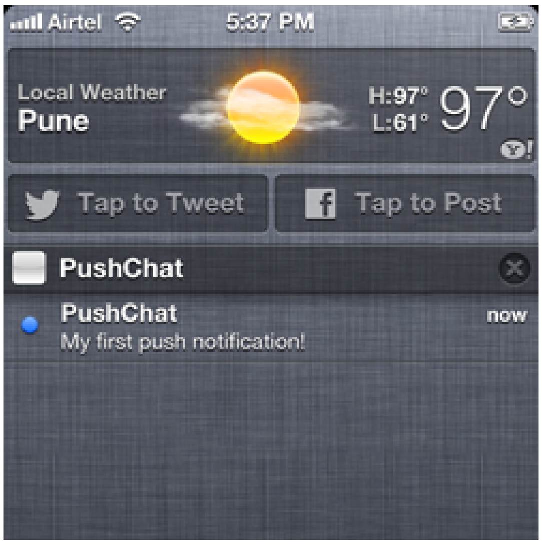
Apple Push Notification Services in iOS 6 Tutorial: Part 1/2

Ali Hafizji on May 23, 2013



PushChat
My first push notification!

Learn how to add Push Notifications into your iPhone app!

Update 4/12/2013: Fully updated for iOS 6 (original post byMatthijs Hollemans, update by Ali Hafizji).

In iOS, apps can't do a lot in the background. Apps are only allowed to do limited set of activities so battery life is conserved.

But what if something interesting happens and you wish to let the user know about this, even if they're not currently using your app?

For example, maybe the user received a new tweet, their favorite team won the game, or their dinner is ready. Since the app isn't currently running, it cannot check for these events.

Luckily, Apple has provided a solution to this. Instead of your app continuously checking for events or doing work in the background, you can write a server-side component to do this instead.

And when an event of interest occurs, the server-side component can send the app a push notification! There are three things a push notification can do:

- Display a short text message
- Play a brief sound
- Set a number in a badge on the app’s icon

You can combine these however you see fit; for example, play a sound and set the badge but not display a message.

In this 2-part tutorial series, you’ll get to try this out for yourself by making a simple app that uses APNS (Apple push notification service)!

In this first part, you’ll learn how to configure your app to receive push notifications and receive a test message.

This tutorial is for intermediate or advanced iOS developers. If you are still a beginner to iOS, you should check out some of the other tutorials on this site first. Also, it’s highly recommended that you review these two tutorials first (or have equivalent knowledge):

- How To Write A Simple PHP/MySQL Web Service for an iOS App
- How To Write an iOS App That Uses A Web Service

Without further ado, let’s push through this!

Getting Started: Brief Overview

Getting push to work for your app takes quite a bit of effort. This is a puzzle with many pieces. Here is an overview:

1. An app enables push notifications. The user has to confirm that he wishes to receive these notifications.
2. The app receives a “device token”. You can think of the device token as the address that push notifications will be sent to.
3. The app sends the device token to your server.
4. When something of interest to your app happens, the server sends a push notification to the Apple Push Notification Service, or APNS for short.
5. APNS sends the push notification to the user’s device.

When the user’s device receives the push notification, it shows an alert, plays a sound and/or updates the app’s icon. The user can launch the app from the alert. The app is given the contents of the push notification and can handle it as it sees fit.

Are push notifications still worth it now that we have local notifications and multitasking? You bet!

Local notifications are limited to scheduling timed events and unlimited background processing is only available to apps that do VOIP, navigation or background audio. If you want to notify the users of your app about external events while the app is closed, you still need push notifications.

In this tutorial, I will explain in detail how the push notification system works and how to build push into your own app. There is a lot to explain, so take your time to let it all sink in.

What You Need for Push Notifications

To add push notifications to your app, you need:

An iPhone or iPad. Push notifications do not work in the simulator, so you will need to test on the device.

An iOS Developer Program membership. You need to make a new App ID and provisioning profile for each app that uses push, as well as an SSL certificate for the server. You do this at the iOS Provisioning Portal.

If you want to follow along with the examples in this tutorial, you will need to create your own provisioning profile and SSL certificate; you cannot use mine. Because it's important to get the certificate right, I'll explain the detailed steps on how to obtain one.

A server that is connected to the internet. Push notifications are always sent by a server. For development you can use your Mac as the server (which you'll do in this tutorial) but for production use, you need at least something like a VPS (Virtual Private Server).

A cheap shared hosting account is not good enough. You need to be able to run a background process on the server, install an SSL certificate, and be able to make outgoing TLS connections on certain ports.

Most shared hosting providers do not let you do this, although they might if you ask. However, I really recommend using a VPS host such as Linode.

Anatomy of a Push Notification

Your server is responsible for creating the push notification messages, so it's useful to know what they look like.

A push notification is a short message that consists of the device token, a payload, and a few other bits and bytes. The payload is what you are interested in, as that contains the actual data you will be sending around.

Your server should provide the payload as a JSON dictionary. The payload for a simple push message looks like this:

```
{
        "aps":
        {
                "alert": "Hello, world!",
                "sound": "default"
        }
}
```

For the JSON uninitiated, a block delimited by curly { } brackets contains a dictionary that consists of key/value pairs (just like an NSDictionary).

The payload is a dictionary that contains at least one item, “aps”, which itself is also a dictionary. In our example, “aps” contains the fields “alert” and “sound”. When this push notification is received, it shows an alert view with the text “Hello, world!” and plays the standard sound effect.

There are other items you can add to the “aps” section to configure the notification. For example:

```
{
        "aps":
        {
                "alert":
                {
                        "action-loc-key": "Open",
                        "body": "Hello, world!"
                },
                "badge": 2
        }
}
```

Now “alert” is a dictionary of its own. The “action-loc-key” provides an alternative text for the “View” button. The “badge” field contains the number that will be shown on the application icon. This notification will not play a sound.

There are many more ways to configure the JSON payload. You can change the sound that is played, you can provide localized text, and you can add fields of your own. For more information, check out the official Local and Push Notification

Programming Guide.

Push notifications are intended to be small; the payload size can be no more than 256 bytes. That leaves you about as much room as fits in an SMS message or a tweet. A smart push server won't waste space on newlines and whitespace and generates something that looks like:

```
{"aps":{"alert":"Hello, world!","sound":"default"}}
```

It's less easy to read for us humans, but it saves enough bytes to make it worth it. Push notifications whose payload exceeds 256 bytes will not be accepted by APNS.

Push Notification Gotchas

Push Notifications Are Unreliable!

They are not reliable! There is no guarantee that push notifications will actually be delivered, even if the APNS server accepted them.

As far as your server is concerned, push notifications are fire-and-forget; there is no way to find out what the status of a notification is after you've sent it to APNS. The delivery time may also vary, from seconds up to half an hour.

Also, the user's iPhone may not be able to receive push notifications all the time. They could be on a WiFi network that does not allow connections to be made to APNS because the required ports are blocked. Or the phone could be turned off.

APNS will try to deliver the last notification it received for that device when it comes back online, but it will only try for a limited time. Once it times out, the push notification will be lost forever!

After looking at the APNS Server Bill

They can be expensive! Adding push functionality to your app is fairly easy and inexpensive if you own the data, but can be expensive if you have a lot of users or data you need to poll.

For example, it's no problem if you want to notify your users when the contents of your own RSS feed change. Because you control that RSS feed and know when it changes — when you update the content on your web site — your server can send out the notifications at the right moment.

But what if your app is an RSS feed reader that allows users to put in their own URLs? In that case you need to come up with some mechanism to detect updates to those feeds.

In practice this means your server will need to continuously poll those feeds for changes. If you have a lot of users, you may have to install a bunch of new servers to handle all that processing and bandwidth. For apps such as these, push can become quite expensive and may not be worth it.

OK, enough theory. It's time to learn how to do this push thing. Before we can get to the good stuff – programming! – there is some boring set-up work to be done on the iOS Provisioning Portal, so let's get that over with as quickly as possible.

Provisioning Profiles and Certificates, Oh My!

> *APNS needs a certificate!*

To enable push notifications in your app, it needs to be signed with a provisioning profile that is configured for push. In addition, your server needs to sign its communications to APNS with an SSL certificate.

The provisioning profile and SSL certificate are closely tied together and are only valid for a single App ID. This is a protection that ensures only your server can send push notifications to instances of your app, and no one else.

As you know, apps use different provisioning profiles for development and distribution. There are also two types of push server certificates:

- **Development**. If your app is running in Debug mode and is signed with the Development provisioning profile (Code Signing Identity is “iPhone Developer”), then your server must be using the Development certificate.
- **Production**. Apps that are distributed as Ad Hoc or on the App Store (when Code Signing Identify is “iPhone Distribution”) must talk to a server that uses the Production certificate. If there is a mismatch between these, push

notifications cannot be delivered to your app.

In this tutorial, you won’t bother with the distribution profiles and certificates and just use the ones for development.

Generating the Certificate Signing Request (CSR)

Remember how you had to go to the iOS Provisioning Portal and make a Development Certificate after you signed up for the iOS Developer Program? If so, then these next steps should be familiar. Still, I advise you to follow them exactly. Most of the problems people have with getting push notifications to work are due to problems with the certificates.

Digital certificates are based on public-private key cryptography. You don’t need to know anything about cryptography to use certificates, but you do need to be aware that a certificate always works in combination with a private key.

The certificate is the public part of this key pair. It is safe to give it to others, which is exactly what happens when you communicate over SSL. The private key, however, should be kept… private. It’s a secret. Your private key is nobody’s business but your own. It’s important to know that you can’t use the certificate if you don’t have the private key.

Whenever you apply for a digital certificate, you need to provide a Certificate Signing Request, or CSR for short. When you create the CSR, a new private key is made that is put into your keychain. You then send the CSR to a certificate authority (in this case that is the iOS Developer Portal), which will generate the SSL certificate for you based on the information in the CSR.

Open Keychain Access on your Mac (it is in Applications/Utilities) and choose the menu option **Request a Certificate from a Certificate Authority…**.

If you do not have this menu option or it says “Request a Certificate from a Certificate Authority with key”, then download and install the WWDR Intermediate Certificatefirst. Also make sure no private key is selected in the main Keychain Access window.

You should now see the following window:

Enter your email address here. I’ve heard people recommended you use the same email address that you used to sign up for the iOS Developer Program, but it seems to accept any email address just fine.

Enter “PushChat” for Common Name. You can type anything you want here, but choose something descriptive. This allows us to easily find the private key later.

Check **Saved to disk** and click **Continue**. Save the file as "PushChat.certSigningRequest".

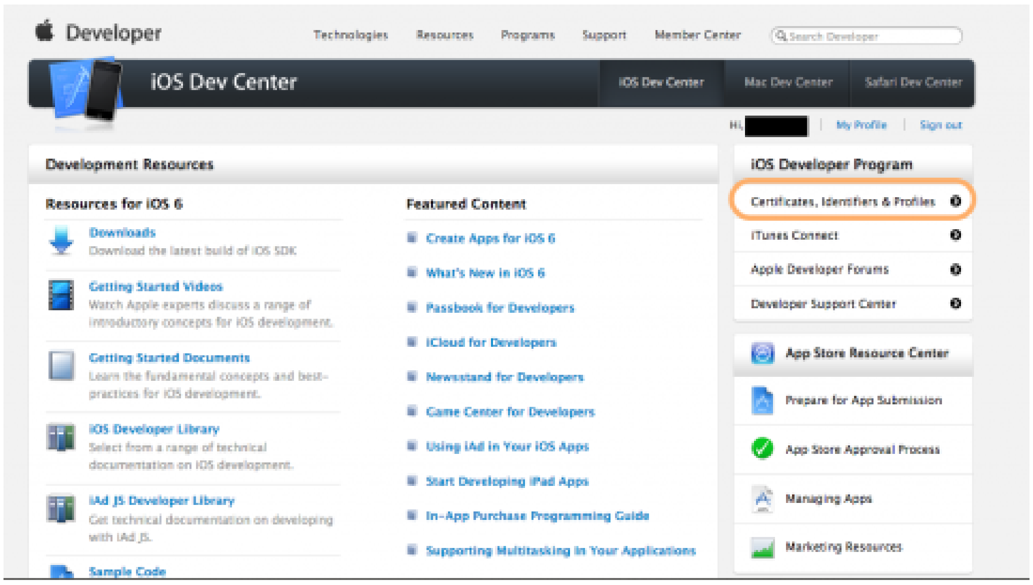
If you go to the Keys section of Keychain Access, you will see that a new private key has appeared in your keychain. Right click it and choose Export.

Save the private key as **PushChatKey.p12** and enter a passphrase.

For the convenience of this tutorial, I used the passphrase "pushchat" to protect the p12 file but you should really choose something that is less easy to guess. The private key needs to be a secret, remember? Do choose a passphrase that you can recall, or you won't be able to use the private key later.

Making the App ID and SSL Certificate

Log in to the iOS Dev Center and "Select the Certificates, Identifiers and Profiles" from the right panel.


Developer   Technologies   Resources   Programs   Support   Member Center   Search Developer

iOS Dev Center   iOS Dev Center   Mac Dev Center   Safari Dev Center

Hi, [redacted] | My Profile | Sign out

Development Resources

Resources for iOS 6

Downloads
Download the latest build of iOS SDK

Getting Started Videos
Watch Apple experts discuss a range of introductory concepts for iOS development.

Getting Started Documents
Learn the fundamental concepts and best-practices for iOS development.

iOS Developer Library
Select from a range of technical documentation on iOS development.

iAd JS Developer Library
Get technical documentation on developing with iAd JS.

Sample Code

Featured Content

- Create Apps for iOS 6
- What's New in iOS 6
- Passbook for Developers
- iCloud for Developers
- Newsstand for Developers
- Game Center for Developers
- Using iAd in Your iOS Apps
- Start Developing iPad Apps
- In-App Purchase Programming Guide
- Supporting Multitasking in Your Applications

iOS Developer Program

- Certificates, Identifiers & Profiles
- iTunes Connect
- Apple Developer Forums
- Developer Support Center

App Store Resource Center

- Prepare for App Submission
- App Store Approval Process
- Managing Apps
- Marketing Resources

You will be presented with the following screen (Doesn't the new dev center UI look sleek :))

Since you're making an iOS app select **Certificates** in the**iOS Apps** section.

Now, you are going to make a new App ID. Each push app needs its own unique ID because push notifications are sent to a specific application. (You cannot use a wildcard ID.)

Go to **App IDs** in the sidebar and click the **+** button.

Fill the following details:

- **App ID Description**: PushChat
- **App Services** Check the Push Notifications Checkbox
- **Explicit App ID**: com.hollance.PushChat

It is probably best if you choose your own Bundle Identifier here – com.yoursite.PushChat – instead of using mine. You will need to set this same bundle ID in your Xcode project. After you're done filling all the details press the **Continue**button. You will be asked to verify the details of the app id, if everything seems okay click **Submit**

Hurray! You have successfully registered a new App ID.

In a few moments, you will generate the SSL certificate that your push server uses to make a secure connection to APNS. This certificate is linked with your App ID. Your server can only send push notifications to that particular app, not to any other apps.

After you have made the App ID, it shows up like this in the list:

Select the **PushChat** app ID from the list. This will open up an accordion as shown below:

Notice in the "Push Notification" row, there are two orange lights that say "Configurable" in the Development and Distribution column. This means your App ID can be used with push, but you still need to set this up. Click on the**Setting** button to configure these settings.

Scroll down to the Push Notifications section and select the**Create Certificate** button in the **Development SSL Certificate** section.

The "Add iOS Certificate" wizard comes up:

The first thing it asks you is to generate a Certificate Signing Request. You already did that, so click **Continue**. In the next step you upload the CSR. Choose the CSR file that you generated earlier and click **Generate**.

It takes a few seconds to generate the SSL certificate. Click**Continue** when it's done.

Now click **Download** to get the certificate – it is named "aps_development.cer".

As you can see, you have a valid certificate and push is now available for development. You can download the certificate again here if necessary. The development certificate is only valid for 3 months.

When you are ready to release your app, repeat this process for the production certificate. The steps are the same.

Note: The production certificate remains valid for a year, but you want to renew it before the year is over to ensure there is no downtime for your app.

You don't have to add the certificate to your Keychain, although you could if you wanted to by double-clicking the downloaded **aps_development.cer** file. If you do, you'll see that it is now associated with the private key.

Making a PEM File

So now you have three files:

- The CSR
- The private key as a p12 file (PushChatKey.p12)
- The SSL certificate, aps_development.cer

Store these three files in a safe place. You could throw away the CSR but in my opinion it is easier to keep it. When your certificate expires, you can use the same CSR to generate a new one. If you were to generate a new CSR, you would also get a new private key. By re-using the CSR you can keep using your existing private key and only the .cer file will change.

You have to convert the certificate and private key into a format that is more usable. Because the push part of our server will be written in PHP, you will combine the certificate and the private key into a single file that uses the PEM format.

The specifics of what PEM is doesn't really matter (in fact, I have no idea) but it makes it easier for PHP to use the certificate. If you write your push server in another language, these following steps may not apply to you.

You're going to use the command-line OpenSSL tools for this. Open a Terminal and execute the following steps.

Go to the folder where you downloaded the files, in my case the Desktop:

```
$ cd ~/Desktop/
```

Convert the .cer file into a .pem file:

```
$ openssl x509 -in aps_development.cer -inform der -out
PushChatCert.pem
```

Convert the private key’s .p12 file into a .pem file:

```
$ openssl pkcs12 -nocerts -out PushChatKey.pem -in
PushChatKey.p12
Enter Import Password:
MAC verified OK
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
```

You first need to enter the passphrase for the .p12 file so that openssl can read it. Then you need to enter a new passphrase that will be used to encrypt the PEM file. Again for this tutorial I used “pushchat” as the PEM passphrase. You should choose something more secure.

Note: if you don’t enter a PEM passphrase, openssl will not give an error message but the generated .pem file will not have the private key in it.

Finally, combine the certificate and key into a single .pem file:

```
$ cat PushChatCert.pem PushChatKey.pem > ck.pem
```

At this point it’s a good idea to test whether the certificate works. Execute the following command:

```
$ telnet gateway.sandbox.push.apple.com 2195
Trying 17.172.232.226...
Connected to gateway.sandbox.push-apple.com.akadns.net.
Escape character is '^]'.
```

This tries to make a regular, unencrypted, connection to the APNS server. If you see the above response, then your Mac can reach APNS. Press Ctrl+C to close the connection. If you get an error message, then make sure your firewall allows outgoing connections on port 2195.

Let’s try connecting again, this time using our SSL certificate and private key to set up a secure connection:

```
$ openssl s_client -connect
gateway.sandbox.push.apple.com:2195
    -cert PushChatCert.pem -key PushChatKey.pem
Enter pass phrase for PushChatKey.pem:
```

You should see a whole bunch of output, which is openssl letting you know what is going on under the hood.

If the connection is successful, you should be able to type a few characters. When you press enter, the server should disconnect. If there was a problem establishing the connection, openssl will give you an error message but you may have to scroll up through the output to find it.

Note: There are two different APNS servers: the “sandbox” server that you can use for testing, and the live server that you use in production mode. Above, we used the sandbox server because our certificate is intended for development, not production use.

Making the Provisioning Profile

You’re not yet done with the iOS Dev Center. Click the**Provisioning Profiles** button in the sidebar and click the **+**button.

This will open up the iOS provisioning profile wizard.

Step 1: Select Type

Select the “iOS App development” option button in the first step of the wizard and press **Continue**.

Step 2: Configure

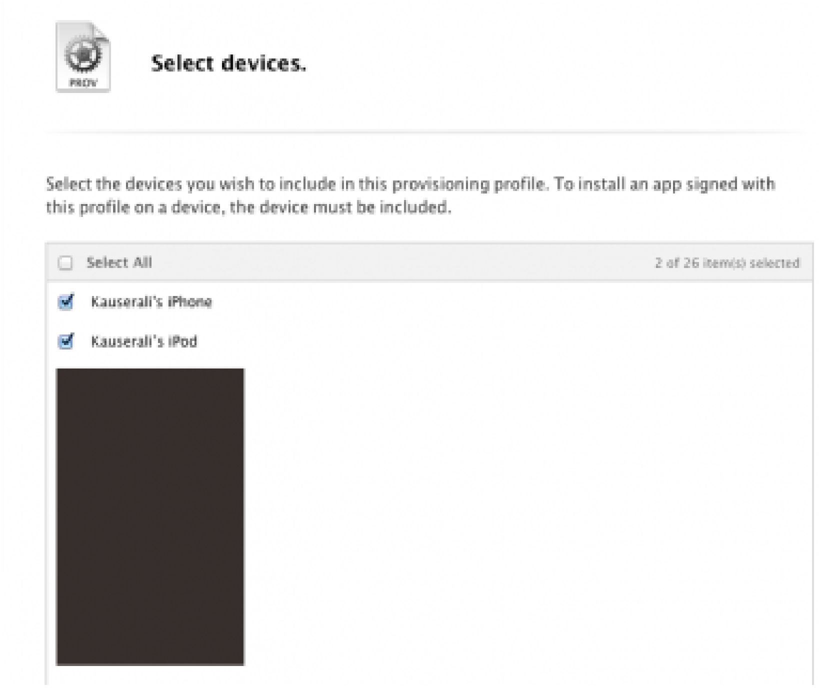
Select the PushChat app id that you created in the previous section. This will ensure that this provisioning profile is explicitly tied to the PushChat app.

Step 3: Generate

In this step you select the certificates you want to include in this provisioning profile. This step should be quite routine by now.

Step 4: Select devices

Select the devices you want to include in this provisioning profile. Since you’re creating the development profile you would typically select the devices you use for development here.



Select devices.

Select the devices you wish to include in this provisioning profile. To install an app signed with this profile on a device, the device must be included.

☐ Select All — 2 of 26 item(s) selected

✓ Kauserali's iPhone

✓ Kauserali's iPod

[redacted]

Step 5: Name this profile

Set the provisioning profile name as "PushChat Development" as shown below.

You're almost done! Finally press the **Download** button, this will download the newly created Development provisioning profile.

Add the provisioning profile to Xcode by double-clicking it or dragging it onto the Xcode icon.

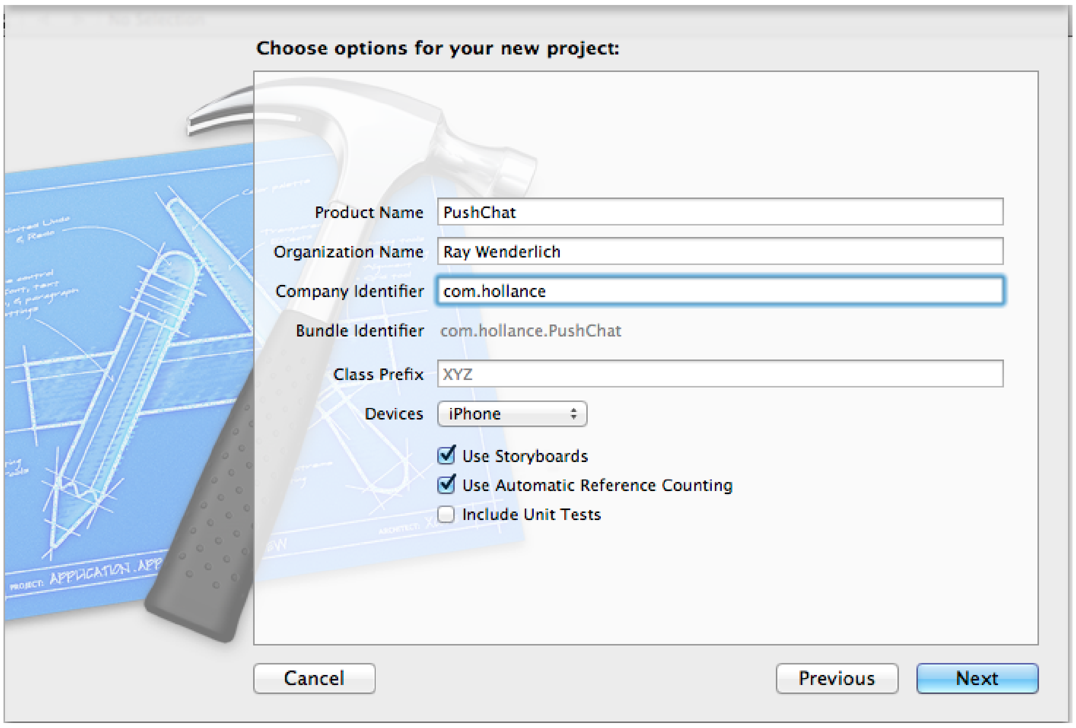
If you're ready to release your app to the public, you will have to repeat this process to make an Ad Hoc or App Store distribution profile.

A Very Basic App

So far things haven't been really exciting, but those preliminaries are necessary. I wanted to show you in detail how to generate the certificate because it's not something you do every day and push won't work without it.

You've already established that your certificate is valid by connecting to the sandbox server. Let's test if you can actually send some push notifications!

Fire up Xcode and choose **File, New Project**. In the assistant, pick **Single View Application** and continue to the next step.


Choose options for your new project:

Product Name: PushChat
Organization Name: Ray Wenderlich
Company Identifier: com.hollance
Bundle Identifier: com.hollance.PushChat
Class Prefix: XYZ
Devices: iPhone
[✓] Use Storyboards
[✓] Use Automatic Reference Counting
[ ] Include Unit Tests

I filled in these fields as follows:

- Product Name: PushChat
- Organization Name: Ray Wenderlich
- Company Identifier: com.hollance
- Device Family: iPhone

The Product Name and Company Identifier together form the Bundle ID. In my case that is “com.hollance.PushChat”. You should choose a Product Name and Company Identifier that correspond to the App ID that you made earlier in the Provisioning Portal (com.yourname.PushChat). Make sure that the “Use Storyboards” and “Use Automatic reference counting” checkboxes are checked.

Finish the assistant and open **AppDelegate.m**. Change the application:didFinishLaunchingWithOptions: method to look like this:

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
        // Let the device know we want to receive push
notifications
        [[UIApplication sharedApplication]
registerForRemoteNotificationTypes:
                (UIRemoteNotificationTypeBadge |
UIRemoteNotificationTypeSound |
UIRemoteNotificationTypeAlert)];

    return YES;
}
```

The new call to registerForRemoteNotificationTypes: tells the OS that this app wants to receive push notifications.

Build & Run the app. You need to do this on your device because the simulator does not support push notifications. Xcode should automatically have selected the new provisioning profile. If you get a code sign error, then make sure the proper profile is selected in the Code Signing build settings.

When the app starts and registers for push notifications, it shows a message to inform the user that this app wishes to send push notifications.

The app asks this only once. If the user picks “OK”, then we should be all set. However, if they choose “Don’t Allow” then our app will never receive push notifications. The user can reverse their decision in the phone’s Settings.

The name of your app will be added to the phone’s Settings, under Notifications. The user can enable or disable the notifications for your app here, including individual settings for badges, sounds, and alerts.

Your app can find out which types of push notifications are enabled through:

```
UIRemoteNotificationType enabledTypes = [[UIApplication
sharedApplication] enabledRemoteNotificationTypes];
```

There is one more thing you need to add in order to be able to receive push notifications. Add the following to**AppDelegate.m**:

```
- (void)application:(UIApplication*)application
didRegisterForRemoteNotificationsWithDeviceToken:(NSData*)dev
{
        NSLog(@"My token is: %@", deviceToken);
}

- (void)application:(UIApplication*)application
didFailToRegisterForRemoteNotificationsWithError:(NSError*)er
{
        NSLog(@"Failed to get token, error: %@", error);
}
```

When your app registers for remote (push) notifications, it tries to obtain a “device token”. This is a 32-byte number that uniquely identifies your device. Think of the device token as the address that a push notification will be delivered to.

Run the app again and you should see something like this in Xcode’s console window:

```
My token is:
<740f4707 bebcf74f 9b7c25d4 8e335894 5f6aa01d a5ddb387
462c7eaf 61bb78ad>
```

The token is an opaque binary data structure, stuffed into an NSData object. Apple doesn't want you to mess around with its internals. For our purposes it is enough to know that it is currently 32 bytes long. As you can see above, the token can be represented by 64 hexadecimal characters. You will be using it in that format, although you still strip off the brackets and leave out the spaces.

If you run the app in the simulator, the application:didFailToRegisterForRemoteNotificationsWithError: method will be called as push notifications are not supported in the simulator.

That's it for the app. There is one more thing to do and then you can finally see some push notifications in action!

Sending Your First Push Notification

As I've mentioned a few times before, you need to set up a server that sends the push notifications to your app. For this first test, you're not going to set up a server just yet. Instead, I'll give you a very simple PHP script that sets up a connection to APNS and sends a push notification to a device token that you specify. You can run this straight from your Mac.

Download the SimplePush code and unzip it. You need to make some changes to **simplepush.php**.

```
// Put your device token here (without spaces):
$deviceToken =
'0f744707bebcf74f9b7c25d48e3358945f6aa01da5ddb387462c7eaf61bb

// Put your private key's passphrase here:
$passphrase = 'pushchat';

// Put your alert message here:
$message = 'My first push notification!';
```

You should copy the device token from the app into the $deviceToken variable. Be sure to leave out the spaces and brackets; it should just be 64 hexadecimal characters. Put your private key's passphrase into $passphrase, and the text you wish to send in $message.

Copy your **ck.pem** file into the SimplePush folder. Remember, the ck.pem file contains both your certificate and the private key.

Then open a Terminal and type:

```
$ php simplepush.php
```

If all goes well, the script should say:

```
Connected to APNS
Message successfully delivered
```

And within a few seconds you should receive your first push notification:

Note that you won't see anything when the app is open. The push message is delivered but you did not build anything in the app to handle it yet. Close the app and try again.

If the simplepush.php script exits with an error message, then check that you have made the PEM file correctly, and that you can connect to the sandbox server without problems (see above).

For now, it is not important what the script exactly does. I will explain more about this in the second part of the series when we build a true push server.

Troubleshooting

This section has a few tips to keep track of if you're facing problems to get push notifications to work.

Some notifications received but not all: If you're sending multiple push notifications simultaneously and you receive only a few, fear not that is intended behaviour. APNS maintains a QoS (Quality of Service) queue for each device with a push app. The size of this queue is 1 so if you send multiple notifications then the last notification is overridden.

Problem connecting to Push Notification Service:

1. One possibility here could be that there is a firewall blocking the ports used by APNs. Make sure youunblock these ports.
2. Another possibility here could be that you've gotten the private key and CSR file wrong. While updating this tutorial I myself made this mistake and fixed it using the correct private key. Remember that each app id has a unique CSR and private key combination.
3. Unable to get local issuer certificate. This error means that the certificate from the server could not be verified. To fix this you need to download the Entrust CA root certificate. This can be done from the Terminal using the command:

   ```
   curl -O
   https://www.entrust.net/downloads/binary/entrust_2048_ca.c
   er
   ```

You then also need to add

```
stream_context_set_option($ctx, 'ssl', 'cafile',
'entrust_2048_ca.cer');
```

to the simplepush.php file after line 16.

Where To Go From Here?

At this point, you've successfully set up an app to receive push notifications, and delivered your first push notification through custom PHP code!

Next check out part 2 of the tutorial series, where I'll cover how to make a simple direct messaging app called PushChat that uses push notifications to deliver the messages.

There you'll learn how to make a complete server side API that delivers push notifications continuously in the background!

In the meantime, if you have any questions, comments, or suggestions, feel free to join in the forum discussion below!