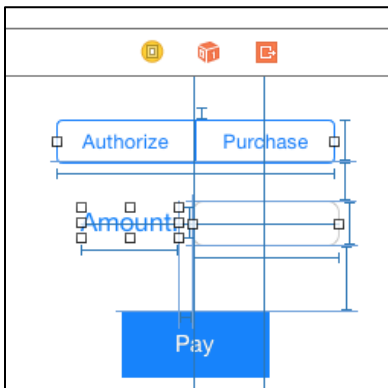
Intrinsic Size: Default (System Defined)

Note that we have specified the size of the widget as well as the space to the right between it and the input field. Select both the input field and label and you can specify the Align Center Y value to neatly align them together.



## First Data Apple Pay Quick Start Guide

The visual elements you have located on your view should show constraints similar to this.



The only exception is that the view shown already has the **Pay** button added. That's next.

## Pay Button

Our last visual component is the **Pay** button. Drag a button onto the view and locate it below the input field and label and center it in the view. Here is the properties pane for the new button.

The image shows a properties pane for a 'Button' component. The pane is divided into several sections: Button, Control, and View. The 'Button' section includes settings for Type (System), State Config (Default), Title (Plain), Text (Pay), Font (System 15.0), Text Color (White Color), Shadow Color (Default), Image (Default Image), Background (Default Background Image), Shadow Offset (0, 0), and various checkboxes for Reverses On Highlight, Shows Touch On Highlight, Highlighted Adjusts Image, and Disabled Adjusts Image. The 'Control' section includes Alignment (Horizontal), Content (Selected, Enabled, Highlighted), and View (Mode: Scale To Fill, Tag: 0). The 'View' section includes Interaction (User Interaction Enabled, Multiple Touch), Alpha (1), Background (blue), Tint (Default), and Drawing (Opaque, Hidden, Clears Graphics Context, Clip Subviews, Autoresize Subviews).

**Button**

Type: System

State Config: Default

Title: Plain

Pay

Font: System 15.0

Text Color: White Color

Shadow Color: Default

Image: Default Image

Background: Default Background Image

Shadow Offset: 0 0

Width Height

☐ Reverses On Highlight

Drawing ☐ Shows Touch On Highlight

☒ Highlighted Adjusts Image

☒ Disabled Adjusts Image

Line Break: Truncate Middle

Edge: Content

Inset: 0 0

Top Bottom

Left Right

**Control**

Alignment: Horizontal

Vertical

Content ☐ Selected ☒ Enabled

☐ Highlighted

**View**

Mode: Scale To Fill

Tag: 0

Interaction ☒ User Interaction Enabled

☐ Multiple Touch

Alpha: 1

Background: [Blue]

Tint: Default

Drawing ☐ Opaque ☐ Hidden

☒ Clears Graphics Context

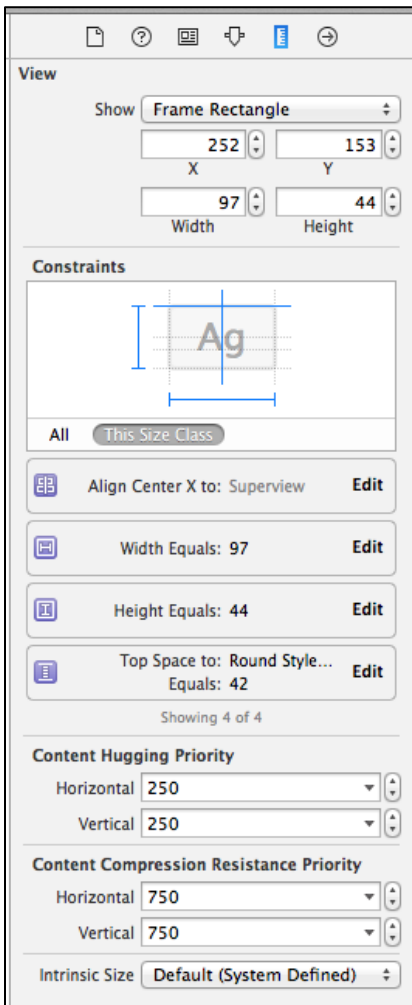
☐ Clip Subviews

☒ Autoresize Subviews

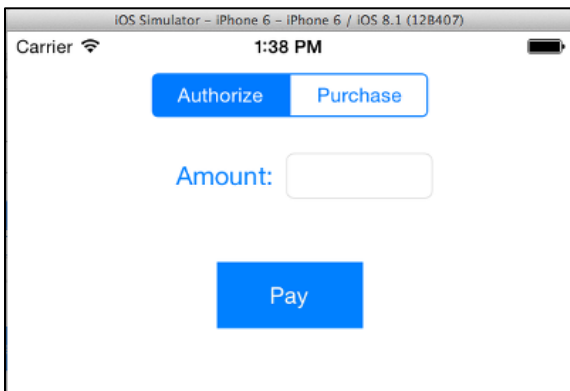
Note that we set the button text to white and the background color to a shade of blue.

## First Data Apple Pay Quick Start Guide

The constraints for our **Pay** button are very simple.



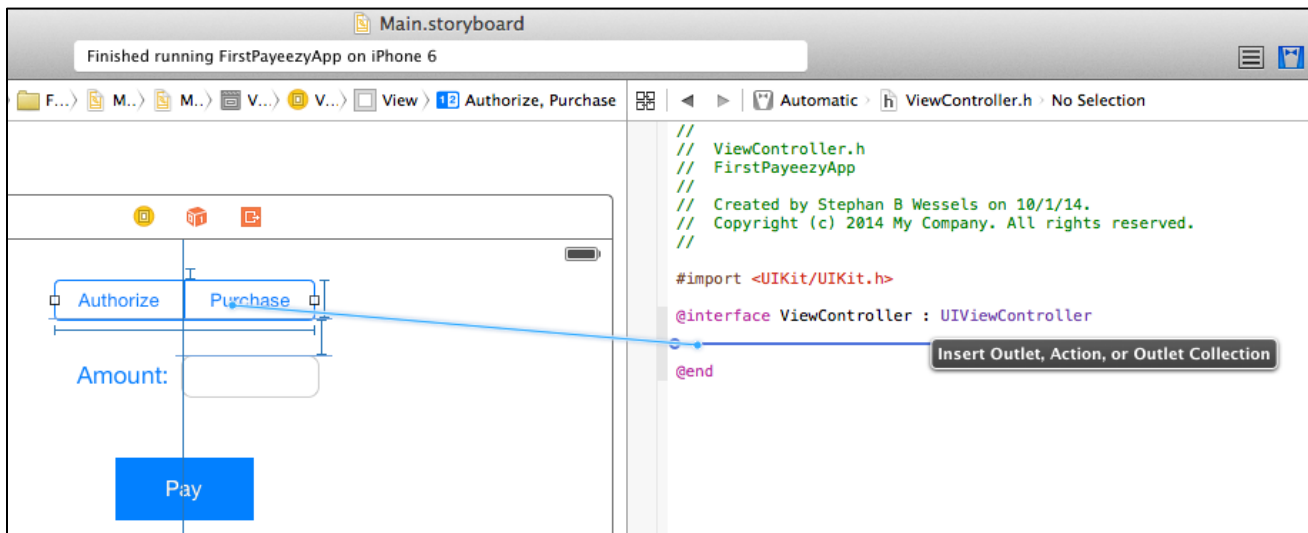
To test out our layout let's run the project using the iPhone 6 as the target in the simulator. You should see the visual objects remain nicely aligned in portrait or landscape orientations.



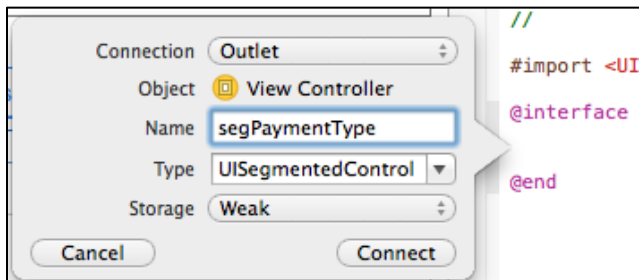
## Create Outlets and an Action for Our View Controller

It's time to hook a few things up. With the Storyboard still selected you can use Xcode's Assistant Editor and pair it with the ViewController.h file. We will need to make outlets for the segment control and the test input field. We will also need to create an action related to pressing the **Pay** button.

With the Storyboard open and the ViewController open, Control-Click and drag from the segmented control to the code pane.

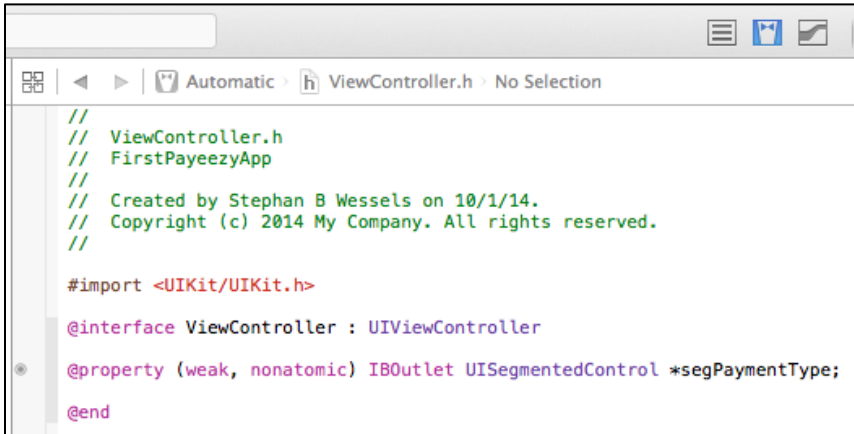


Xcode will present a pop-up where you can define the name of the instance variable to associate with the visual object's outlet. Fill out the pop-up.



Name the property `segPaymentType` and click **connect**. The following line of code will be added to your `ViewController.h`.

## First Data Apple Pay Quick Start Guide



```
//
// ViewController.h
// FirstPayeezyApp
//
// Created by Stephan B Wessels on 10/1/14.
// Copyright (c) 2014 My Company. All rights reserved.
//

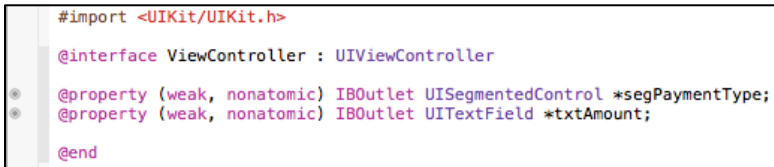
#import <UIKit/UIKit.h>

@interface ViewController : UIViewController

@property (weak, nonatomic) IBOutlet UISegmentedControl *segPaymentType;

@end
```

Do the same thing for the Text Field and name the property txtAmount. Your header file should now look like this.



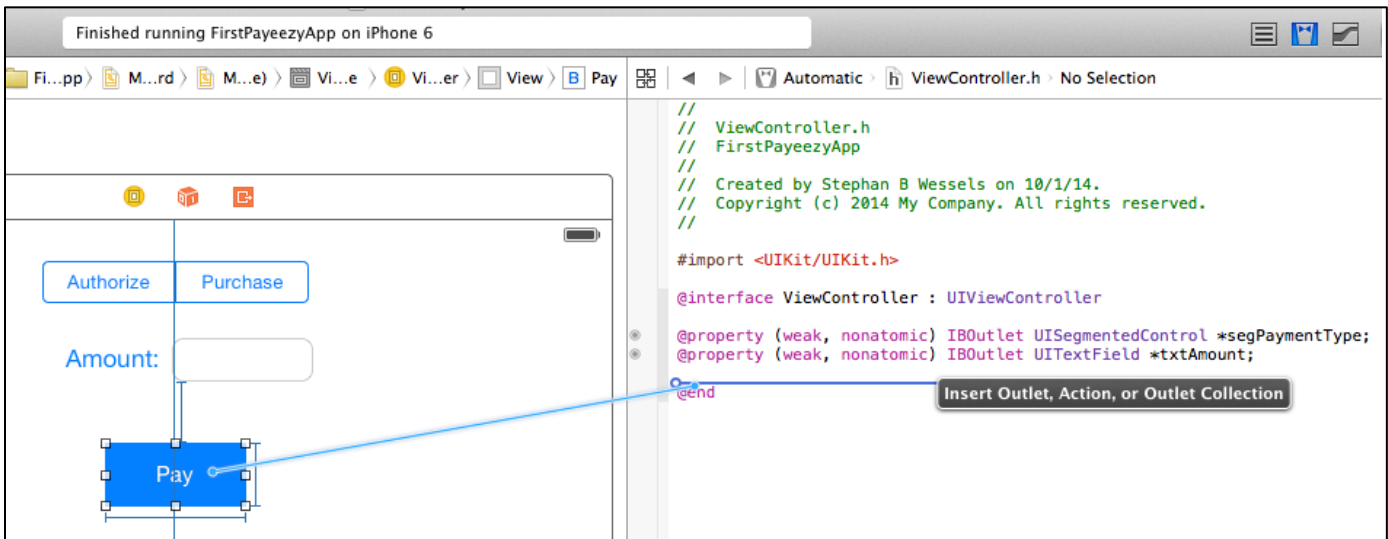
```
#import <UIKit/UIKit.h>

@interface ViewController : UIViewController

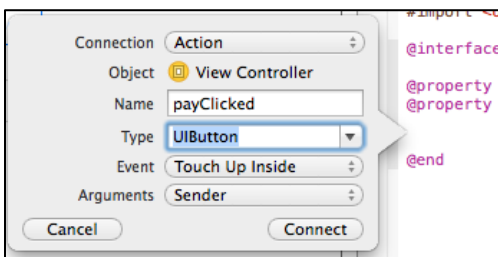
@property (weak, nonatomic) IBOutlet UISegmentedControl *segPaymentType;
@property (weak, nonatomic) IBOutlet UITextField *txtAmount;

@end
```

We need to create an action related to pressing the **Pay** button. Control-drag from the **Pay** button to the header file.



When Xcode presents the property dialog, select Action in the popup and then set the other values as follows.



After clicking on **Connect** your header code should now look like this.

```
#import <UIKit/UIKit.h>

@interface ViewController : UIViewController

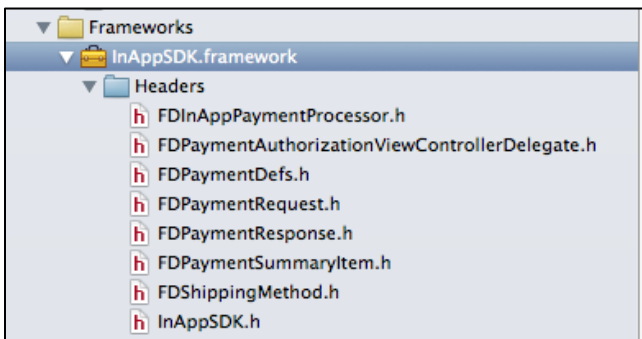
@property (weak, nonatomic) IBOutlet UISegmentedControl *segPaymentType;
@property (weak, nonatomic) IBOutlet UITextField *txtAmount;

- (IBAction)payClicked:(UIButton *)sender;

@end
```

## Design Considerations

The frameworks you installed includes one named `InAppSDK.framework`. If you expand the Headers for the framework you can see the various `.h` files that define the developer accessible SDK.



The `FDInAppPaymentProcessor` is an instance you will create, typically in your App Delegate, that is used to send messages to Payeezy. The headers for the following...

```
FDPaymentRequest
FDPaymentResponse
FDPaymentSummaryItem
FDShippingMethod
```

are for the object models you will create as you interact with Payeezy. You will also assign the `FDPaymentAuthorizationViewControllerDelegate` protocol for callback messages as you interact with Apple Pay and Payeezy.

The `InAppSDK.h` is a convenience that pulls all the pieces together for you.

To begin to bolt all this together, let's go back to the `AppDelegate.h` file. We need to define constants that were obtained during the registration process and declare them in the app delegate.

## First Data Apple Pay Quick Start Guide

Change your AppDelegate.h by adding these define statements as shown.

```
#import <UIKit/UIKit.h>

#define kApiKey          @"xHiPf8dp8TDzkRunn1Ovvf2hrOK5Y5YT"
#define kApiSecret       @"9cbb9acfc23a2440ef075d4696f13a373e26f9e9ab7c0a8ef9c7048ea3430bf8"
#define kMerchantToken   @"fdoa-a480ce8951daa73262734cf102641994c1e55e7cdf4c02b6"
#define kApplePayMerchantId @"merchant.com.firstpayeezy"

@interface AppDelegate : UIResponder <UIApplicationDelegate>
@property (strong, nonatomic) UIWindow *window;
```

The values used can be found from the Payeezy portal and/or in this document as part of the work you did earlier.

Use your own numbers/keys when you develop this example application.

It's time to add the `FDInAppPaymentProcessor` to the header. Change the header file to have the forward class definition and the new property for the processor we will create. When you are done your header should look like this (with your own keys substituted).

```
#import <UIKit/UIKit.h>

@class FDInAppPaymentProcessor;

#define kApiKey          @"xHiPf8dp8TDzkRunn1Ovvf2hrOK5Y5YT"
#define kApiSecret       @"9cbb9acfc23a2440ef075d4696f13a373e26f9e9ab7c0a8ef9c7048ea3430bf8"
#define kMerchantToken   @"fdoa-a480ce8951daa73262734cf102641994c1e55e7cdf4c02b6"
#define kApplePayMerchantId @"merchant.com.firstpayeezy"

@interface AppDelegate : UIResponder <UIApplicationDelegate>

@property (strong, nonatomic) UIWindow *window;
@property (strong, nonatomic) FDInAppPaymentProcessor *fdPaymentProcessor;

@end
```

Switch to the AppDelegate.m. Add the import statement to use the In App SDK. Include one new line as follows.

```
#import "AppDelegate.h"
#import <InAppSDK/InAppSDK.h>

@implementation AppDelegate
```

With this change all of the import statements we will need are available to us.

Add a new worker method to instantiate the payment processor for us.

```
- (void)instantiateFDLibService
{
    self.fdPaymentProcessor = [[FDInAppPaymentProcessor alloc] initWithApiKey:kApiKey
                                                                    apiSecret:kApiSecret
                                                                    merchantToken:kMerchantToken
                                                                    merchantIdentifier:kApplePayMerchantId];

    // The app can choose which mode to send transactions in: pre-auth only or purchase
    // The default is purchase
    self.fdPaymentProcessor.paymentMode = FDPPreAuthorization;
}
```

We will instantiate our payment process when the application launches. Change the `application:didFinishLaunchingWithOptions:` method.

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    // Override point for customization after application launch.
    [self instantiateFDLibService];
    return YES;
}
```

It's time to work on the ViewController. We've created a button that initiates an action via the `payClicked:` method. For reasons that will become clear as we progress, we need to declare our ViewController to be an `FDPaymentAuthorizationViewControllerDelegate`. In the `ViewController.h` file change the header to import and declare the delegate protocol.

```
#import <UIKit/UIKit.h>
#import <InAppSDK/FDPaymentAuthorizationViewControllerDelegate.h>

@interface ViewController : UIViewController <FDPaymentAuthorizationViewControllerDelegate>

@property (weak, nonatomic) IBOutlet UISegmentedControl *segPaymentType;
@property (weak, nonatomic) IBOutlet UITextField *txtAmount;

- (IBAction)payClicked:(UIButton *)sender;

@end
```

As we interact with the payment processing system we will receive specific callback messages through this delegate protocol. We'll cover details on those soon. For now, we have completed the edits to the `ViewController.h` file.



## First Data Apple Pay Quick Start Guide

In the `ViewController.m` file we will begin to add behavior to our `payClicked:` method. We'll need a convenient way to interact with our App Delegate so we add that first. Imports for the App delegate and our new SDK will be needed too. Update the code as follows:

```
#import "ViewController.h"
#import "AppDelegate.h"
#import <InAppSDK/InAppSDK.h>

@interface ViewController ()

@end

@implementation ViewController

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

- (IBAction)payClicked:(UIButton *)sender
{
    AppDelegate *appDelegate = (AppDelegate *)[[UIApplication sharedApplication] delegate];
}

@end
```

There are a few warnings now and they are related to our new unused variable and the fact that we have not implemented the required `FDPaymentAuthorizationViewControllerDelegate` methods yet.

To proceed we get to make a design decision. When a request is made of Apple Pay, the application does not know which kinds of payment cards the user has installed in their Passbook App. However we do know which payment networks our application will support. We need to declare those now. Whether the user actually has a matching card for one of the networks we seek is not known.

Add the following to `payClicked:`.

```
- (IBAction)payClicked:(UIButton *)sender
{
    AppDelegate *appDelegate = (AppDelegate *)[[UIApplication sharedApplication] delegate];
    NSArray *supportedNetworks = @[
        FDPaymentNetworkVisa,
        FDPaymentNetworkMasterCard,
        FDPaymentNetworkAmericanExpress
    ];
}
```

Here we are declaring an array to contain the payment networks our application supports.

There are two critical checks we need to make next. First it is possible someone may have installed our new application on an Apple device that does not support Apple Pay. So we need to check that first. Second, even if the device can support Apple Pay the ability to make payments on that device is a user controlled setting.

We need to also pass that check. This is easily done as follows in our method. Once we know using Apple Pay is possible we check if the user has at least one card in our supported networks.

```
- (IBAction)payClicked:(UIButton *)sender
{
    AppDelegate *appDelegate = (AppDelegate *)[[UIApplication sharedApplication] delegate];
    NSArray *supportedNetworks = @[
        FDPaymentNetworkVisa,
        FDPaymentNetworkMasterCard,
        FDPaymentNetworkAmericanExpress
    ];

    // Does this device support In-App payments?
    if ([FDInAppPaymentProcessor canMakePayments])
    {
        // Is a card registered on the device for one of the merchant's supported card networks?
        if ([FDInAppPaymentProcessor canMakePaymentsUsingNetworks:supportedNetworks])
        {
        }
        else
        {
            NSLog(@"Ability to make payments of merchant-supported network types was rejected
by FD SDK");
        }
    }
    else
    {
        NSLog(@"Ability for device to make payments was rejected by FD SDK");
    }
}
```

For this example we'll just log the error condition. You'll want better user feedback in your own application.

If we proceed, then next step is to create an `FDPaymentRequest` object.

## First Data Apple Pay Quick Start Guide

```
- (IBAction)payClicked:(UIButton *)sender
{
    AppDelegate *appDelegate = (AppDelegate *)[UIApplication sharedApplication] delegate];
    NSArray *supportedNetworks = @[
        FDPaymentNetworkVisa,
        FDPaymentNetworkMasterCard,
        FDPaymentNetworkAmericanExpress
    ];

    // Does this device support In-App payments?
    if ([FDInAppPaymentProcessor canMakePayments])
    {
        // Is a card registered on the device for one of the merchant's supported card networks?
        if ([FDInAppPaymentProcessor canMakePaymentsUsingNetworks:supportedNetworks])
        {
            // Populate the payment request
            FDPaymentRequest *pmtRqst = [[FDPaymentRequest alloc] init];
            pmtRqst.merchantIdentifier = kApplePayMerchantId;
            pmtRqst.supportedNetworks = supportedNetworks;
            pmtRqst.countryCode = @"US";
            pmtRqst.currencyCode = @"USD";
            pmtRqst.merchantCapabilities = FDMerchantCapability3DS | FDMerchantCapability3EMV;
            pmtRqst.requiredShippingAddressFields = FDAddressFieldNone;
        }
        else
        {
            NSLog(@"Ability to make payments of merchant-supported network types was rejected
by FD SDK");
        }
    }
    else
    {
        NSLog(@"Ability for device to make payments was rejected by FD SDK");
    }
}
```

Next we look at the GUI we made and determine whether we are performing an Authorize or Purchase action. Our `segPaymentType` property holds the chosen option.

We tell the payment processor about it by setting a property. Add one line of code inside where we have been building up our payment request.

```
{
    // Populate the payment request
    FDPaymentRequest *pmtRqst = [[FDPaymentRequest alloc] init];
    pmtRqst.merchantIdentifier = kApplePayMerchantId;
    pmtRqst.supportedNetworks = supportedNetworks;
    pmtRqst.countryCode = @"US";
    pmtRqst.currencyCode = @"USD";
    pmtRqst.merchantCapabilities = FDMerchantCapability3DS | FDMerchantCapability3EMV;
    pmtRqst.requiredShippingAddressFields = FDAddressFieldNone;
    // Set the payment type
    appDelegate.fdPaymentProcessor.paymentMode =
    (self.segPaymentType.selectedSegmentIndex==0 ? FDPreAuthorization : FDPurchase);
}
```

The next step is to declare an `FDShippingMethod`. Add the following code so your conditional code looks as follows:

```
{
    // Populate the payment request
    FDPaymentRequest *pmtRqst = [[FDPaymentRequest alloc] init];
    pmtRqst.merchantIdentifier = kApplePayMerchantId;
    pmtRqst.supportedNetworks = supportedNetworks;
    pmtRqst.countryCode = @"US";
    pmtRqst.currencyCode = @"USD";
    pmtRqst.merchantCapabilities = FDMerchantCapability3DS | FDMerchantCapability3EMV;
    pmtRqst.requiredShippingAddressFields = FDAddressFieldNone;
    // Set the payment type
    appDelegate.fdPaymentProcessor.paymentMode =
    (self.segPaymentType.selectedSegmentIndex==0 ? FDPreAuthorization : FDPurchase);
    FDShippingMethod *shipping = [[FDShippingMethod alloc] init];
    shipping.identifier = @"Two Day Shipping";
    shipping.detail = @"Two day shipping to the Continental US";
    pmtRqst.shippingMethods = @[shipping];
}
```

Now we have to add the details about what is being purchased. If we have more than one item in our order we would add more lines. Here, we'll add one item and purchase some "large shoes". Not really very descriptive, but...

The amount is pulled from the interface again in our `txtAmount` property.

```
{
    // Populate the payment request
    FDPaymentRequest *pmtRqst = [[FDPaymentRequest alloc] init];
    pmtRqst.merchantIdentifier = kOsloMerchantId;
    pmtRqst.supportedNetworks = supportedNetworks;
    pmtRqst.countryCode = @"US";
    pmtRqst.currencyCode = @"USD";
    pmtRqst.merchantCapabilities = FDMerchantCapability3DS | FDMerchantCapability3EMV;
    pmtRqst.requiredShippingAddressFields = FDAddressFieldNone;

    // Set the payment type
    appDelegate.fdPaymentProcessor.paymentMode =
    (self.segPaymentType.selectedSegmentIndex==0 ? FDPreAuthorization : FDPurchase);
    FDShippingMethod *shipping = [[FDShippingMethod alloc] init];
    shipping.identifier = @"Two Day Shipping";
    shipping.detail = @"Two day shipping to the Continental US";
    pmtRqst.shippingMethods = @[shipping];
    // Create a sample order
    FDPaymentSummaryItem *item1 = [[FDPaymentSummaryItem alloc] init];
    item1.label = @"Large Shoes";
    item1.amount = [NSDecimalNumber decimalNumberWithString:self.txtAmount.text];
}
```

We're building up an array of individual items on the order. In this example there is just one.

## First Data Apple Pay Quick Start Guide

The last item in the array is the total line.

```
{
    // Populate the payment request
    FDPaymentRequest *pmtRqst = [[FDPaymentRequest alloc] init];
    pmtRqst.merchantIdentifier = kOsloMerchantId;
    pmtRqst.supportedNetworks = supportedNetworks;
    pmtRqst.countryCode = @"US";
    pmtRqst.currencyCode = @"USD";
    pmtRqst.merchantCapabilities = FDMerchantCapability3DS | FDMerchantCapability3EMV;
    pmtRqst.requiredShippingAddressFields = FDAddressFieldNone;

    // Set the payment type
    appDelegate.fdPaymentProcessor.paymentMode =
    (self.segPaymentType.selectedSegmentIndex==0 ? FDPAuthorization : FDPurchase);

    FDShippingMethod *shipping = [[FDShippingMethod alloc] init];
    shipping.identifier = @"Two Day Shipping";
    shipping.detail = @"Two day shipping to the Continental US";
    pmtRqst.shippingMethods = @[shipping];

    // Create a sample order
    FDPaymentSummaryItem *item1 = [[FDPaymentSummaryItem alloc] init];
    item1.label = @"Large Shoes";
    item1.amount = [NSDecimalNumber decimalNumberWithString:self.txtAmount.text];

    // Total line
    FDPaymentSummaryItem *item2 = [[FDPaymentSummaryItem alloc] init];
    item2.label = @"FD Test Merchant1";
    item2.amount = [NSDecimalNumber decimalNumberWithString:self.txtAmount.text];
    NSArray *itemArray = [NSArray arrayWithObjects: item1, item2, nil];
    pmtRqst.paymentSummaryItems = itemArray;
}
```

The `itemArray` contains the items for the order. Apple recommends your total line should contain the merchant name as the description. After creating the order items array we set the property on our payment request (`pmtRqst`).

There's one more value we can use in the `pmtRqst`. Modify the last few lines of our conditional branch by adding the new code shown.

```
    // Total line
    FDPaymentSummaryItem *item2 = [[FDPaymentSummaryItem alloc] init];
    item2.label = @"FD Test Merchant1";
    item2.amount = [NSDecimalNumber decimalNumberWithString:self.txtAmount.text];
    NSArray *itemArray = [NSArray arrayWithObjects: item1, item2, nil];
    pmtRqst.paymentSummaryItems = itemArray;

    // Send a sample application data payload
    NSString *appDataString = @"RefCode:12345; TxID:34234089240982304823094823432";
    pmtRqst.applicationData = [appDataString dataUsingEncoding:NSUTF8StringEncoding];
}
else
```

The `NSString *appDataString` put into `pmtRqst.applicationData` after encoding, is an optional parameter. Apple recommends this can be used for additional data as appropriate for your app—for example, a shopping cart identifier or an order number. See Apple's docs about this parameter [here](#).

This is an optional value that you control. We are putting the string `RefCode:12345;TxID:34234089240982304823094823432` in our payment request for this example. The data will be encrypted before it gets sent to the server.

The payment request object is ready. At this point the application must present the payment request object to the `FDInAppPaymentProcessor`.

Add the lines shown below after we set the optional `applicationData`.

```

    BOOL bPaymentOK = [appDelegate.fdPaymentProcessor
presentPaymentAuthorizationViewControllerWithPaymentRequest:pmtRqst presentingController:self
delegate:self];

    if( !bPaymentOK ) {
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Error"
                                                                message:@"Payment request was rejected by
Apple Pay
server" delegate:self
                                                                cancelButtonTitle:@"Dismiss"
                                                                otherButtonTitles:nil];

        [alert show];
        NSLog(@"Payment request was rejected by Apple Pay server");
    }

    // Delegate methods are driving from here...

```

When we interact with the payment processor we are turning over control to Apple's Passbook code. Apple will present the next view controller and handle interaction with the customer.

If there was a problem the boolean value will be `NO` and your application can present an alert dialog. If the payment request was accepted by the iPhone all following operations are sent as messages to your delegate. We will add those delegate messages soon.

## First Data Apple Pay Quick Start Guide

To help you make sure there were no errors made as you built-up the previous section of code the entire payClicked: method is repeated here for reference.

```
- (IBAction)payClicked:(UIButton *)sender
{
    AppDelegate *appDelegate = (AppDelegate *)[UIApplication sharedApplication] delegate];
    NSArray *supportedNetworks = @[
        FDPaymentNetworkVisa,
        FDPaymentNetworkMasterCard,
        FDPaymentNetworkAmericanExpress
    ];

    // Does this device support In-App payments?
    if ([FDInAppPaymentProcessor canMakePayments])
    {

        // Is a card registered on the device for one of the merchant's supported card networks?
        if ([FDInAppPaymentProcessor canMakePaymentsUsingNetworks:supportedNetworks])
        {

            // Populate the payment request
            FDPaymentRequest *pmtRqst = [[FDPaymentRequest alloc] init];
            pmtRqst.merchantIdentifier = kOsloMerchantId;
            pmtRqst.supportedNetworks = supportedNetworks;
            pmtRqst.countryCode = @"US";
            pmtRqst.currencyCode = @"USD";
            pmtRqst.merchantCapabilities = FDMerchantCapability3DS | FDMerchantCapability3EMV;
            pmtRqst.requiredShippingAddressFields = FDAddressFieldNone;

            // Set the payment type
            appDelegate.fdPaymentProcessor.paymentMode =
(self.segmentPaymentType.selectedSegmentIndex==0 ?
            FDPreAuthorization : FDPurchase);
            FDShippingMethod *shipping = [[FDShippingMethod alloc] init];
            shipping.identifier = @"Two Day Shipping";
            shipping.detail = @"Two day shipping to the Continental US";
            pmtRqst.shippingMethods = @[shipping];

            // Create a sample order
            FDPaymentSummaryItem *item1 = [[FDPaymentSummaryItem alloc] init];
            item1.label = @"Large Shoes";

            item1.amount = [NSDecimalNumber decimalNumberWithString:self.txtAmount.text];

            // Total line
            FDPaymentSummaryItem *item2 = [[FDPaymentSummaryItem alloc] init];
            item2.label = @"FD Test Merchant1";
            item2.amount = [NSDecimalNumber decimalNumberWithString:self.txtAmount.text];
            NSArray *itemArray = [NSArray arrayWithObjects: item1, item2, nil];
            pmtRqst.paymentSummaryItems = itemArray;

            // Send a sample application data payload
            NSString *appDataString = @"RefCode:12345; TxID:34234089240982304823094823432";
            pmtRqst.applicationData = [appDataString dataUsingEncoding:NSUTF8StringEncoding];

            BOOL bPaymentOK = [appDelegate.fdPaymentProcessor
            presentPaymentAuthorizationViewControllerWithPaymentRequest:pmtRqst presentingController:self
            delegate:self];

            if( !bPaymentOK ) {
                UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Error"
                    message:@"Payment request was rejected by
Apple Pay server" delegate:self
                    cancelButtonTitle:@"Dismiss"
                    otherButtonTitles:nil];
            }
        }
    }
}
```

```

                                otherButtonTitles:nil];
        [alert show];
        NSLog(@"Payment request was rejected by Apple Pay server");
    }
    // Delegate methods are driving from here...
}
else
{
    NSLog(@"Ability to make payments of merchant-supported network types was rejected by
FD SDK");
}
else
{
    NSLog(@"Ability for device to make payments was rejected by FD SDK");
}
}
}

```

Don't forget you will want to handle your `else` conditions with something more helpful than `NSLog`.

### The `FDPaymentAuthorizationViewControllerDelegate` Methods

There are four delegate methods we need to provide. After the `payClicked:` method add a pragma statement to the code to make these delegate methods easier to find.

```
#pragma mark - FDPaymentAuthorizationViewControllerDelegate
```

Add the first of our delegate methods.

```

- (void)paymentAuthorizationViewController:(UIViewController *)controller
                                didAuthorizePayment:(FDPaymentResponse *)paymentResponse
{
    NSString *authStatusMessage = nil;

    if (paymentResponse.validationStatus != FDPaymentValidationStatusSuccess)
    {
        authStatusMessage = @"Transaction Validation or communication failure. Please try
again.";
    }
    else if (paymentResponse.authStatus == FDPaymentAuthorizationStatusFailure)
    {
        authStatusMessage = [NSString stringWithFormat:@"Transaction was validated but
authorization failed with reason: %@", paymentResponse.transStatusMessage];
    }
    else if (paymentResponse.authStatus == FDPaymentAuthorizationStatusSuccess)
    {
        authStatusMessage = [NSString stringWithFormat:@"Transaction
Successful\rType:%@\rTransaction ID:%@\rTransaction Tag:%@",
            paymentResponse.transactionType,
            paymentResponse.transactionID,
            paymentResponse.transactionTag];
    }

    UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"First Data Payment Authorization"
                                                    message:authStatusMessage delegate:self
                                                    cancelButtonTitle:@"Dismiss"
                                                    otherButtonTitles:nil];

    [alert show];
}

```



## First Data Apple Pay Quick Start Guide

```
// App can send order to the merchant's back-end server now
}
```

The `paymentAuthorizationViewController:didAuthorizePayment:` delegate method is called when the customer has authorized the payment.

The `paymentResponse` can be checked by the techniques shown to know if the response was both valid and authorized.

Your application can notify the merchant's back-end server at this point.

The next delegate method is called as the SDK dismisses the View Controller it used to interact with the customer. Add this delegate method now.

```
- (void)paymentAuthorizationViewControllerDidFinish:(UIViewController *)controller
{
    // Nothing to do here - the SDK handles all cleanup

    NSLog(@"ViewController:paymentAuthorizationViewControllerDidFinish invoked");
}
```

The only thing we do in here is log that the delegate was invoked.

The two remaining delegate methods are related to customer choices involving shipping information. Add this delegate method.

```
- (void)paymentAuthorizationViewController:(UIViewController *)controller
    didSelectShippingMethod:(FDShippingMethod *)shippingMethod
{
    // The customer has asked the merchant to use a specific shipping method. We need to update
    the amount.
    // We can also get here if the customer said the heck with it and canceled.

    NSLog(@"ViewController:didSelectShippingMethod invoked");
}
```

When the customer selects a shipping method your application is called by the SDK. This allows you to update the total amount if shipping method is impactful.

Add our final delegate method.

```
- (void)paymentAuthorizationViewController:(UIViewController *)controller
    didSelectShippingAddress:(ABRecordRef)address
{
    // The customer is telling the merchant shipping address information. We need to update the
    amount.
    // We can also get here if the customer said the heck with it and canceled.

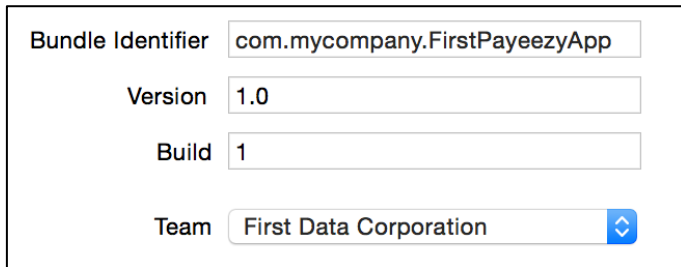
    NSLog(@"ViewController:didSelectShippingAddress invoked");
}
```

Your application is also called through this delegate method if the customer has changed the shipping address. This mechanism provides for a way to change the amount if necessary.

## Managing Entitlements

Entitlements are automatically added to your Xcode project when you turn on Apple Pay settings. However there are a few things we must check before we proceed.

Select your project in Xcode and go to the General tab. Look up the Identity information for your project.



Bundle Identifier: com.mycompany.FirstPayeezyApp

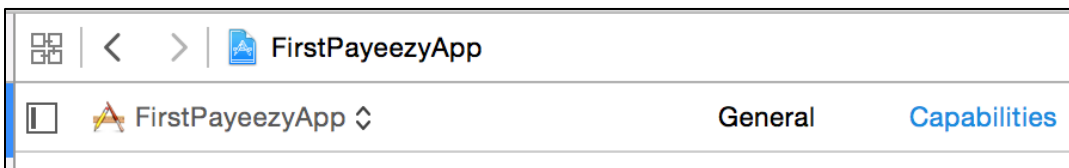
Version: 1.0

Build: 1

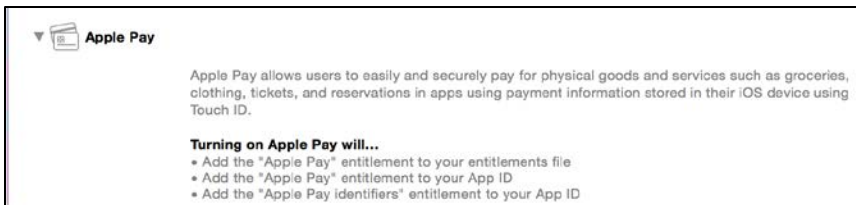
Team: First Data Corporation

Ensure you have the proper Team selected and that there are no issues to resolve.

Select the **Capabilities** tab.

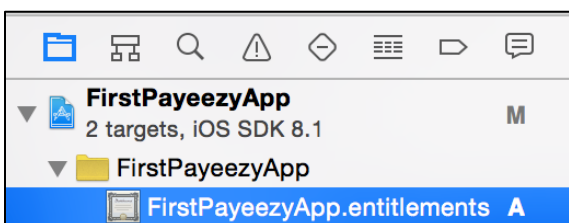


Toggle the disclosure triangle on Apple Pay. You should see a description of what will happen when you enable this capability.



Turn the capability on.

Several things will happen automatically. You should notice a new entitlements file has been added to your project.




## First Data Apple Pay Quick Start Guide

When you select the entitlements file it will contain a Dictionary containing an empty Array.



Key	Type	Value
▼ Entitlements File	Dictionary	⌵ (1 item)
▶ com.apple.developer.in-app-pay...	Array	⌵ (0 items)

Under the Apple Pay capability a list of known Apple Pay Identifiers is shown.


 **Apple Pay**

Apple Pay Identifiers:

<input type="checkbox"/>	merchant.com.cat.PKN000
<input type="checkbox"/>	merchant.com.cat.test
<input type="checkbox"/>	merchant.com.cert.PKN000

Select one or more of the Merchant IDs from this list. You can also add a new one by using the + button below the table.

 **Add a new identifier**

Xcode will create a new identifier if the named identifier doesn't already exist, add it to your App ID, and add the new identifier to your app's entitlements.

1stAppTest

Cancel


OK


Once you have Apple Pay enabled you will also need to enable In-App Purchase.

 **In-App Purchase**

You may find errors as you proceed. Xcode will identify what you need to do.

Steps: ✓ Add the "Apple Pay" entitlement to your entitlements file

 Add the "Apple Pay" entitlement to your App ID

 Add the "Apple Pay identifiers" entitlement to your App ID

Fix Issues

## Appendix

### Document Edit History

Version	Date	Author	Description
1.2	02-Mar-2018	M. Hayes	Changed FD Developer Portal registration and CSR generation to updated Portal process and UI
1.1	02-Jul-2015	P. Sahu / S. Wessels	Updates and corrections based upon latest SDK and Git Hub access. Also clarified Merchant ID lookup information.
1.0.1	01-Jun-2015	S Wessels	Changed overall page layouts. Cleanup of source code sections. Edit for typos and consistent formats.
1.0	26-Oct-2014	S Wessels	Initial release.
0.9.6	23-Oct-2014	S Wessels	Added content.
0.9.5	22-Oct-2014	S Wessels	Added content.
0.9.4	20-Oct-2014	S Wessels	Initial draft.