// tutorial

Using the Push Notifications Extension
Part 2: Using the extension

Version 1.0

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

- Including the extension in your application
- Registering for notifications
- Listening for notifications
- Sending a test notification

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.

Application Descriptor

To enable push notifications in your application you will need to add some additional information to your application descriptor.

iOS

Under iOS the information goes into an iPhone node in the application descriptor. It should look something like the following:

```
    <iPhone>
        <InfoAdditions><![CDATA[
            <key>UIDeviceFamily</key>
            <array>
            <string>1</string>
            </array>
        ]]></InfoAdditions>
        <requestedDisplayResolution>high</requestedDisplayResolution>
        <Entitlements><![CDATA[
            <key>get-task-allow</key>
            <true/>
            <key>aps-environment</key>
            <string>development</string>
            <key>application-identifier</key>
            <string>X5LW23Q6CJ.com.distriqt.test</string>
            <key>keychain-access-groups</key>
            <array>
                <string>X5LW23Q6CJ.*</string>
            </array>
        ]]></Entitlements>
    </iPhone>
```

The important parts to note in these additions are the areas highlighted in red. These all relate to your application’s entry in the iOS Provisioning Portal.

The first field is the **aps-environment**. This field indicates whether we are using the development or the production environment. It must be either **development** or **production** and depends on which configuration you are using. If you are running a debug build you should use development. If you are looking to publish the application to the AppStore, you should use production.

If you took note of the applications Bundle Seed ID (App ID Prefix) and Bundle Identifier (App ID Suffix) previously then you can use them here, otherwise see the next section on how to “Acquire the App ID Prefix and Suffix” and return here with that information.

An App ID is the combination of a unique ten character string called the "Bundle Seed ID" and a traditional CF Bundle ID (or Bundle Identifier). The Bundle Seed ID portion of your App ID can be utilized to share keychain access between multiple applications you build with a single App ID. In addition, it can be incorporated into any external hardware accessories you wish to pair your iOS application with. Registration of your App ID is required to utilize the Push Notifications (APNs) and to register an application to incorporate In-App Purchases.

The Bundle Seed ID or App ID Prefix is the first part of the full application ID, **application-identifier**, that you must supply in the application descriptor. It should be of the form of 10 alpha-numeric characters, in our example case its X5LW23Q6CJ. You should enter this value into the **keychain-access-groups** field highlighted in red.

The second part of the **application-identifier** is the Bundle Identifier (App ID Suffix) and is what we normally consider the AIR application ID.

You should be able to construct the full application ID now and insert it into the **application-identifier** field. This is as follows:

[BUNDLE SEED ID].[BUNDLE IDENTIFIER]

Eg:

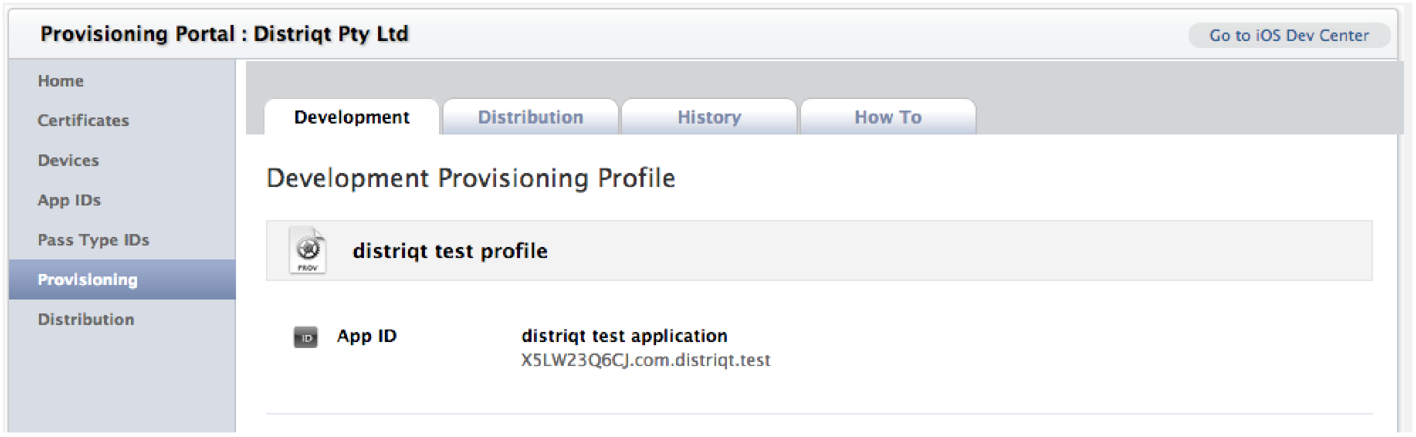
X5LW23Q6CJ.com.distriqt.test

Acquire the App ID Prefix and Suffix

To acquire the Bundle Seed ID and the Bundle Identifiers you need to log into the iOS Provisioning Portal.

Select "Provisioning" from the menu and locate the provisioning profile for your application and click on it. You should be displayed with the following screen, showing the App ID which is of the format:

[BUNDLE SEED ID].[BUNDLE IDENTIFIER]



Development Provisioning Profile

distriqt test profile

App ID distriqt test application
X5LW23Q6CJ.com.distriqt.test

Android

The application descriptor additions under Android are located in a node called “android”. It should look something like the following:

```
<android>
    <manifestAdditions><![CDATA[
        <manifest android:installLocation="auto">
            <uses-sdk android:minSdkVersion="8" android:targetSdkVersion="16"/>
            <uses-permission android:name="android.permission.INTERNET"/>
            <uses-permission android:name="android.permission.GET_ACCOUNTS" />
            <uses-permission android:name="android.permission.VIBRATE"/>
            <uses-permission android:name="android.permission.WAKE_LOCK" />

            <uses-permission android:name="com.google.android.c2dm.permission.RECEIVE" />
            <permission android:name="air.com.distriqt.test.permission.C2D_MESSAGE"
android:protectionLevel="signature" />
            <uses-permission android:name="air.com.distriqt.test.permission.C2D_MESSAGE" />

            <application>
                <receiver android:enabled="true" android:exported="true"
android:name="com.distriqt.extension.pushnotifications.PushNotificationsBroadcastReceiver"
android:permission="com.google.android.c2dm.permission.SEND" >
                    <intent-filter>
                        <action android:name="com.google.android.c2dm.intent.RECEIVE" />
                        <action android:name="com.google.android.c2dm.intent.REGISTRATION" />
                        <category android:name="air.com.distriqt.test" />
                    </intent-filter>
                </receiver>
                <service android:enabled="true" android:exported="true"
android:name="com.distriqt.extension.pushnotifications.gcm.GCMIntentService" />
            </application>
        </manifest>
    ]]></manifestAdditions>
</android>
```

There are two important things here. If you are adding this to an existing application, you must make sure you have only one **application** node inside your manifest additions. You’ll need to copy the receiver and service from that node and place that inside your current application node. If you have two application nodes the push notifications may not work.

Secondly you need to replace all the application ID’s (highlighted in red) with your application ID. It is important here that you use the full application ID with the **air.** prefix as this will be your actual ID in the Android system.

Code Practice

The extension is implemented as a singleton, basically meaning the function is always accessed through a static variable defined on the extension class.

We suggest that you use a try/catch around most of the calls to the code while developing as we occasionally throw errors that will help you debug a problem.

Initialise the extension

Finally, we're ready to use the Push Notification extension in your code.

The first call must be to the initialisation function of the extension which will setup the extension for use. You should only call this once in your application lifetime and before you call any other function of the extension. You'll need to provide your developer key you received when you signed up to the extension.

```
PushNotifications.init( DEV_KEY );
```

At this stage you can also use the `isSupported` flag to determine whether the device will support push notifications.

There are many reasons this flag may return false on a device, including:

- Incorrect setup of the application descriptor additions, most commonly this is caused by incorrectly modifying the application ids (Bundle IDs etc)
- Incorrect usage of APNS certificates
- No Google account on an older Android device
- Google Play isn't installed on an older Android device
- The user has denied the application access to push notifications

You should always check this flag and respond accordingly, either informing your users or gracefully falling back to another notification method.

Registering to notification service

The next step is the register to the push notification service. The details of what happens here is dependant on the platform but the result is to receive a token or id of the device.

```
PushNotifications.service.register( GCM_SENDER_ID );
```

This is an asynchronous call and the extension will respond with an event at a later time indicating the result of the registration. Under Android you must pass this registration function your GCM Sender ID that you created as part of the signup process for GCM (see the previous setup section). Under iOS this is ignored so you can safely pass your GCM sender ID on both platforms.

You should listen for the events as you would a normal event listener:

```
PushNotifications.service.addEventListener( EVENT_TYPE, HANDLER );
```

In particular the following events are related to the register call.

- `PushNotificationEvent.REGISTERING`
    - This event is dispatched when the service is attempting to register
- `PushNotificationEvent.REGISTER_SUCCESS`
    - Dispatched on successful registration with the server.
    - The call to `getDeviceToken` will now return a valid registration id, either the "device token" on iOS or the "registration id" on Android.
- `PushNotificationEvent.ERROR`
    - An error has occurred
    - The data field will contain more information about the error that occurred and you should refer to the documentation on the `PushNotificationEvent` class for more details.

These three events, in particular the later two, are the important events to listen for in the registration process. Once you've received the `REGISTER_SUCCESS` event, you are ready to receive notifications, however if you've received the `ERROR` event, you should generally wait for a small period and retry the registration (depending on the error data if the error was critical or not).

Importantly on the REGISTER_SUCCESS handler you need to send the device's ID/Token to your server and add it to the list of devices that wish to receive notifications. For the moment we can just trace out the information.

```
PushNotifications.service.addEventListener( PushNotificationEvent.REGISTER_SUCCESS,
registerSuccessHandler );
PushNotifications.service.register( GCM_SENDER_ID );

private function registerSuccessHandler( event:PushNotificationEvent ):void
{
	trace( "PN registration succeeded with reg ID: " + event.data );
	// Send this id to your server
}
```

Receiving notifications

In order for your application to receive a notification you must listen for the `PushNotificationEvent.NOTIFICATION`. This event will be dispatched whenever a push notification is received by the extension. It's as simple as listening for the event and processing the data.

```
PushNotifications.service.addEventListener( PushNotificationEvent.NOTIFICATION, notificationHandler
);

private function notificationHandler( event:PushNotificationEvent ):void
{
	trace("Push notification received!" );
	trace( event.data );

	try
	{
		JSON.parse( event.data );
		// Process your data
	}
	catch (e:Error)
	{
		trace( "ERROR::"+e.message );
	}
}
```

The event data will contain the information that you sent from your server formatted as a JSON string.

Sending a Test Notification

We've supplied several scripts with the extension example that allow you to send a notification to a specific device id. Once you have setup your application and successfully registered for notifications you can copy the device ID into these scripts and send a test notification. The following examples use `php` to send the notification so you need to have this installed on your machine before they will work.

As each platform is completely different in it's approach we've supplied separate scripts for the platforms.

iOS

The following is the test script from the example application supplied with the ANE called apns-push.php. It is a php script which requires you to enter a few details before it will work. You need to have the certificate file (ck.pem) file you created as part of the APNS setup in the same directory as the script (or change the path in the line `$certfile = 'ck.pem';`

Then you'll need to update the pass phrase on the certificate file and copy your device token from your successfully registered device.

Lastly, if you wish, you can modify the message that will be sent to your device.

You can now send a push notification to your device using the following command from a terminal window:

```
php apns-push.php
```

```
/**
 * This is a test script to send a message to an APNS device, i.e. an Apple iOS device using the APNS.
 *
 * To use this script you need to set the following 4 peieces of information
 *  - message
 *  - device token
 *  - certificate file
 *  - pass phrase
 *
 * Then run it using php:
 * <code> php apns-push.php </code>
 *
 */

// Message to send
$message = 'the test message';

// Put your device token here (without spaces):
$deviceToken = 'DEVICE_TOKEN_ID';


//
//     Certificate Key Details
$certfile = 'ck.pem';
```

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

////////////////////////////////////////////////////////////////////////////////

$ctx = stream_context_create();
stream_context_set_option($ctx, 'ssl', 'local_cert', $certfile);
stream_context_set_option($ctx, 'ssl', 'passphrase', $passphrase);

// Open a connection to the APNS server
$fp = stream_socket_client(
       'ssl://gateway.sandbox.push.apple.com:2195', $err,
       $errstr, 60, STREAM_CLIENT_CONNECT|STREAM_CLIENT_PERSISTENT, $ctx);

if (!$fp)
       exit("Failed to connect: $err $errstr" . PHP_EOL);

echo 'Connected to APNS' . PHP_EOL;

// Create the payload body
$body['aps'] = array(
       'alert' => $message,
       'badge' => 0,
       'sound' => 'default'
       );

// Encode the payload as JSON
$payload = json_encode($body);

// Build the binary notification
$msg = chr(0) . pack('n', 32) . pack('H*', $deviceToken) . pack('n', strlen($payload)) . $payload;

// Send it to the server
$result = fwrite($fp, $msg, strlen($msg));

if (!$result)
       echo 'Message not delivered' . PHP_EOL;
else
       echo 'Message successfully delivered' . PHP_EOL;

// Close the connection to the server
fclose($fp);
```

Note: This example uses the sandbox url for the APNS service so you should use your development provisioning profile and APNS certificates here. You can change this to the production server if you wish.

Android

Once you've got your api key and your device id you're ready to send a message. The PHP function below implements a simple "send notification" function using a JSON formatted message package.

```php
/**
 * The following function will send a GCM notification using curl.
 *
 * @param $apiKey       [string]              The Browser API key string for your GCM account
 * @param $registrationIdsArray [array]       An array of registration ids to send this notification to
 * @param $messageData [array] A named array of data to send as the notification payload
 */
function sendNotification( $apiKey, $registrationIdsArray, $messageData )
{
   $headers = array("Content-Type:" . "application/json", "Authorization:" . "key=" . $apiKey);
   $data = array(
       'data' => $messageData,
       'registration_ids' => $registrationIdsArray
   );

   $ch = curl_init();

   curl_setopt( $ch, CURLOPT_HTTPHEADER, $headers );
   curl_setopt( $ch, CURLOPT_URL, "https://android.googleapis.com/gcm/send" );
   curl_setopt( $ch, CURLOPT_SSL_VERIFYHOST, 0 );
   curl_setopt( $ch, CURLOPT_SSL_VERIFYPEER, 0 );
   curl_setopt( $ch, CURLOPT_RETURNTRANSFER, true );
   curl_setopt( $ch, CURLOPT_POSTFIELDS, json_encode($data) );

   $response = curl_exec($ch);
   curl_close($ch);

   return $response;
}
```

To use this function you'll need to assemble the message data and an array of registration ids, as demonstrated below. The message data we construct here is an example of usage with our extension to automate the display of a notification when the GCM arrives on the device.

```
// Message to send
$message      = "the test message";
$tickerText   = "ticker text message";
$contentTitle = "content title";
$contentText  = "content body";

$registrationId = 'DEVICE_ID';
$apiKey = "YOUR_BROWSER_API_KEY";

$response = sendNotification(
               $apiKey,
               array($registrationId),
               array('message' => $message, 'tickerText' => $tickerText, 'contentTitle' =>
$contentTitle, "contentText" => $contentText) );

echo $response;
```

The echoed response will show the JSON GCM message package sent to the server, and as long as you have the correct device ID and a registered device, your notification will appear on your device. Happy messaging!

Further examples can be found on our site:

* How to use C# to send a GCM message: http://labs.distriqt.com/post/1223
* How to use PHP to send a GCM message: http://labs.distriqt.com/post/1273

Understanding Error Events

SERVICE_NOT_AVAILABLE

This is normal, it does happen occasionally under Android.

From the GCM documentation: http://developer.android.com/guide/google/gcm/gcm.html

"The device can't read the response, or there was a 500/503 from the server that can be retried later. The Android application should use exponential back-off and retry."

Our extension implements the exponential backoff retry, but you can implement your own if you wish, ie retry the registration at your own intervals.

FAQ

We use Adobe flex 4.6 with AIR 3.1 / 3.3 / 3.4 SDK, will the extensions support that?

Yes, the extensions support version 3.1+. In some cases you may require a version of the iOS SDK as well.

We are planning to integrate the notification through Urban Airship, will the extension still be useful?

The extension implements the client side registration of the notification service with the native service server (APNS or GCM). You can implement whichever server you require. We have not tested with Urban Airship but there is no reason it shouldn't work.

The extension handles receipt of notifications from these servers and notifies your application.

It will not register with your server, only with the GCM/APNS server to receive a registration id. You're required to do handle the registration ids in your own code.

What platforms does the extension support?

The extension uses APNS on iOS and GCM on Android.

Will the extension help to open the mobile app in a specific place once the notification received?

This is unavailable at the moment. Due to the way native extensions are implemented, we don't have access to the application startup options so we can't receive application startup notifications. You'll need to check your server for notifications at startup.

What Android versions are supported?

Our extension should work on any version of Android that supports AIR and GCM.

What is the iOS versions supported?

iOS 5.0+

iOS: Do we need a mac with xcode installation environment to generate the flex IPA with your extension?

No, but you need access to an iOS SDK v5.0+.

How do you play a custom sound when a notification arrives?

Using a custom sound on APNS is quite simple.

Firstly you need to create your sound and export as an iOS supported format such as caf.

Then add the sound to your AIR application and make sure it's packaged with the application in:

Project / Properties / Actionscript Build Packaging / Apple iOS / Package Contents

- make sure the sound is checked to be included.

The use the name of the sound file in the sound notification payload:

```
{
    aps =     {
        alert = "the test message";
        sound = "example.caf";
    };
}
```

The sound should play when your notification arrives.

Android: How do I get a notification to appear in the notification area automatically?

Under Android there is no "default" way of displaying the notifications. We have implemented a simple message that will get displayed in the notification area when you supply certain variables in the message content. Basically you need to supply two variables "tickerText" and "contentTitle" with an optional "contentText".

tickerText - displayed in the notification bar contentTitle & contentText - displayed in the pull down notification area

You can see the example on our site (in PHP):

http://labs.distriqt.com/post/1273

So you'll need to send something like the following:

```
"data": {
    "tickerText" => "some ticker text",
    "contentTitle" => "some content title",
    "contentText" => "some content description text",
    "otherCustomData" => ...
}, ...
```

Android: The isSupported flag is returning false?

The isSupported flag will return false if you haven't correctly setup the permissions in the manifest file.

Check the example application descriptor file for the correct usage. You just have to change all the YOUR_APP_ID references to your application id. In the example application the application id is **air.com.distriqt.test**. You must include the air prefix for AIR applications.

```
<android>
      <manifestAdditions><![CDATA[
            <manifest android:installLocation="auto">
                  <uses-sdk android:minSdkVersion="8" android:targetSdkVersion="16"/>
                  <uses-permission android:name="android.permission.INTERNET"/>
                  <uses-permission android:name="android.permission.GET_ACCOUNTS" />
                  <uses-permission android:name="android.permission.WAKE_LOCK" />
                  <uses-permission android:name="com.google.android.c2dm.permission.RECEIVE" />

                  <permission android:name="YOUR_APP_ID.permission.C2D_MESSAGE"
android:protectionLevel="signature" />
                  <uses-permission android:name="YOUR_APP_ID.permission.C2D_MESSAGE" />

                  <application>
                        <receiver android:enabled="true" android:exported="true"
android:name="com.distriqt.extension.pushnotifications.PushNotificationsBroadcastReceiver"
android:permission="com.google.android.c2dm.permission.SEND" >
                              <intent-filter>
                                    <action android:name="com.google.android.c2dm.intent.RECEIVE" />
                                    <action
android:name="com.google.android.c2dm.intent.REGISTRATION" />
                                    <category android:name="YOUR_APP_ID" />
                              </intent-filter>
                        </receiver>
                        <service android:enabled="true" android:exported="true"
android:name="com.distriqt.extension.pushnotifications.gcm.GCMIntentService" />
                  </application>
            </manifest>
```

```
        ]]></manifestAdditions>
</android>
```

The next step you'll need to do is to setup your application listing to be ready for in-app purchases. The process for this is different in each of the stores. Below we've outlined the process for each of the currently supported stores.

Further Information

For more information:

- Refer to the documentation: http://extensions.distriqt.com/docs/pushnotifications/docs/
- Refer to tutorials and tips on: http://labs.distriqt.com/tag/native-extension
- And lastly if you still need further help, you can contact us at support@distriqt.com

GCM:

- http://developer.android.com/google/gcm/gs.html

APNS:

- http://developer.apple.com/library/ios/#documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Introduction/Introduction.html#//apple_ref/doc/uid/TP40008194-CH1-SW1

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