This tutorial is the second part of our tutorials covering setting up and running the Push Notifications Native Extension for Adobe AIR from distriqt.

The Push Notifications extension enables the push notification functionality in your mobile AIR application allowing you to send (or push) notifications and data to users of your application. The extensions API will have you up and running with remote notifications in minutes.

Importantly this extension allows you to write code once and have access to push notifications on multiple platforms allowing you to quickly integrate push notifications no matter what platform you are developing for!

- Apple Push Notification Service (APNS)
- Google Cloud Messaging (GCM)

The extension provides a completely native implementation of push notifications allowing you to receive notifications from the associated services.

In this first tutorial we will run through the concepts involved in getting your application setup and running with push notifications including:

- Setup and registering certificates for APNS
- Setup for GCM

Please note that this is not a server implementation. You’ll need to investigate appropriate server code for your application and to manage your user tokens and to dispatch notifications appropriately. We will however show you how to send a single “test” notification using several methods in Part 2.
Setup

Firstly we’re going to cover how to correct setup your application in the application stores. This involves many steps and is the most complex part of the whole integration.

There are a lot of steps to go through to get your application setup in the application stores correctly. It is very important that you follow the steps below precisely and don’t skip any steps as not generating the certificates in the correct order can leave you frustrated with an application that doesn’t work.

We’ll address each of the systems separately and if you only want one of the notification systems i.e. GCM and not APNS it is safe to ignore the section you don’t need.
Apple Push Notification Service (APNS) is the push notification system for use on iOS with iPhones, iPads and other such iOS devices.

The system is quite complex and has a lot of different components as you can see in the diagram below.

When the device finally receives the notification, the notification will be displayed to the user (depending on the options specified in the notification, more about these later). The user can then respond to the notification as they see fit, dismissing it or opening your application.

Your server will be required to implement a solution to gather device tokens and to send the notification to the APNS server with the required devices specified. This solution will not be part of this tutorial but we will show...
you everything up to and including sending a test notification.

Requirements

Before we start, it’s important to check that you have all of the following:

- An iOS device (they do not work in the simulator)
- An iOS Developer program membership
- A server connected to the internet, ideally that can run background processes (for testing this can simply be a local installation of Apache + PHP)
- Access to an OSX machine.

It’s important that you have all of these organised before starting the APNS setup. It is potentially possible to do parts of this without an OSX machine but we won’t be showing that in this tutorial.

The APNS Data Packet

It’s important to understand that at the very base of the APNS implementation is a very simple JSON packet. They are intended to be small, delivering only the simplest amount of information and can be no more than 256 bytes.

```
{
    "aps":
    {
        "alert": "A notification message",
        "sound": "default"
    }
}
```

The notification packet can contain a range of information that will determine what happens on the phone when the packet is received.

Details on the notification payload are best described in the “Local and Push Notification Programming Guide” as part of the iOS Developer Library Documentation. You can find this here:


Caveats and Characteristics

Firstly and foremost, push notifications are not reliable and the APNS server does not guarantee delivery of a notification that your server sends it. The delivery time will also vary from a few seconds up to half an hour.
Additionally a device may not be able to receive a notification, for instance it may be on a network that blocks the ports used for push notification communication. Or it could be as simple as the device is switched off.

Push notifications will become expensive if you don’t prepare for it. You must take care that your application when deployed to lots of users doesn’t quickly use your notification quotas. Be aware of what notifications are being sent to your applications and be prepared to pay if you don’t.

You must take these into consideration when designing your notifications system.

Lastly, you must abide by the Apple App Store Review Guidelines for Push Notifications. They are all fairly reasonable but it’s always important to check before you design your application requirements.


Certificates

This is the painful part of the APNS setup, gathering all the required provisioning profiles and certificates. Get a cup of coffee and take your time through the process and you won’t end up pulling your hair out.

There are two different provisioning profiles associated with an application, the development and the distribution (or production) profiles. Similarly there are two types of certificates for your application server to use to communicate with the APNS server:

- Development - for use when you’re running your application in debug mode and signed with the developer profile. With this certificate you can talk to the APNS sandbox environment to test your application.
- Production - for use when you are distributing your application either as an ad-hoc application or through the AppStore.

If you use the wrong certificate then the push notification will fail and your application won’t receive the notification. The process for creating these certificates is very similar for both cases so we’ll just concentrate on the development profiles and certificates.

Certificate Signing Request (CSR)

This process is very similar to when you first created your developer account in the iOS developer program. These certificates are based off the classic form of public-private key cryptography, but don’t worry you don’t need to know anything about them to make this work. Just be aware that you will be creating certificates and exchanging them between the apple servers and yourself.

Most importantly you’ll create a certificate that is to be used to communicate from your server to the APNS servers. This is different and separate from your developer certificates you use to sign your applications. It is however “private“ and you should not share the certificate with anyone not related to the development of these push notifications.
To create these certificates you create a “Certificate Signing Request” (CSR) and send the CSR to a certificate issuing authority, in this case the APNS section of the iOS developer portal. The authority then issues you with a certificate that you can use to secure your communications.

On OSX you’ll create your CSR by using the **Keychain Access** application. You’ll find this in the following location:

Applications / Utilities / Keychain Access

Choose the option under the main application menu (as shown in the screenshot below):

![Keychain Access screenshot](image-url)

You should then be presented with a window asking for some details about the certificate request.

Here you should enter your email address, it’s recommended you use the same email as you use to sign up for the developer program but this is not essential, and a common name for your certificate. The common name just needs to be something descriptive to allow you to easily identify the certificate.

Make sure you’ve checked the **Saved to disk** option and then click Continue. Save the file somewhere convenient, you’ll need it again soon. We suggest you change the name of the file to something like `ApplicationPushNotifications.certSigningRequest`. 
Next we need to export the private key as a p12 file. To do this return to the Keychain Access program and open the keys category. You should be able to see a public and a private key with the Common name you used in the previous step. You can also use the search in the top left to locate them.

We export this certificate by right clicking on the **private** key and selecting export.

You'll need to enter a passphrase, which will be used to protect your private key. You should enter something that is secure but also that you can easily recall. Normal rules with passwords apply here!
Now we move on to the provisioning in the iOS portal.
iOS Provisioning Portal

Now you’ll need to log into your iOS Provisioning Portal with your iOS developer Apple ID.

https://daw.apple.com/cgi-bin/WebObjects/DSAuthWeb.woa/wa/login?appIdKey=d4f7d769c2abecc664d0dadfed6a67f943442b5e9c87524d4587a95773750cea&path=%2F%2Fios%2Fmanage%2Foverview%2Findex.action

Here we are going to make a new application ID and use the CSR to create an SSL certificate for your application server communications.

You can update an existing application ID to include push notifications, as long as you follow the steps after creating the application ID to enable push notifications in your application and make sure you download the provisioning profile for the application again! It will have changed and won’t work if you don’t update your profile.

To make a new application ID, go to the **App IDs** section on the left and click the **New App ID** button.
You will then be presented with the “Create App ID” form. Enter your application details, for example:

Description: Distriqt Test Application
Bundle Identifier: com.distriqt.test

Take note of the **Bundle Seed ID (App ID Prefix)** and the **Bundle Identifier (App ID Suffix)**. We’ll need these later.

After you have created the application it will appear in your App ID list, with something like the following beside it. Note that Push Notifications are listed as “Configurable”, both for development and production.

Click **Configure**. You should then be presented with the **Configure App ID** page. Look for the section as shown below and click the checkbox to enable the App ID for Apple Push Notification service. This should activate the Configure actions on the right. Click the button beside the Development Push SSL Certificate. (You can come back to this same place to configure the production certificate at a later date).
This will now ask you to upload the CSR file we created earlier. This is the ApplicationPushNotifications.certSigningRequest file not the p12 file. Choose the file and click **Generate**.

Your certificate should now get generated and after a while you’ll get a message saying “Your APNs SSL Certificate has been generated. Please continue to the next step.”.
You’ll now get to download the SSL certificate. Click download and save the certificate somewhere safe. It should be called `aps_development.cer`. We suggest you rename it to something matching the application that it’s associated with.
You should now see the Status for the Development Push SSL Certificate is green and labelled as Enabled.

![Enable for Apple Push Notification service](image)

You should see the expiration date of the certificate listed. It is very important that you note this date and make sure you return and update the certificate before the date is reached. Otherwise you'll experience a period where notifications will not work in your application!

Lastly while we’re still in the Provisioning Portal lets create a provisioning profile for the application. Go to the Provisioning section and on the Development tab, click **New Profile**.

![Provisioning Portal: District Pty Ltd](image)

Give the profile a name and select the App ID you’ve just created, along with any devices you wish to test on. Make sure you select your developer certificate from the Certificates section. Click Submit and the profile will
be generated.

This is no different from any other provisioning profile, all we are doing is making sure we retrieve the profile after the application has been setup for push notifications.

You may need to refresh the page if the profile still is in the Pending state.

Click the Download action and save the provisioning profile file along with all your other development files. You won’t need to use this file on your application server, you simply have to use this to develop your application.

**Using the Certificate**

You should have 3 files saved somewhere at this point (ignoring the mobile provisioning file):

- the Certificate Signing Request (CSR)
- the private key - exported as a p12 file
- the SSL certificate downloaded from the iOS provisioning portal (aps_development.cer)

These files are very important in the communication between your application server and the APNS servers. It’s suggested you keep them in a safe place. You can throw the CSR away if you wish but we’ve always kept them. If you need to renew the certificates at a later date you can use the same CSR and skip a few of the earlier steps.

What we need to do now is convert the certificate and private key into a format that we can use easily.

Our test notification example for APNS uses PHP so we’ll convert them into a single file in the PEM format. We’ll use this PEM file later when we are sending test notifications but here we will show you how to create it.

We’ll be using the OpenSSL tools for this. Place the p12 and cer files into a directory and open a Terminal window in that directory. We’re going to create to pem files from each of the p12 and cer files and then combine them into a single file.

Convert the cer file to pem:

```
openssl x509 -in aps_developer_identity.cer -inform der -out YOUR_CERTIFICATE.pem
```

Convert the p12 file into a pem file. It will ask you for the password to the private key. This is the password you used when you exported the key from your Keychain. It should then ask you for a new “PEM pass phrase”, this is a separate password to the PEM file you’re creating so again use something secure. You must enter a passphrase. If you don’t your private key may not get included correctly in the final PEM.

```
openssl pkcs12 -nocerts -out YOUR_PRIVATE_KEY.pem -in YOUR_PRIVATE_KEY.p12
```

Finally, combine the two files:

```
cat YOUR_CERTIFICATE.pem YOUR_PRIVATE_KEY.pem > ck.pem
```
This ck.pem file is what we’ll use in the php script later so keep this file handy.

You can test the connection to the APNS sandbox (sandbox as we created the development certificate through this process) at this point if you wish with the following command:

```bash
openssl s_client -connect gateway.sandbox.push.apple.com:2195 -cert YOUR_CERTIFICATE.pem -key YOUR_PRIVATE_KEY.pem
```

This should connect to the server and dump out a whole heap of information about the process. Eventually after the successful connection you should be able to type a few characters and the server should disconnect.
Google Cloud Messaging (GCM) is Android’s push notification system. The GCM system is a very simple system to get up and running.

Google Cloud Messaging for Android (GCM) is a free service that helps developers send data from servers to their Android applications on Android devices. This could be a lightweight message telling the Android application that there is new data to be fetched from the server (for instance, a movie uploaded by a friend), or it could be a message containing up to 4kb of payload data (so apps like instant messaging can consume the message directly). The GCM service handles all aspects of queueing of messages and delivery to the target Android application running on the target device.

1. Your application registers for push notifications
2. Our extension uses the native implementation to register with the GCM servers
3. The server returns the device registration ID to the application which you’ll receive a “registered” event and be able to retrieve the registration ID.
4. You send the registration ID to your server application and store for later use
5. When you have some information to send to your server, your server sends a notification packet to the GCM Server
6. The server delivers the notification to your application and our extension dispatches a notification event.

Architecturally it’s very similar to the APNS process, though “device token” is called “registration ID” the process is relatively identical. The actual implementation is very different though as we’ll see in the following.
Caveats and Characteristics

Here are the primary characteristics of Google Cloud Messaging (GCM):

- It allows 3rd-party application servers to send messages to their Android applications.
- An Android application on an Android device doesn't need to be running to receive messages. The system will wake up the Android application via Intent broadcast when the message arrives, as long as the application is set up with the proper broadcast receiver and permissions.
- It does not provide any built-in user interface or other handling for message data. GCM simply passes raw message data received straight to the Android application, which has full control of how to handle it. For example, the application might post a notification, display a custom user interface, or silently sync data.
- It requires devices running Android 2.2 or higher that also have the Google Play Store application installed, or an emulator running Android 2.2 with Google APIs. However, you are not limited to deploying your Android applications through Google Play Store.
- It uses an existing connection for Google services. For pre-3.0 devices, this requires users to set up their Google account on their mobile devices. A Google account is not a requirement on devices running Android 4.0.4 or higher.

It’s important to note a few things here. Firstly the device needs to have the Google Play Store application installed and secondly that there needs to be an active Google account on the device (pre Android v4.0.4). You must be aware of these limitations when designing and testing your application.

Set up the GCM Service

You'll need to sign up to the Google Cloud Messaging service. This process is outlined in the "Getting Started" section on the Android developer site:


You need to follow these instructions to get your API keys. If you’re using our native extension in AIR then anything further can be ignored (i.e. anything from "Install the Helper Libraries"), though you will need it when developing your own native application.

We’ve summarised the instructions below.

Creating a Google API Project

To create a Google API project:

- Open the Google APIs Console Page
- If you haven't created an API project yet, this page will prompt you to do so:
Note: If you already have existing projects, the first page you see will be the Dashboard page. From there you can create a new project by opening the project drop-down menu (upper left corner) and choosing **Other projects > Create**.

- Click **Create project**. Your browser URL will change to something like:

https://code.google.com/apis/console/#project:4815162342

- Take note of the value after `#project:` (4815162342 in this example). This is your project number, and it will be used later on as the **GCM sender ID**.

**Enabling the GCM Service**

To enable the GCM service:

- In the main Google APIs Console page, select **Services**.
- Turn the **Google Cloud Messaging** toggle to ON.
- In the Terms of Service page, accept the terms.
Obtaining an API Key

To obtain an API key:

- In the main Google APIs Console page, select **API Access**. You will see a screen that resembles the following:

![API Access screen](image)

- Click **Create new Server key**. Either a server key or a browser key should work. The advantage to using a server key is that it allows you to whitelist IP addresses. The following screen appears:

![Configure Server Key screen](image)

- Click **Create**: 
Take note of the API key value (YourKeyWillBeShownHere) in this example, as it will be used later on.

**Note:** If you need to rotate the key, click **Generate new key**. A new key will be created while the old one will still be active for up to 24 hours. If you want to get rid of the old key immediately (for example, if you feel it was compromised), click **Delete key**.

You should now have two important pieces of information:

- Your **GCM Sender ID**
- Your **API key**

You’ll need these later so make note of them.

If you are building your own application server you should read the following documentation:


**Credits**

We’d like to acknowledge the following people who’ve helped immensely with creating this extension:

- Matthijs Hollemans https://twitter.com/mhollemans for his extensive knowledge on the APNS certificates generation process