

MODULE: Deep Dive into iOS App Development

SESSION [3]

SESSION TITLE: [NOTIFICATIONS]

Sources:

1. [Professional iOS Programming] [Chapter 10][ISBN No 9781118661130]
2. [Wiley Book Name] [Chapter Number][ISBN No]
3. [Wiley Book Name] [Chapter Number][ISBN No]
4. [Wiley Book Name] [Chapter Number][ISBN No]

Example: [Cloud Computing for Dummies] [Chapter 5] [978-0-470-48470-8]

MODULE Objectives

At the end of this module, you will be able to:

- Implement internal and external notifications
- Implement push notifications

Session Objectives

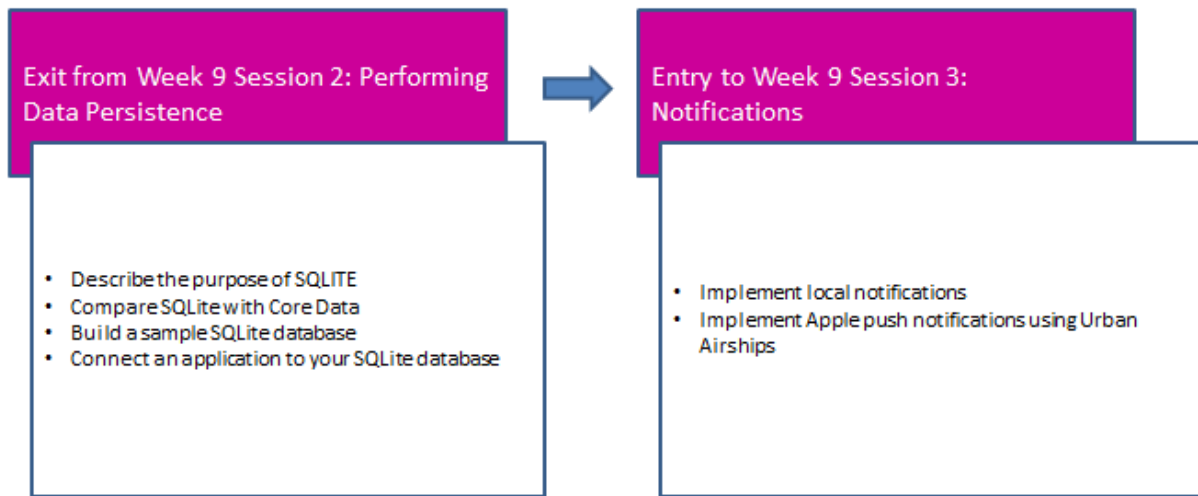
At the end of this session, you will be able to:

- Implement local notifications
- Implement Apple push notifications using Urban Airships
- Register customer URL schemes

<H1> Introduction

Local notifications and push notifications are technologies you can implement in your iOS application that inform your application (when not running in the foreground) that an event happened or new information is available. Like in a mail application when your application is running in the background and you receive a new mail, a local notification is created to let you know a new mail has been delivered to your Inbox.

In this session, you will learn how to implement notifications from a background process or if you receive any file from an external source.



<H1> [Implementing Local notifications]

Source: [Professional iOS Programming] [10] [336]

Local notifications are notifications that are generated and received by your application. The purpose of the local notification is to inform your application's user that something interesting for him/her happened while the application was not running in the foreground.

<H2> [Understanding Local Notifications]

The main characteristics of a local notification are:

- They are launched and delivered by iOS on the same device.
- They can be scheduled by date and time.
- They can be created instantly based on an event.
- They can include an `NSDictionary` with custom data.
- They can contain a sound to play (sound notification), an application badge number (badge notification), and the title of an action button (alert).

Definition: Badge Notification

A badge can be used when a notification occurs. The badge shows the number of notifications as a small icon. It can be added on the main menu icon or can be checked when the user launches an application.

Definition: Sound Notification

You can implement a custom sound on a notification. It is like when you get a call, your phone rings with a particular sound tone. Similarly, you can implement any sound tone on your notifications.

Definition: Alerts

Alerts are used to give important information to users while they are working on the application. In other words, alerts come when an application is in process. The option you have available for alerts in iOS is the alert view.

To learn and understand how to work with local notifications, you will now create a new application that will act like an alarm clock.

1. Start Xcode and create a new project using the **Single View Application** template. Name the project as **AlarmClock** using the options shown in **Figure 1**.

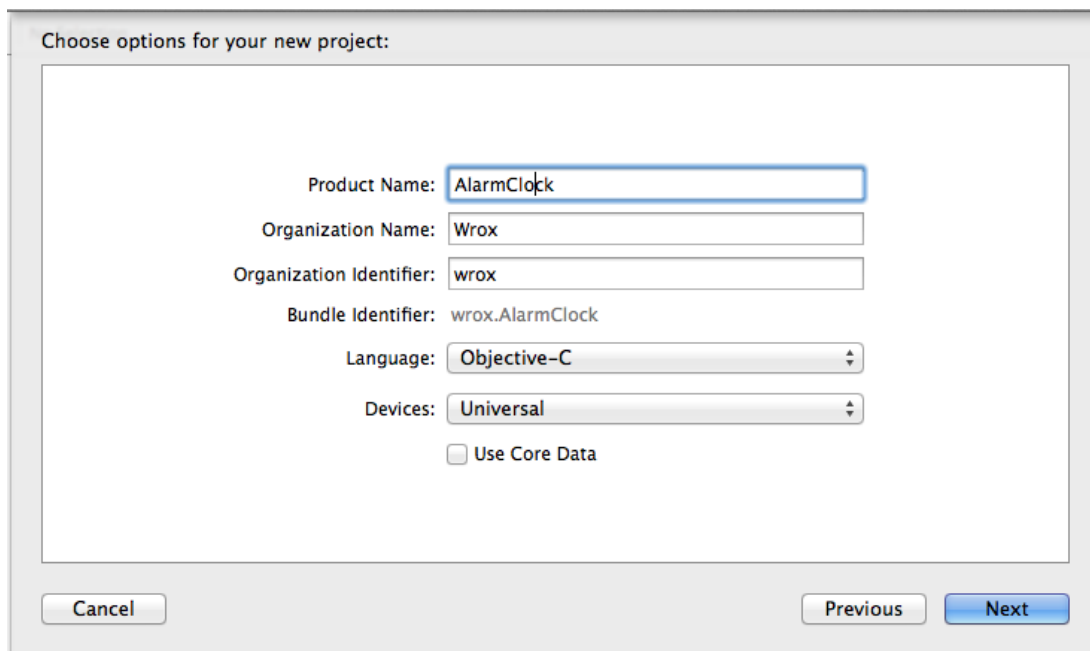


Figure 1: Create a New Project, AlarmClock, in Xcode

2. Use Interface Builder and the Assistant Editor to create a user interface for the `YDViewController.xib` file, as shown in **Figure 2**.

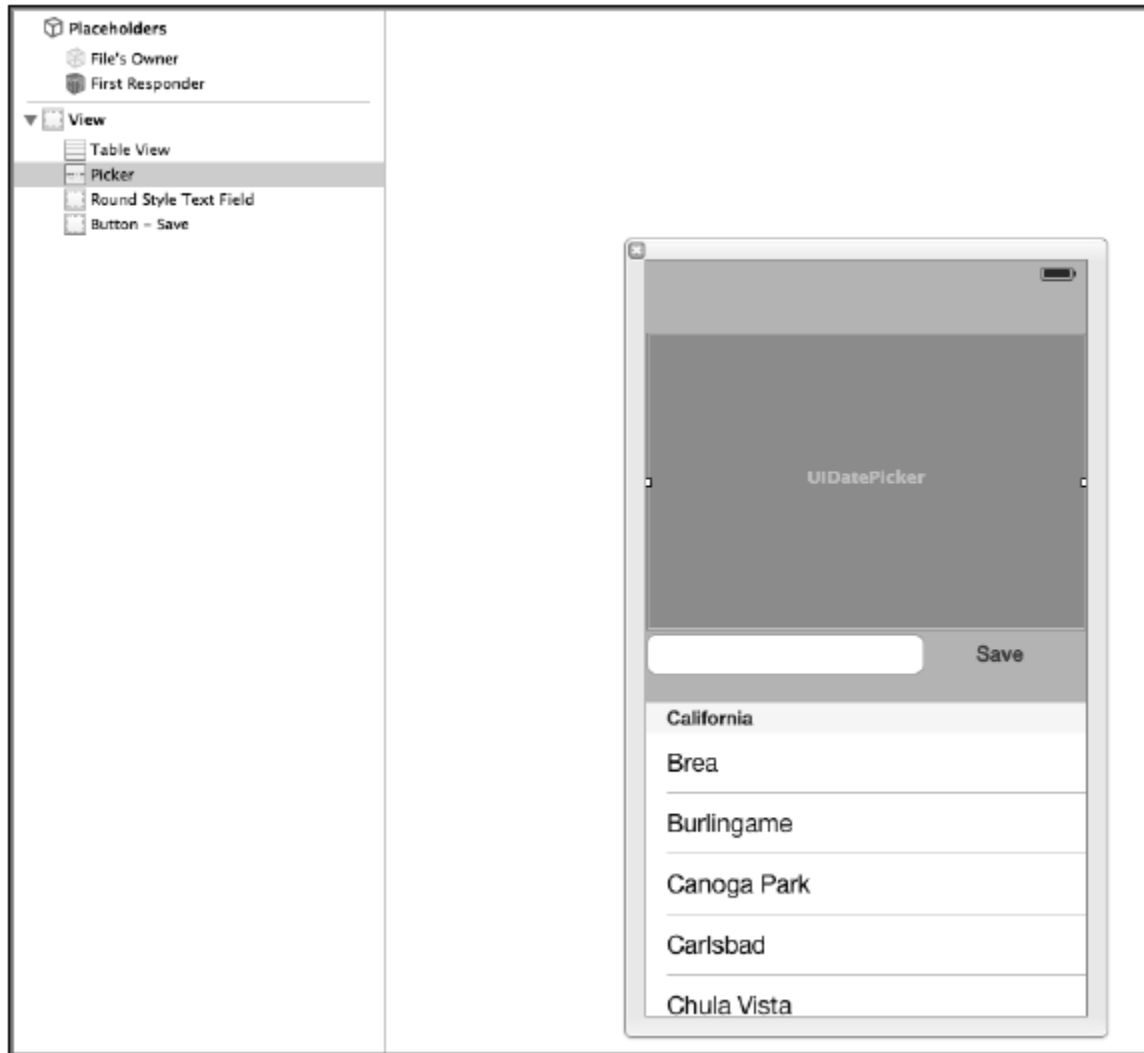


Figure 2: Create a User Interface for YDViewController.xib

<H3> [Creating a Notification]

Verify your YDViewController.h file is equal to the one shown below:

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

```
@interface YDViewController : UIViewController
```

```
@property (weak, nonatomic) IBOutlet UIDatePicker *dPicker;
```

```
@property (weak, nonatomic) IBOutlet UITextField *EventText;
```

```
@property (weak, nonatomic) IBOutlet UITableView *mTableView;
```

```
- (IBAction)saveEvent:(UIButton *)sender;
@end
```

The following code snippet shows the implementation of your `YDViewController.h` file. Here, the `UILocalNotification` object is created and initialized in the `saveEvent:` method. The `DatePicker` value is converted to a date and time on which the event should be launched, and the `alertBody` is set. Finally, you create a custom `NSDictionary` with values and keys, which you would like to pass to the `userinfo` dictionary.

```
#import "YDViewController.h"

@interface YDViewController ()

@end

@implementation YDViewController

- (void)viewDidLoad
{
    [super viewDidLoad];
}

#pragma mark Table view data source

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return 1;
}

- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSectionSection:
    (NSInteger)section {
    // Return the number of rows in the section.
    return [[[UIApplication sharedApplication] scheduledLocalNotifications]
        count];
}

// Customize the appearance of table view cells.
- (UITableViewCell *)tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath {

    static NSString *CellIdentifier = @"Cell";
```

```

        UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:
CellIdentifier];
        if (cell == nil) {
            cell = [[UITableViewCell alloc] initWithStyle:
UITableViewCellStyleSubtitle reuseIdentifier:CellIdentifier];
        }
        NSArray *notificationArray = [[UIApplication sharedApplication]
scheduledLocalNotifications];
        UILocalNotification *notification = [notificationArray
objectAtIndex:indexPath.row];

        [cell.textLabel setText:notification.alertBody];
        [cell.detailTextLabel setText:[notification.fireDate description]];

        return cell;
    }
    - (IBAction)saveEvent:(UIButton *)sender {
        [self.EventText resignFirstResponder];

        NSCalendar *calendar = [NSCalendar autoupdatingCurrentCalendar];

        // Get the current date
        NSDate *pickerDate = [self.dPicker date];

        // Break the date up into components
        NSDateComponents *dateComponents =
            [calendar components:
            ( NSYearCalendarUnit |
            NSMonthCalendarUnit |
            NSDayCalendarUnit )
            fromDate:pickerDate];
        NSDateComponents *timeComponents =

        [calendar components:
            ( NSHourCalendarUnit |
            NSMinuteCalendarUnit |
            NSSecondCalendarUnit )
            fromDate:pickerDate];

        // Set up the fire time

```

```

NSDateComponents *dateComps = [[NSDateComponents alloc] init];
[dateComps setDay:[dateComponents day]];
[dateComps setMonth:[dateComponents month]];
[dateComps setYear:[dateComponents year]];
[dateComps setHour:[timeComponents hour]];
    // Notification will fire in one minute
[dateComps setMinute:[timeComponents minute]];
[dateComps setSecond:[timeComponents second]];
NSDate *itemDate = [calendar dateFromComponents:dateComps];

UILocalNotification *localNotif = [[UILocalNotification alloc] init];
if (localNotif == nil)
    return;
localNotif.fireDate = itemDate;
localNotif.timeZone = [NSTimeZone defaultTimeZone];

localNotif.alertBody = [self.EventText text];

localNotif.alertAction = @"View";

localNotif.soundName = UILocalNotificationDefaultSoundName;
localNotif.applicationIconBadgeNumber = 1;

    // Specify custom data for the notification
NSMutableDictionary *infoDict = [[NSMutableDictionary alloc] init];
[infoDict setObject:[self.EventText text] forKey:@"msg"];
[infoDict setObject:@"alarmclock" forKey:@"sender"];

    // NSDictionary *infoDict = [NSDictionary
        dictionaryWithObject:@"someValue"
        forKey:@"someKey"];
localNotif.userInfo = infoDict;

    // Schedule the notification
[[UIApplication sharedApplication]
scheduleLocalNotification:localNotif];

[self.mTableView reloadData];
}
- (void)didReceiveMemoryWarning
{

```

```

    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}
@end

```

In **Swift**, you can also add a local notification with a badge notification, sound notification, and alerts. A user notification can be delivered using `UILocalNotification`. For this, you first register for the notification in the `AppDelegate.Swift` class with the following code:

```

// registration for notification in AppDelegate.Swift
func application(application: UIApplication,
didFinishLaunchingWithOptions launchOptions: [NSObject: AnyObject]?) -> Bool
{
    application.registerUserNotificationSettings(UIUserNotificationSettings(forTypes:
        [UIUserNotificationType.Sound, UIUserNotificationType.Alert,
        UIUserNotificationType.Badge], categories: nil))
    return true
}

```

After registration, you add code in `ViewController.Swift` class file as follows:

```

var localNotif: UILocalNotification = UILocalNotification()
localNotif.fireDate = NSDate()
localNotif.timeZone = [NSTimeZone defaultTimeZone]
localNotif.alertBody = [self.EventText text]
localNotif.soundName = UILocalNotificationDefaultSoundName
localNotif.applicationIconBadgeNumber = 1
[[UIApplication sharedApplication] scheduleLocalNotification:
localNotif];

```

<H3>[Receiving a Notification]

Receiving a notification is usually handled within the application's delegate implementation. Simply implement the following method to create a `UIAlertView` displaying the message you have passed to the custom object.

```

- (void)application:(UIApplication *)app didReceiveLocalNotification:
    (UILocalNotification *)notif {
    // Handle the notification when the app is running

    UIAlertView* alert = [[UIAlertView alloc]
initWithTitle:@"Alert" message:[notif userInfo]
valueForKey:@"msg"] delegate:self cancelButtonTitle:@"Ok"

```



```

        otherButtonTitles:nil, nil];
        [alert show];
    }

```

When the application is launched from the notification, you have to process the notification in the `application:didFinishLaunchingWithOptions:` method by implementing the following code:

```

application.applicationIconBadgeNumber = 0;

// Handle launching from a notification
UILocalNotification *localNotif =
[launchOptions
objectForKey:UIApplicationLaunchOptionsLocalNotificationKey];
if (localNotif) {

    UIAlertView* alert = [[UIAlertView alloc]
                           initWithTitle:@"Alert"

                           message:[localNotif userInfo] valueForKey:@"msg"]

                           delegate:self
                           cancelButtonTitle:@"Ok"
                           otherButtonTitles:nil, nil];
    [alert show];
}

```

If a user opens the app on receiving a local notification, the `launchOptions` dictionary passes to the `application:willFinishLaunchingWithOptions:` method and the `application:didFinishLaunchingWithOptions:` method. These methods contain the `UIApplicationLaunchOptionsLocalNotificationKey` key.

In **Swift**, the format for `didReceiveLocalNotification` is as follows:

```

func application(application: UIApplication!,
didReceiveLocalNotification notification: UILocalNotification!) { ...
}

// Format for didFinishLaunchingWithOptions
func application(application: UIApplication!,
didFinishLaunchingWithOptions launchOption: [NSObject : AnyObject]?) -> Bool
{
    ....
}

```

Real Life Connect

Notifications have a wonderful use for applications. Currently, programmers implement notifications in almost every application, including messengers and social networking site applications. One of the best examples where notification is implemented is the WhatsApp application, where every time a message comes, you get a notification.

<H1>Understanding Push Notifications

Source: [Professional iOS Programming][10][341]

The Apple Push Notification Service (APNS) is the centerpiece in the architecture of sending external notifications to iOS devices.

A push notification consists of two elements:

- **Unique Identifier:** This is a unique identifier of the target device. It is also known as the device token.
- **Payload:** The payload is a JavaScript Object Notation (JSON)-defined property list that allows the receiving application to parse and present as desired.

Figure 3 illustrates the flow of a push notification.

You can find the complete architectural components and security flow of push notifications at <https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Introduction.html>.

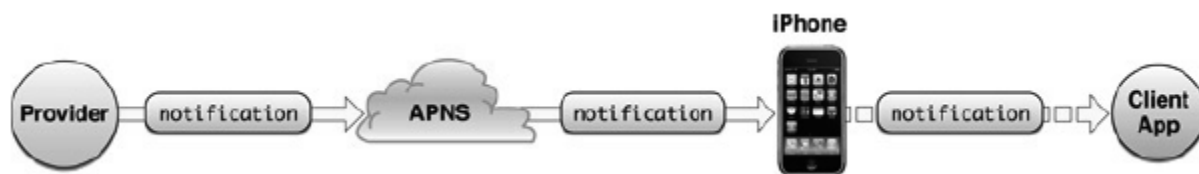


Figure 3: Flow of a Push Notification

During each project where push notifications are required, you first decide which provider to use and how to use it. The provider that initiates the sending of the push notification is presented as a process. You can develop your own custom push notification provider in your preferred programming language, or you can use a publicly available service like Urban Airship (<http://urbanairship.com/>) that provides you with a ready-to-use infrastructure and technology that is easy to implement using its iOS software development kit (SDK).

Quick Tip

iOS can only play a sound that is packaged and delivered in the App Store build.

In this section, you will learn how to implement Urban Airship push notifications in your iOS application.

Start **Xcode** and create a new project using the **Single View Application** template. Name the project as **PushDemo** using the options as shown in **Figure 4**.

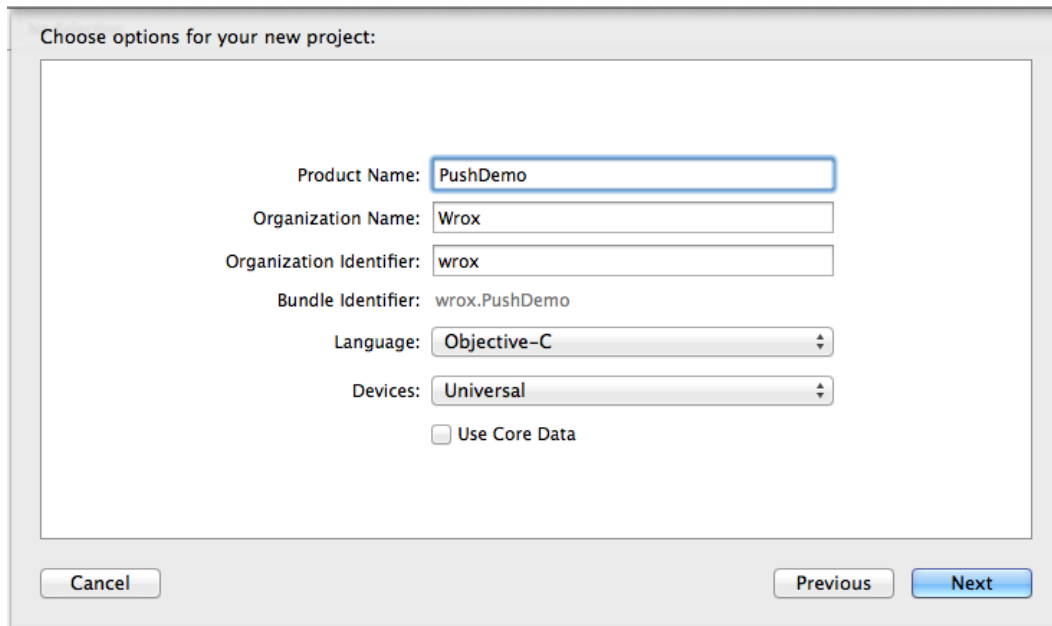


Figure 4: Create the PushDemo Project in Xcode

The full Urban Airship SDK requires you to add several frameworks and libraries to your project. Open the project target and select the **Build Phases** tab as shown in **Figure 5**. Under the **Link Binary with Libraries** section, add the following frameworks/libraries:

- MobileCoreServices.framework
- MapKit.framework
- MessageUI.framework
- Security.framework
- SystemConfiguration.framework
- CoreTelephony.framework
- CoreLocation.framework
- StoreKit.framework
- AudioToolbox.framework
- CFNetwork.framework

- Libsqlite3.dylib
- Libz.dylib

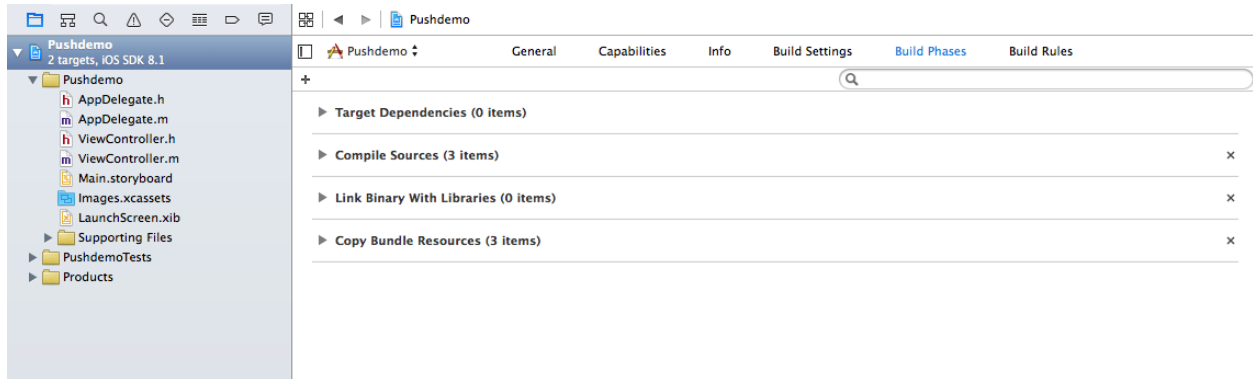


Figure 5: Build Phases View in Project

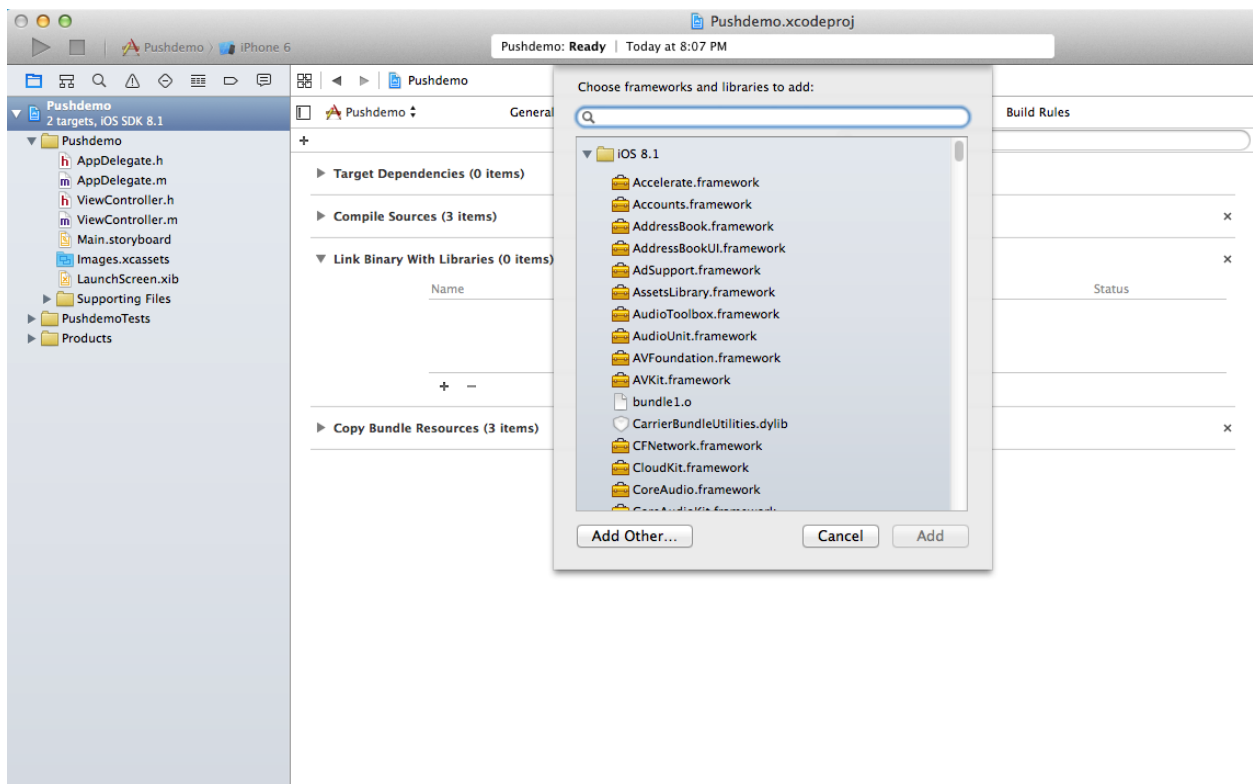


Figure 6: Link Binary with Libraries

These frameworks are written in Objective-C, so you need to import the frameworks in your **Swift** code. In **Swift** code, you need to add a bridging header for importing these files and frameworks with a product module name called "-Bridging-Header.h".

```
//format for importing in Objective-C
    @import FrameworkName
```

```
//format for importing in Swift
import FrameworkName
```

<H2> [Configuring the Developer Portal]

Log on to <http://developer.apple.com> and sign in using your credentials in the member center.

Once you are logged in, select the **Certificates, Identifiers & Profiles** option on the **Developer Program Resources** page, as shown in **Figure 7**.

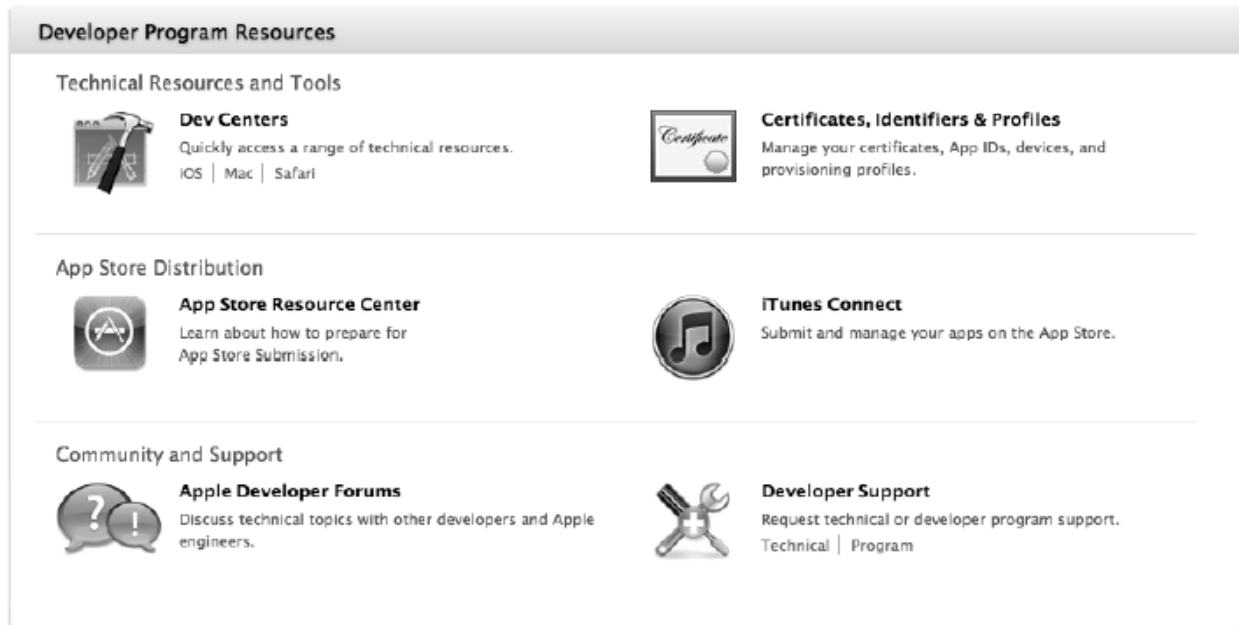


Figure 7: Select the Certificates, Identifiers & Profiles Option

On the next screen, select **Identifiers** within the **iOS Apps** section as shown in **Figure 8**.



Figure 8: Select the Identifiers Option

On the next page, register your new application **PushDemo** by selecting various options. Make sure you select the **Push Notifications** check box and use an explicit App ID with a unique bundle ID, such as **net.yourdeveloper.pushdemo**, as shown in Figure 9.

Quick Tip

To give an explicit App ID with a unique bundle ID name, you can use the reverse domain name style suffixed by the application name, for example **net.yourdeveloper.pushdemo**.

Name:

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

App Services

Select the services you would like to enable in your app. You can edit your choices after this App ID has been registered.

Enable Services:

☐ Data Protection

☐ Complete Protection

☐ Protected Unless Open

☐ Protected Until First User Authentication

☒ Game Center

☐ iCloud

☒ In-App Purchase

☐ Passbook

☒ Push Notifications

App ID Prefix

Value:

App ID Suffix

☒ **Explicit App ID**

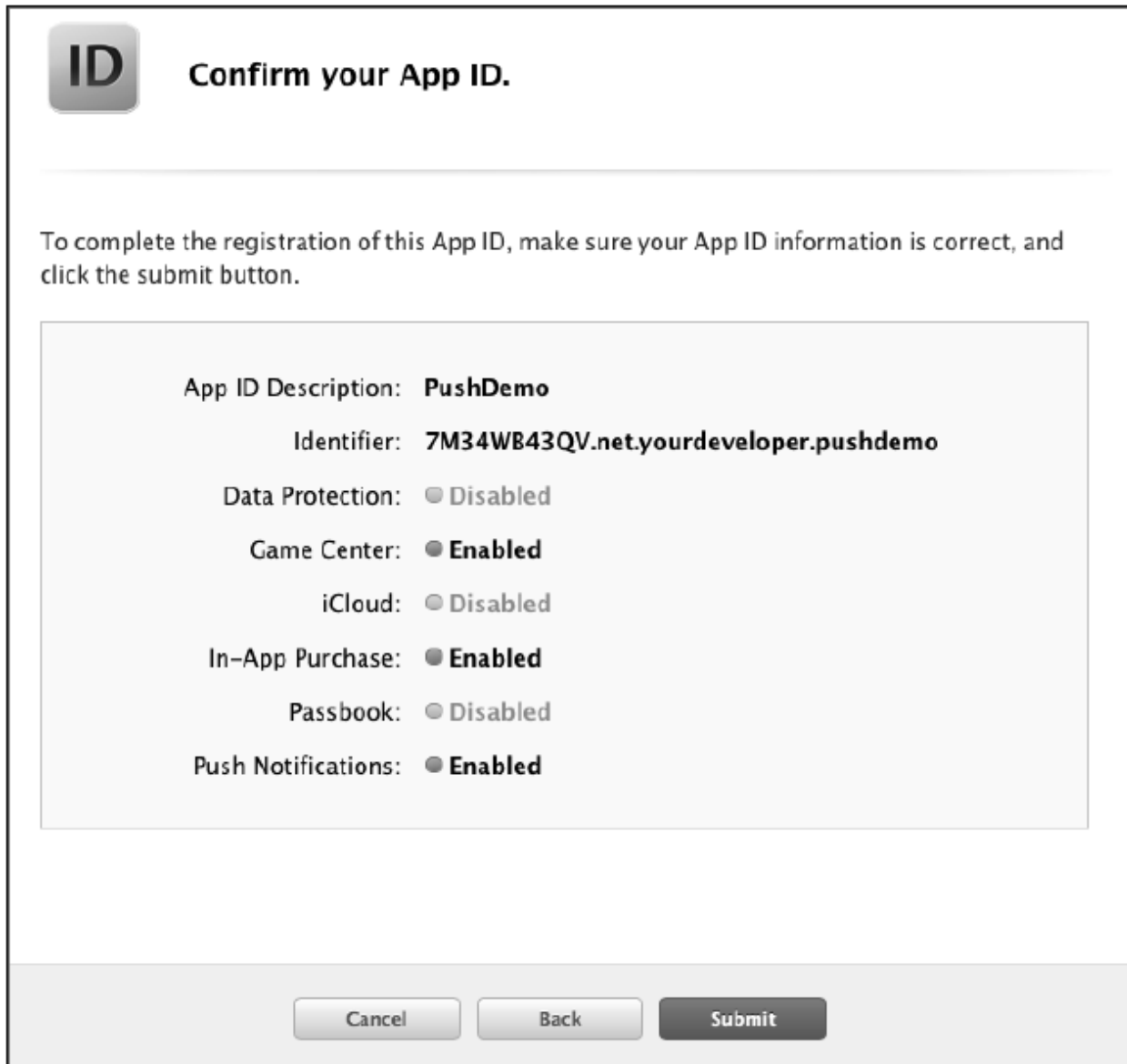
If you plan to incorporate app services such as Game Center, In-App Purchase, Data Protection, and iCloud, or want a provisioning profile unique to a single app, you must register an explicit App ID for your app.

To create an explicit App ID, enter a unique string in the Bundle ID field. This string should match the Bundle ID of your app.

Bundle ID:

Figure 9: Register Your New Application

Follow the instructions on the screen and confirm your application submission. The confirmation screen will summarize the options you have selected and display the identifier for your application as shown in **Figure 10**.



ID **Confirm your App ID.**

To complete the registration of this App ID, make sure your App ID information is correct, and click the submit button.

App ID Description: **PushDemo**

Identifier: **7M34WB43QV.net.yourdeveloper.pushdemo**

Data Protection: ☐ Disabled

Game Center: ☒ **Enabled**

iCloud: ☐ Disabled

In-App Purchase: ☒ **Enabled**

Passbook: ☐ Disabled

Push Notifications: ☒ **Enabled**

Figure 10: Confirm your App ID

When you click the **Submit** button as shown in **Figure 10**, your application will be registered. Now, if you select the application line from the grid, it will be displayed as shown in **Figure 11**.

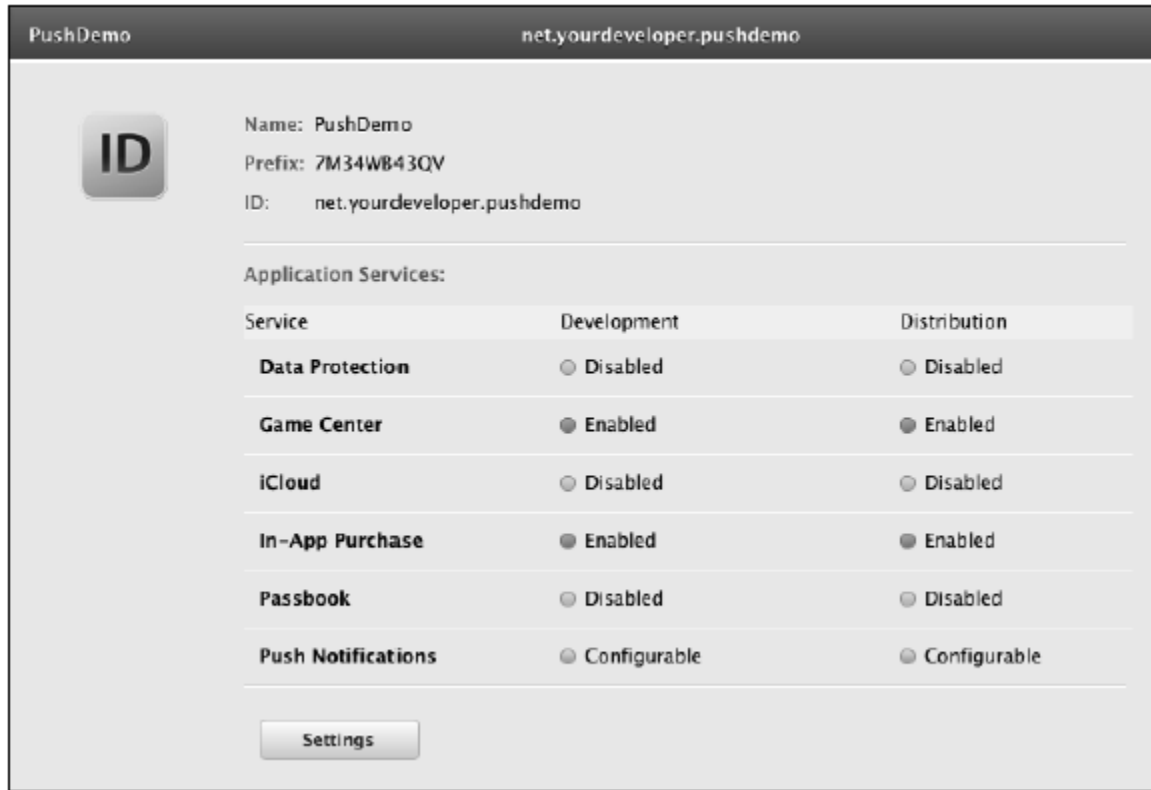


Figure 11: Registered Application Screen

<H2> [Obtaining Certificates]

Communication between the APNS and the iOS device is a secure connection. Therefore, you need to obtain certificates for the following purposes:

- To sign your application
- To make sure that the Urban Airship configuration (or your customer push notification service provider) is able to sign the request when sending the payload to the server

To obtain certificates, click the **Settings** button on the listed application screen, as shown in **Figure 11**. This will display the **Push Notifications** screen, as shown in **Figure 12**. On this screen, you need to create the Development SSL Certificate and the Production SSL Certificate.

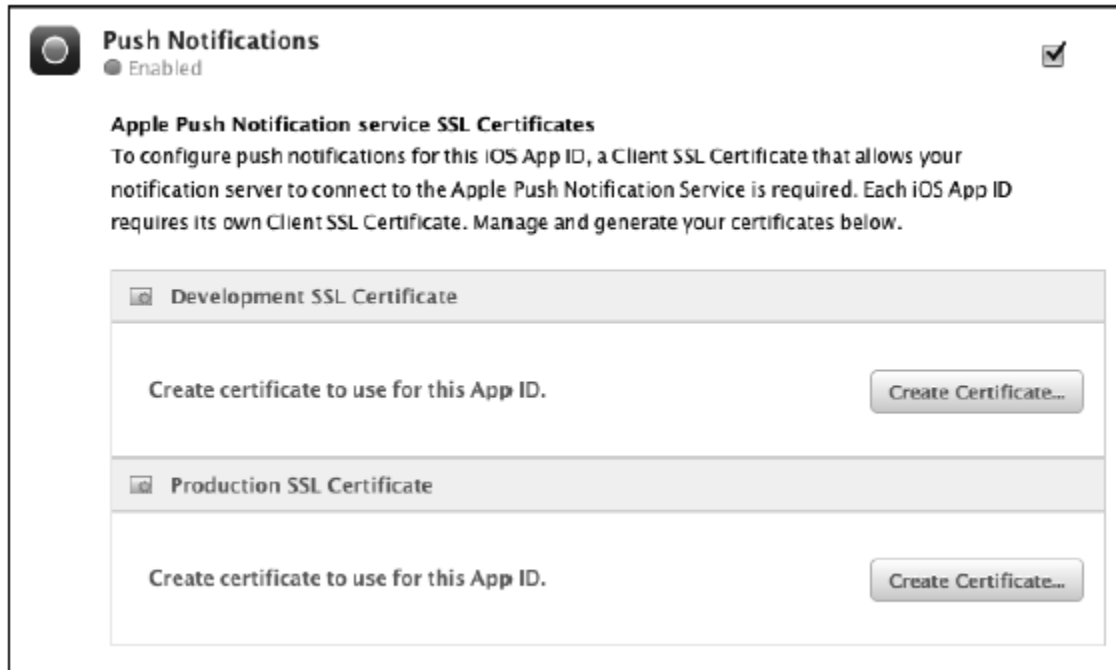


Figure 12: Create Development SSL Certificate and Production SSL Certificate

To create the **Development SSL Certificate** and the **Production SSL Certificate**, click the corresponding **Create Certificate** button.

This will open the **About Creating a Certificate Signing Request (CSR)** screen as shown in **Figure 13**. Follow the instructions given on the screen.

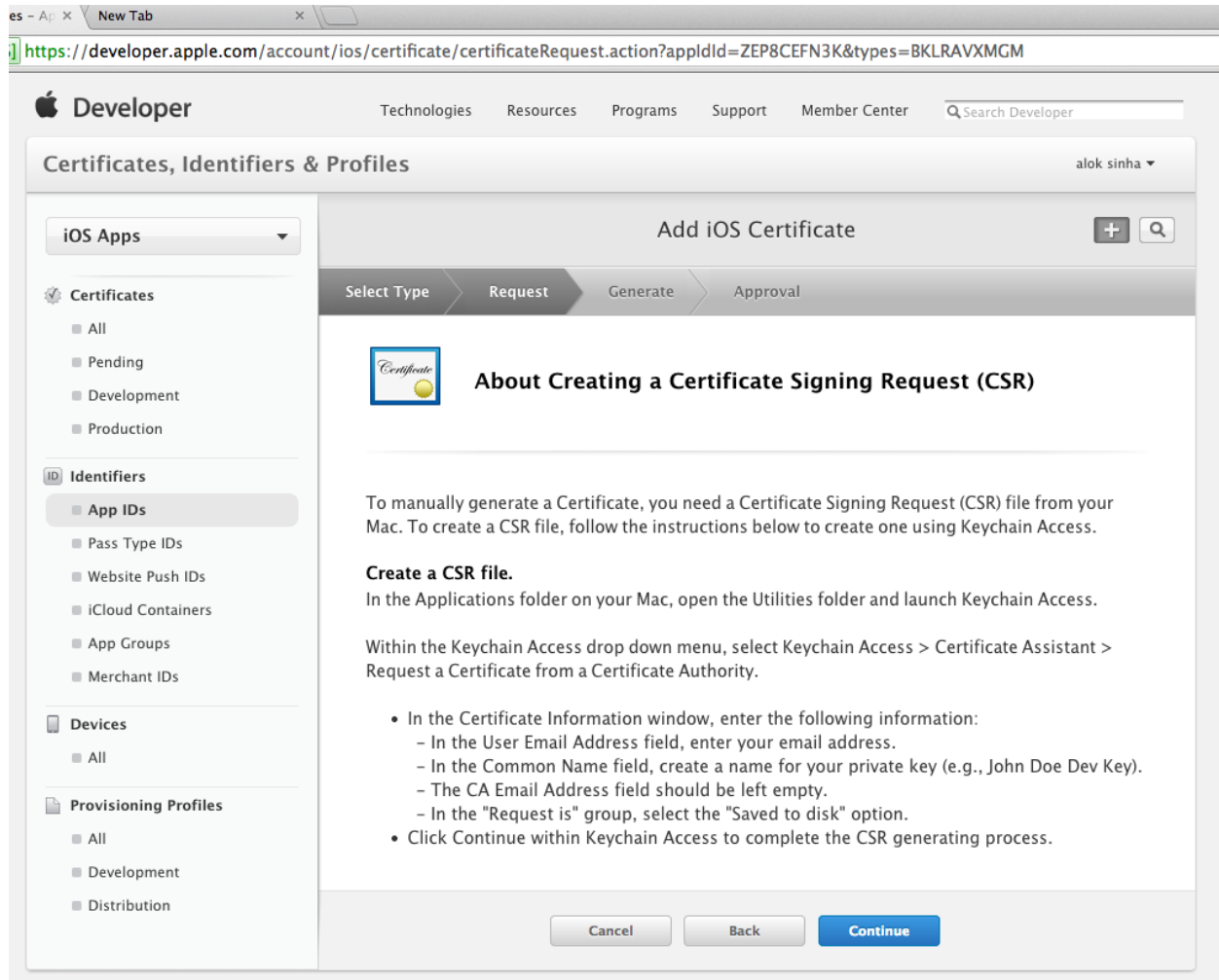


Figure 13: Follow Steps to Generate a CSR File

In the **Applications** folder, open the **Utilities** folder and search for **Keychain Access** as shown in **Figure 14** and select it.

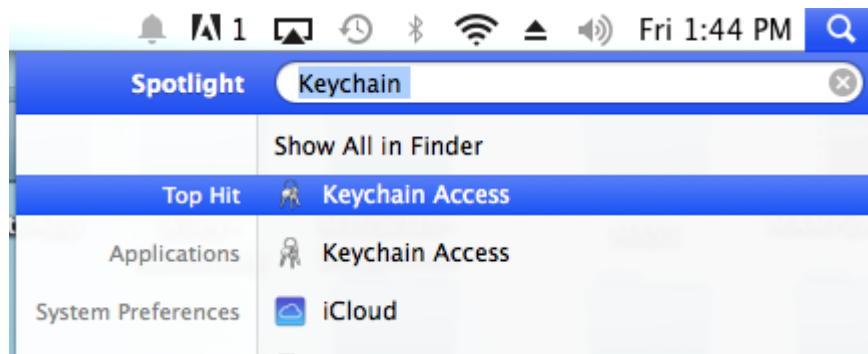


Figure 14: Search for Keychain Access

To request a certificate, select **Keychain Access** → **Certificate Assistant** → **Request a Certificate from a Certificate Authority** as shown in **Figure 15**.

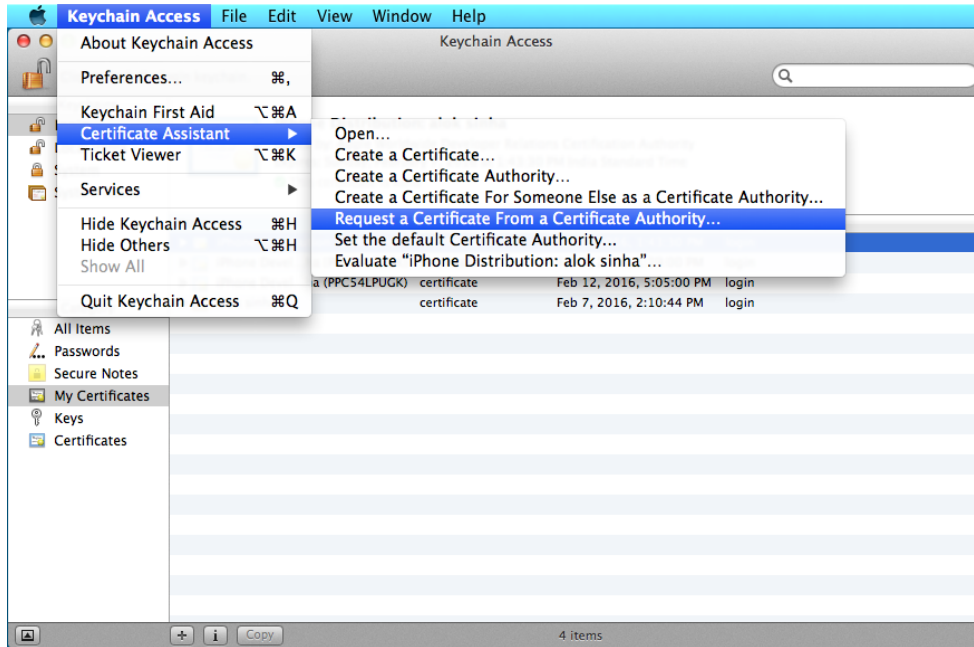


Figure 15: Request for Certificate File in Keychain Access

On the **Certificate Assistant** screen, enter your email address in the **User Email Address** field and a name for your private key in the **Common Name** field. Choose the **Saved to disk** option and click **Continue** as shown in **Figure 16**.

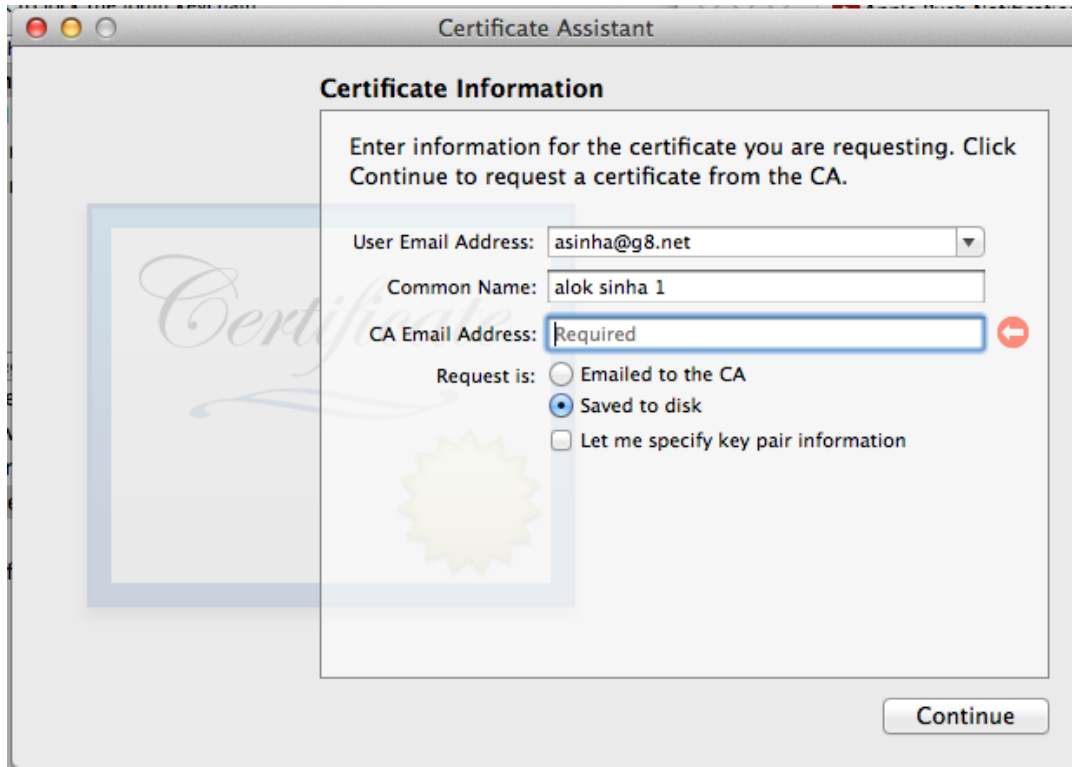


Figure 16: Specify Certificate Information and Click Continue to Create a CSR File

In the dialog box that opens, name your certificate generated, select the folder to save this certificate, and click **Save** as shown in **Figure 17**.

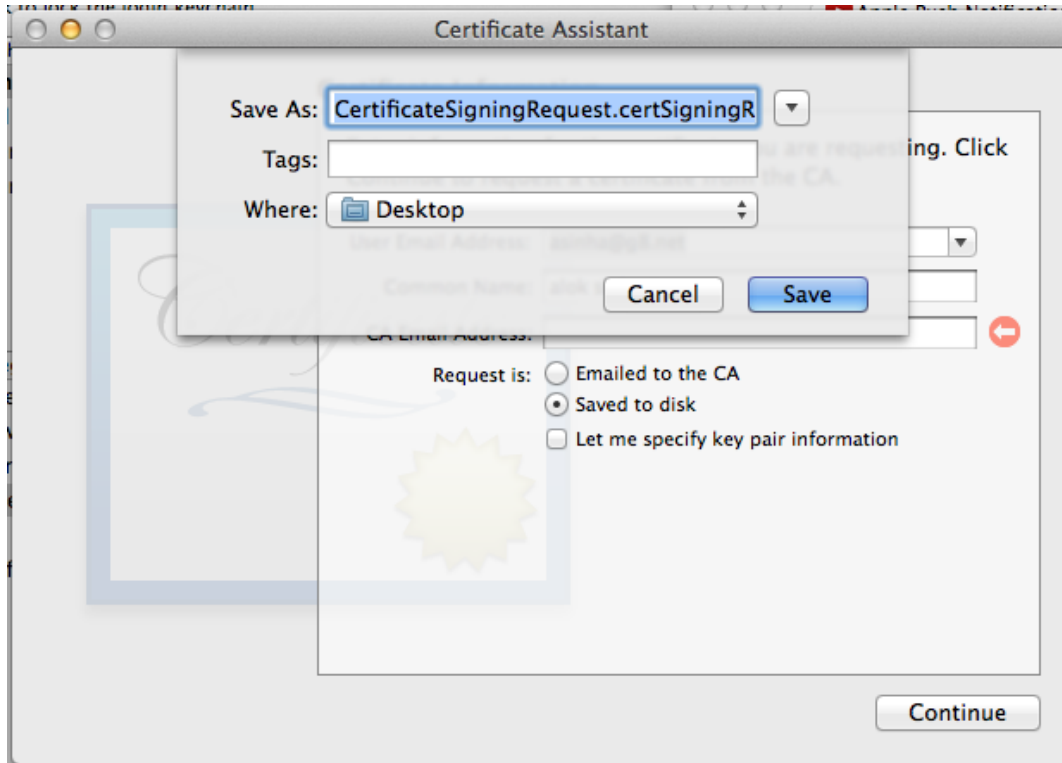


Figure 17: Choose Folder to Save This Certificate and Click Continue

The certificate is now generated and saved on the desktop. Now select the CSR file generated on the desktop by clicking the **Choose File** button as shown in **Figure 18**. After choosing the CSR file, click the **Generate** button.

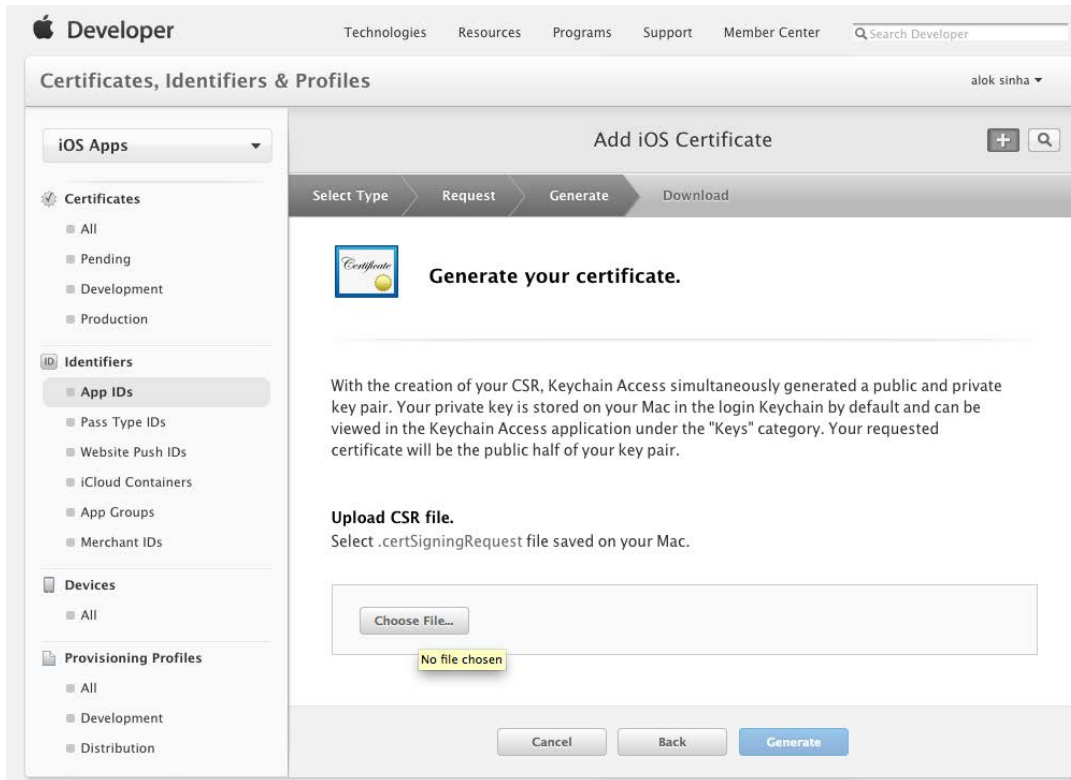


Figure 18: Choose the CSR File Generated

Finally, on the **Your certificate is ready** screen, click the **Download** button to download the certificate as shown in **Figure 19**.

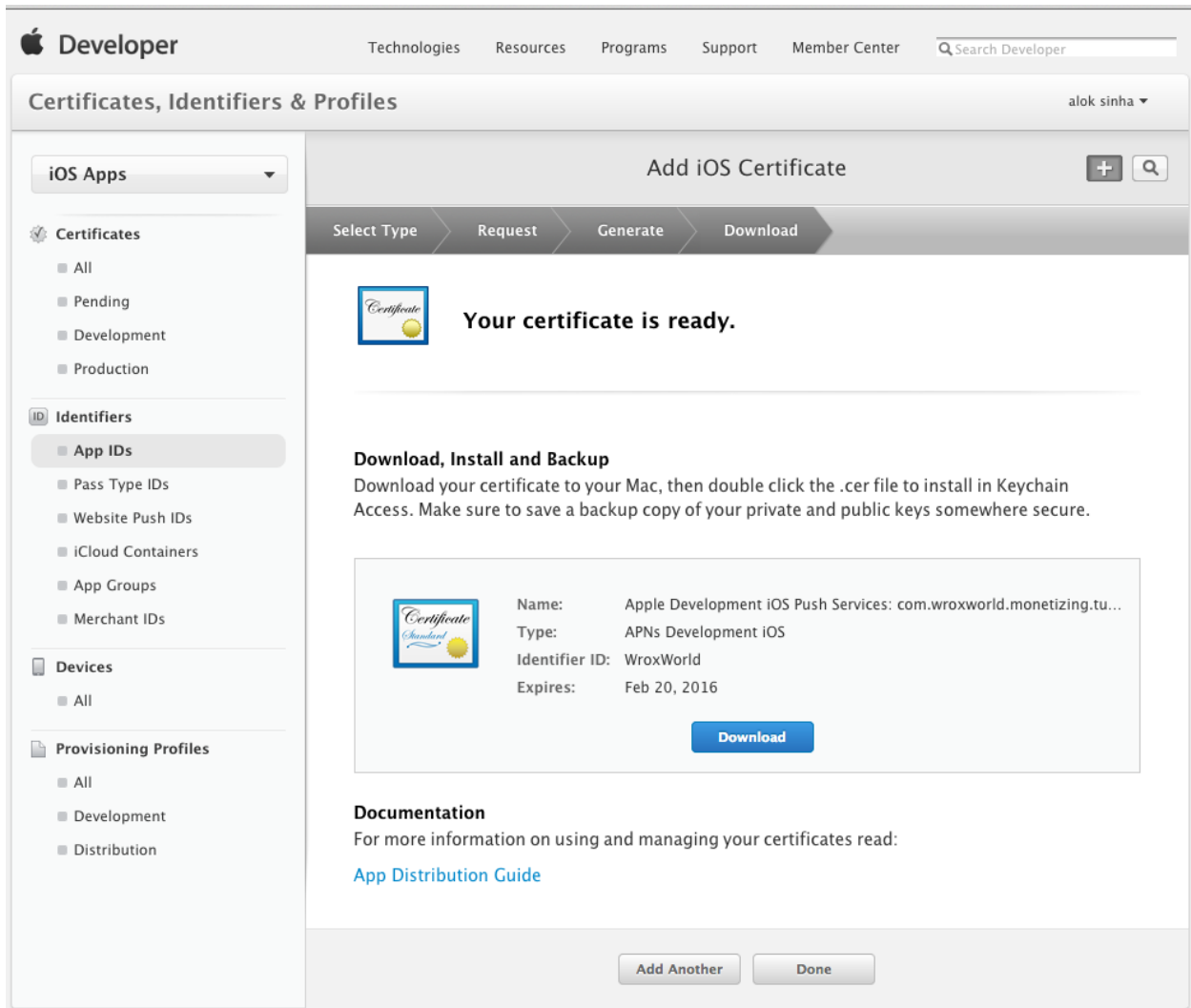


Figure 19: Download the Generated Certificate

After you have created both Development SSL Certificate and Production SSL Certificate, your screen will look like **Figure 20**, allowing you to download both certificates.

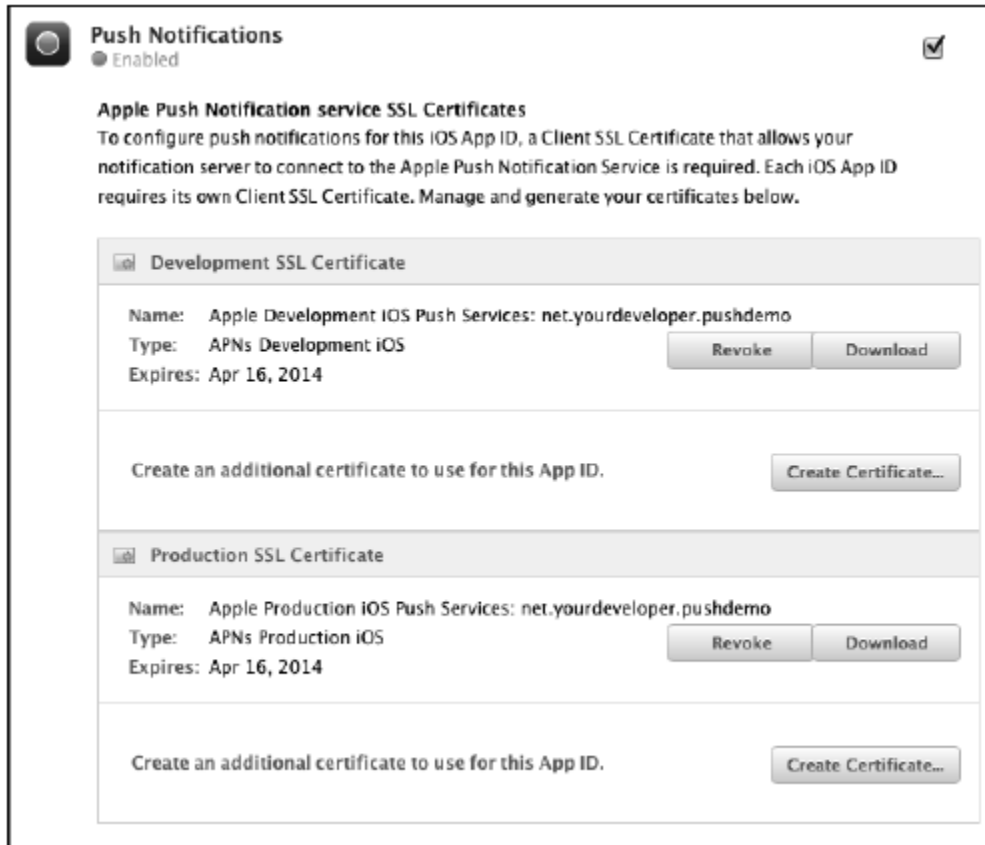


Figure 20: Download Certificates

<H2>[Locating and Saving Certificates]

After downloading these certificates, you can use the **Finder** folder in your Mac to locate them and move them to a folder where you keep track of all certificates.

For example, you have an **aps_development.cer** certificate and an **aps_production.cer** certificate. Use **Finder** and click both certificates to have them automatically installed in the keychain access as shown in **Figure 21**.

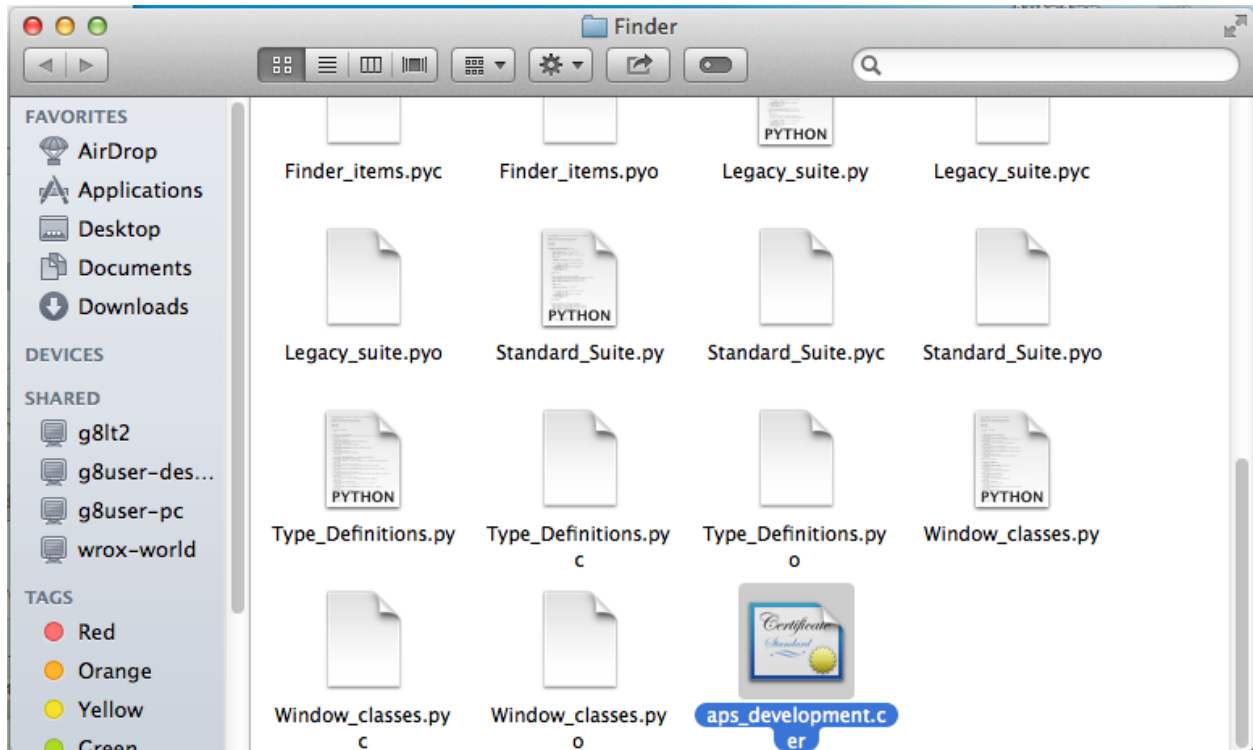


Figure 21: Use Finder to Find Downloaded Certificates

Open **Keychain Access** and select **My Certificates** under the **Category** tab as shown in **Figure 22**. You will see a certificate in the right-hand pane with the name **Apple Development iOS Push Services: xxxxxx**, where **xxxxxx** is the name of your application.

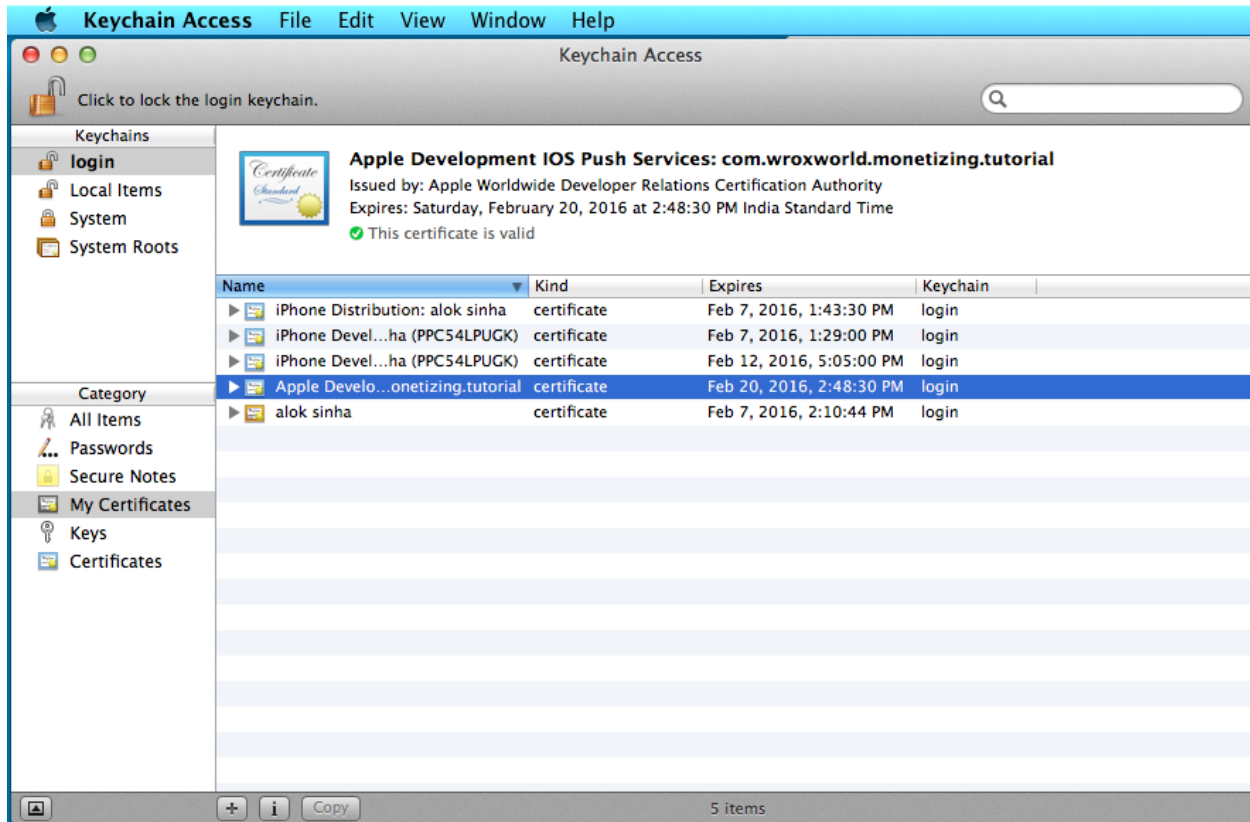


Figure 22: Certificates Saved in Keychain Access

Select the certificate you need (development or production) in the right pane. Right-click on the selected certificate and choose **Export “Certificate Name”** from the right-click menu, as shown in **Figure 23**. Alternatively, you can choose **File** ⇨ **Export** from the menu.

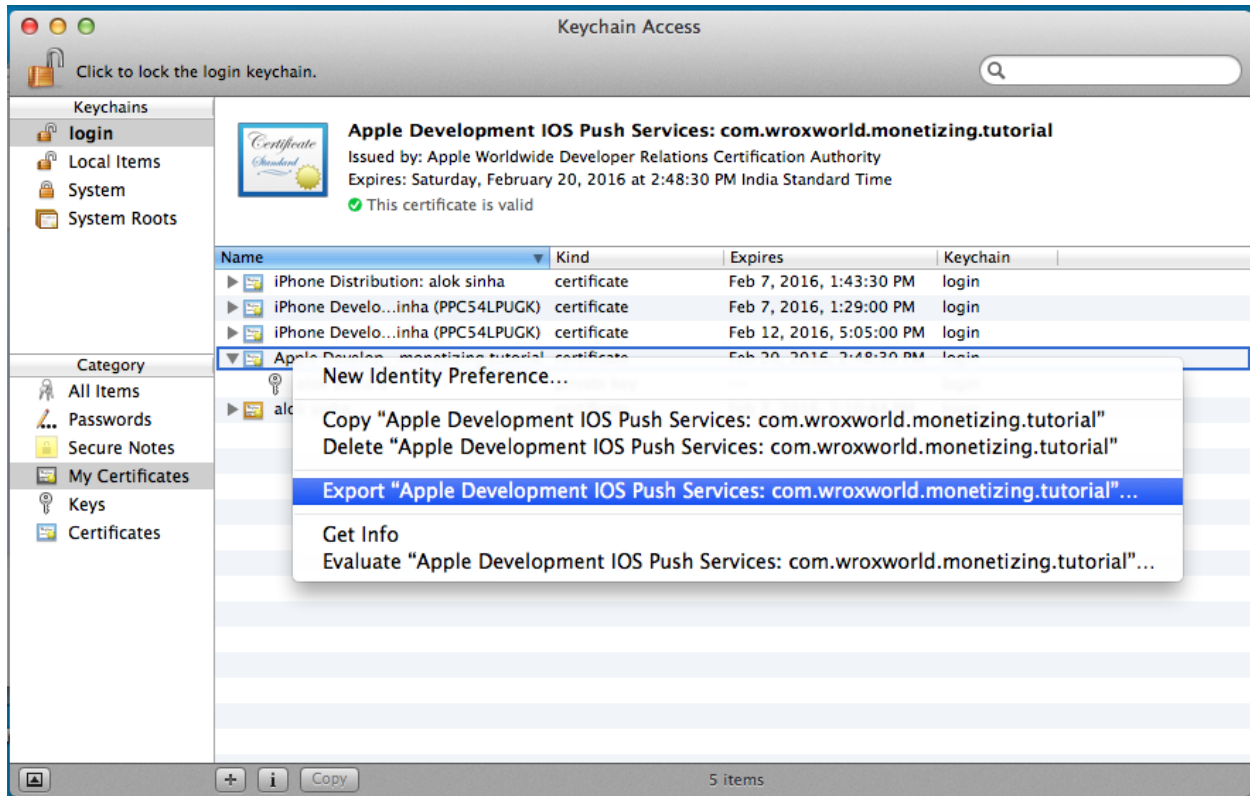


Figure 23: Export the Certificate Generated

In the pop-up screen that appears, you can enter the name of the certificate and choose the location where you want to import it. Choose **Personal Information Exchange (.p12)** as the **File Format** and click **Save**, as shown in **Figure 24**.

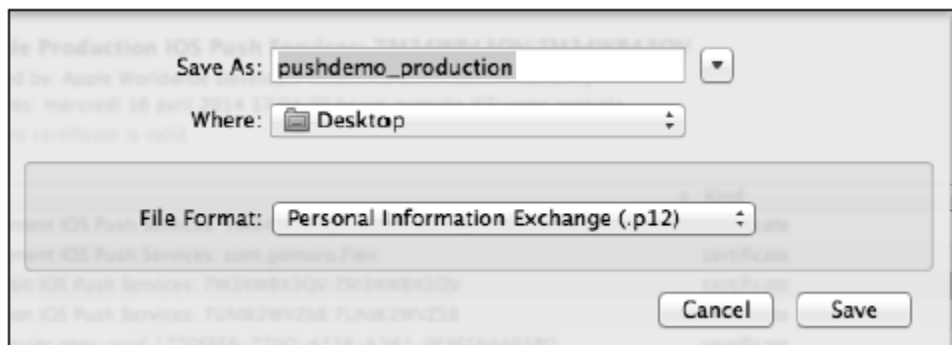


Figure 24: Save the Certificate to Import

In the dialog box that opens, enter a strong password, verify it, and click **OK** as shown in **Figure 25**.



Figure 25: Enter a Password to Protect Certificates

Make sure the .p12 certificates are also copied in the folder where you keep the certificates so they are all in one place.

<H2>[Implementation with Urban Airship]

This section describes the steps to implement push notifications using Urban Airship as the provider.

<H3>[Creating a Free Account]

To create a free account, follow the link <https://go.urbanairship.com/accounts/register/trial/>. Follow the instructions on the website to register and gain access to the developer portal. Use the credentials you received/created to log on to the developer portal.

Go to **Apps** (Your Applications) and select **+ New App** to add a new application. Follow the instructions on the screen, name your application, upload an icon, and specify whether this configuration is for development or production.

During the configuration of the Urban Airship services, select **Apple Push Notification Service** and click the **Configure** button. The system will ask you to upload a certificate (this is the exported .p12 certificate you created earlier) and enter the certificate password you used during the export of this certificate. Enter the correct password, upload the certificate, and click **Save**. Your configuration in Urban Airship is now done.

Under **Settings** you can find an option named **API Keys**. When you click it, you will find an App Key, an App Secret, and an App Master Secret. You need to copy the first two because you need them in the configuration.

To send test messages after you have finalized the implementation of your application, you can use the submenus available under the application you created earlier. It is outside the scope of this session to explain in detail which capabilities are available with Urban Airship.

<H3>**[Downloading the SDK]**

The exact location of the download link to the SDK has changed various times in the past, but a steady factor is a higher-level page you can find at <http://urbanairship.com/resources/developer-resources>. Find the latest stable iOS SDK library and download it from there, following the latest instructions on the website.

<H3>**[Implementing the SDK]**

To implement the SDK, simply drag and drop the complete Urban Airship folder structure you have downloaded into your project structure, as shown in **Figure 26**.

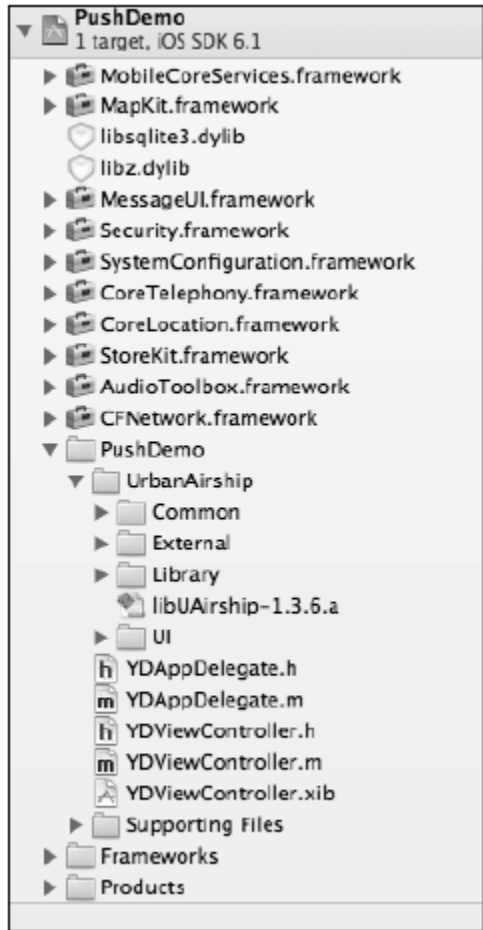


Figure 26: Drag the Urban Airship Folder Structure into PushDemo Project Structure

In the **Supporting Files** group, add a new plist file named `AirshipConfig.plist` and create the settings as shown in **Figure 27**. Use the App Key and App Secret you found in the Urban Airship portal.

Key	Type	Value
▼ Root	⊕ Dictionary ⌵ (5 items)	
APP_STORE_OR_AD_HOC_BUILD	String	YES
DEVELOPMENT_APP_KEY	String	
DEVELOPMENT_APP_SECRET	String	
PRODUCTION_APP_KEY	String	
PRODUCTION_APP_SECRET	String	

Figure 27: Add plist File into Supporting Files

<H2> [Enhancing Your AppDelegate]

The following code snippet shows the implementation of the push notification logic using the Urban Airship provider library. After importing the Urban Airship library, the `application:didFinishLaunchingWithOptions:` method is extended with the `UAPush` initialization code.

The `registerForRemoteNotificationTypes:` response method calls the `application:didRegisterForRemoteNotificationsWithDeviceToken:` method and the device token is sent to Urban Airship.

If the APNS server is sending a payload to this device, the `application:didReceiveRemoteNotification:handleNotification:applicationState:` method is called by sending the payload to the device token. You can choose to let the Urban Airship SDK manage the notification, or you can write your own logic to process the information that is sent as payload in the `UserInfo` object.

```
#import "YAppDelegate.h"

#import "YDViewController.h"
#import "UAirship.h"
#import "UAPush.h"
#import "UAUtils.h"
@implementation YAppDelegate

-(BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:
(NSDictionary *)launchOptions
{
    self.window = [[UIWindow alloc] initWithFrame:[UIScreen mainScreen]
    bounds]];

    // the app is launched from a push notification
NSMutableDictionary *takeOffOptions = [[NSMutableDictionary alloc] init];
    [takeOffOptions setValue:launchOptions forKey:
    UAirshipTakeOffOptionsLaunchOptionsKey];
    [UAPush setDefaultPushEnabledValue:YES];
    [UAirship takeOff:takeOffOptions];

    [[UAPush shared] resetBadge]; //zero badge on startup
```



```

        [[UAPush shared]
registerForRemoteNotificationTypes:
    (UIRemoteNotificationTypeBadge |
    UIRemoteNotificationTypeSound |
    UIRemoteNotificationTypeAlert)];
    // Override point for customization after application launch.
    self.viewController = [[YDViewController alloc]
        initWithNibName:
@"YDViewController" bundle:nil];
    self.window.rootViewController = self.viewController;
    [self.window makeKeyAndVisible];
    return YES;
}

    // Implement the iOS device token registration callback
- (void)application:(UIApplication *)application
didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken {

    [[UAPush shared] registerDeviceToken:deviceToken];
}
- (void)application:(UIApplication *)application
didReceiveRemoteNotification:
(NSDictionary *)userInfo {
    [[UAPush shared] handleNotification:userInfo
    applicationState:application.applicationState];
    // Reset the badge if you are using that functionality
    [[UAPush shared] resetBadge]; // zero badge after push received
}

```

Once your device token is registered, you can log in to Urban Airship and send a test push notification message. It is important to realize that push notifications do not work on the iOS Simulator and will not work in real devices if the project is not signed with an ad-hoc or production profile.

Quick Tip

Push notifications services are of great advantage for developers. They provide better means to get updated and save battery life rather than running background processes.

<H1>[External Notifications]

If you want your application to be launched or accessible by other iOS applications on the user's device, you can use uniform resource locator (URL) schemes.

For example, you want to develop an application where the user can record a movie and send an email to a friend with a link to share this movie. A custom URL scheme is used in the email. Therefore, when the URL is opened on an iPad or iPhone where the application is installed, the recorded movie opens and plays in the application and the back-end management system is updated with relevant information from the receiving user.

To develop this application, start Xcode and create a new project using the **Single View Application** template, and name it `UrlSchemeDemo`. In this small project, you will implement a custom URL scheme and the logic to make use of it in your own applications.

<H2>[Defining a Custom URL Scheme]

To define a custom URL scheme, navigate to the `UrlSchemeDemo-Info.plist` file and open it. To create a custom URL scheme, you start by adding a row (right-click and choose **Add Row**) and selecting **URL Types** from the drop-down list. Navigate to the created node, and under **Item 0** add a new row of type **URL Schemes**, which is an array of items. At **Item 0**, create the identifier you would like to use, for example `urldemo`, as shown in **Figure 28**.

Key	Type	Value
▼ Information Property List	Dictionary	(15 items)
Application fonts resource path	String	
Localization native development region	String	en
Bundle display name	String	\$(PRODUCT_NAME)
Executable file	String	\$(EXECUTABLE_NAME)
Bundle identifier	String	net.yourdeveloper.\$(PRODUCT_NAME:rfc1034identifier)
InfoDictionary version	String	6.0
Bundle name	String	\$(PRODUCT_NAME)
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.0
Bundle creator OS Type code	String	????
Bundle version	String	1.0
Application requires iPhone environment	Boolean	YES
► Required device capabilities	Array	(1 item)
▼ URL types	Array	(1 item)
▼ Item 0	Dictionary	(1 item)
▼ URL Schemes	Array	(1 item)
Item 0	String	urldemo
► Supported interface orientations	Array	(3 items)

Figure 28: Create the Identifier for Custom URL Scheme

This will create a custom URL scheme with an identifier named `urldemo`. This means that you can trigger your application by calling `urldemo://xxxxxxx`, where `xxxxxxx` is free and `urldemo:` is the identifier you have created.

<H3>[Responding to the URL Request]

The `UIApplication` has a delegate method named `application:openURL:sourceApplication:annotation:` that allows you to write your custom processing code to respond to the URL request.

The following code shows a sample implementation of this method in the `YDAppDelegate.m` file:

```
- (BOOL)application:(UIApplication *)application
    openURL:(NSURL *)url
    sourceApplication:(NSString *)sourceApplication
    annotation:(id)annotation {
    //You can perform an additional check to make sure you're not picking up an
    //url identifier from another application
    if ([[url host] isEqualToString:@"net.yourdeveloper.somevalue"])
    {
        //parse the query string if used to get access
        //to the query string variables and values
        NSArray *querySplitter = [[url query]
            componentsSeparatedByString:@"&"];
        if ([querySplitter count] == 1)
        {
        }
    }
    return YES;
}
```

Set a breakpoint in the above method and run your application. Press the **Home** button and launch Safari on your device or simulator, and your application will keep running in the background.

To enable your application to enter the foreground mode and stop execution at the set breakpoint, open Safari, type `urldemo://net.yourdeveloper/whatever` in the URL bar, and press **Enter**.

Within this method, you can verify the URL that has been passed, access the query string to parse the values and keys that have been passed, and implement your process based on the receiving values.

By using custom URL schemes, you can provide functionality to other applications to launch your applications, and even pass information using query strings in the URL.

Big Picture

External notifications enable you to send a link to download software, songs, movies, and games. You can implement this link in your application so that while playing a game app, you can invite your friends and choose opponents and play simultaneously.

QUICK TIP

It is convenient to create a folder containing all your certificates by storing them in a subfolder with the application name.

Lab Connect

During the lab hour of this session, you will be able to create an application for local notifications.

Cheat Sheet

- Local notifications are notifications that are generated and received by your application. The purpose of the local notification is to inform your application's user that something interesting for him happened while the application was not running in the foreground.
- The main characteristics of a local notification are:
 - They are launched and delivered by iOS on the same device.
 - They can be scheduled by date and time.
 - They can be created instantly based on an event.
 - They can include an `NSDictionary` with custom data.
 - They can contain a sound to play, an application badge number, and the title of the action button.
- A push notification consists of two elements. One is the unique identifier of the target device, also known as the device token, and the other is the payload.
- Because the communication between the Apple Push Notification Server and the iOS device is a secure connection, you need to obtain certificates. Certificates enable you to not only sign your application but also make sure that the Urban Airship configuration, or your customer Push Notification Service Provider, is able to sign the request when sending the payload to the server.
- If you want your application to be accessible or launched by other iOS applications on the user's device, you can use URL schemes.