

Easy Cross-platform Notifications system



User Guide v1.04
By Carlos Fernandez Musoles

Support: carlos.fernandez.musoles@gmail.com

Introduction

Setting your own remote notifications system up in any of the two mobile platforms (iOS or Android) can be a daunting task, and not because it's per se a difficult thing to do, but because if you don't get all the details right, it simply won't work.

With Easy Cross Platform Notifications (ECPN) I intend to explain, step by step, what needs to be done on your project in order to get a truly platform-agnostic notification service that will send messages from any device running iOS to any device running Android, all sitting comfortably within a Unity application.

But ECPN is not only a guide or a tutorial; it comes with all the server-side and client-side code, together with an example Unity Scene to show its capabilities.

Unfortunately, due to the nature of the remote notifications –mostly because all of the certificates and permissions that need to be set beforehand- it is not possible to give an off-the-shelf solution that will allow your current project to drop a prefab and have the system do the rest. But if you bear with me till the end of the guide, I promise you, you will not only understand how this all work, but you will be able to make improvements and additions to the simple mechanics!

Disclaimers about notifications (iOS and android)

- 1) In iOS/android, it may take a while before the devices receive the registration IDs from APSN/Google servers.
- 2) Once registered, sending messages should be quick; however, receiving them may take several minutes –as per documentation, it is not guaranteed that neither Google nor APSN (Apple) servers will ever send the message, so you should design your project accordingly.

What are Remote Notifications?

If you are reading this, you probably know the answer to this. Remote Notifications are, simply put, messages that are sent to users of your application from either yourself (i.e. server-generated message to all your users) or from other users (i.e. user-generated message at login). They have the virtue of reaching the destination device whether the application is running at the moment or not.

Their nature varies depending on the platform, but the way they are sent is the same: your app requests a message to be sent, which goes to your server. Your server builds the notification (message + recipients) and requests the APNS (Apple's notification server) or Google Server to deliver it to a list of registered users. Previously, your users must agree to register their device with your app. Don't worry if you didn't get all of it, we will see it in detail further down.

Where do we start?

ECPN is a cross-platform system, but it can be used in iOS or android exclusively. If that is your case, I suggest you skip the section that does not suit you, as it won't affect the rest of the tutorial. Of course, if you need cross-platform notifications you will need to read it all! But hey, it'll be worth it.

Server set up: PHP + MySQL / MS SQL

As mentioned in the Asset Store, you need to have a remote server that is able to run PHP files and with MySQL (or MS SQL) database. This server will act as an intermediary, requesting the APNS and the Google servers to send notifications to your users.

It is not the purpose of this guide to help you pick a server or to set it up, but you should be able to find more than enough information over the Internet. Fortunately all we need is a remote directory where to place our PHP files and a MySQL table that we will create.

Important note! From version 1.02 ECPN includes support for MS SQL as well (thanks to *hrlarsen*). Just use the php files included in the **PHP Files (MS SQL)** folder instead of the normal ones and you will be ready to go!

1) Create a table in MySQL (or MS SQL)

We will need a table in our database to store the user's device tokens, which is what APNS / Google servers use to identify the recipients of the notifications. This should be easy, just import the following table:

MySQL:

```
CREATE TABLE IF NOT EXISTS `ECPN_table` (  
  `uid` int(10) NOT NULL AUTO_INCREMENT,  
  `unityID` varchar(100) COLLATE latin1_spanish_ci NOT NULL,  
  `deviceID` text COLLATE latin1_spanish_ci NOT NULL,  
  `os` varchar(10) COLLATE latin1_spanish_ci NOT NULL,  
  PRIMARY KEY (`uid`)  
) ENGINE=MyISAM DEFAULT CHARSET=latin1  
COLLATE=latin1_spanish_ci AUTO_INCREMENT=1;
```

MS SQL:

```
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id  
= OBJECT_ID(N'[dbo].[ECPN_table]'))  
  
AND type in (N'U'))
```

```
CREATE TABLE [dbo].[ECPN_table] (
    uid int IDENTITY(1,1) PRIMARY KEY CLUSTERED,
    unityID varchar(100) COLLATE Modern_Spanish_CI_AS NOT
    NULL,
    deviceID varchar(255) COLLATE Modern_Spanish_CI_AS NOT
    NULL,
    os varchar(10) COLLATE Modern_Spanish_CI_AS NOT NULL,
)
```

2) Upload all PHP files to your server

The files provided in this package under the PHP folder need to be uploaded to your server. However, before you can do that you need to tweak them a bit.

From version 1.02 ECPN offers users the possibility of having MySQL or MS SQL databases. Depending on which technology you prefer, use the set of php files in one folder:

- PHP Files (MySQL): use these ones if you want to use MySQL database
- PHP Files (MS SQL): Use these if you prefer MS SQL.

Once you have chosen a set of files, go to the correspondent folder and open the settings.php file and enter the following information:

- \$loginURL
- \$username
- \$password
- \$database
- GOOGLE_API_KEY (we will discuss this one in the Android section)

It is all explained in the same php file and you should know things like the username/password of the database, or its URL. If you are not the administrator of your database, please contact him/her and request this information.

After modifying this file, place all the php files all in the same server folder and keep a note on the URL of that folder (for instance, www.yourwebsite.com/myPHPfiles). You will need to enter this information in your Unity project later on.

iOS push notifications – steps

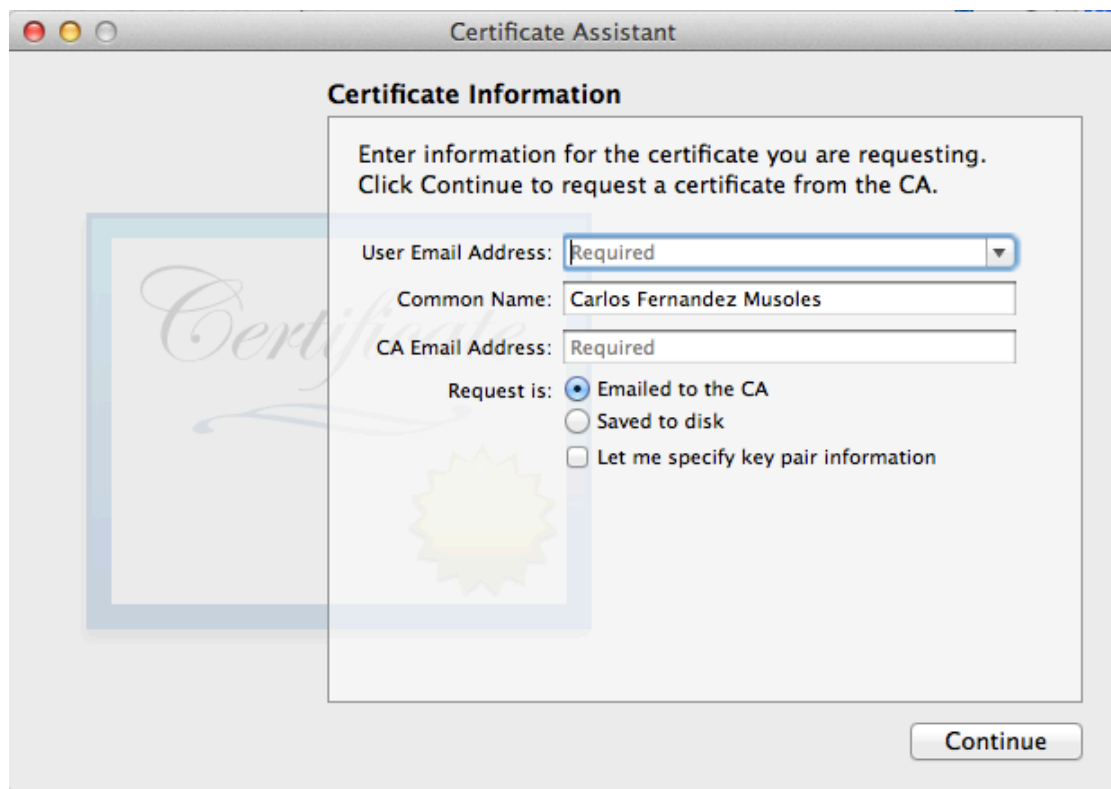
What you need:

- Unity iPhone / iPad / iPod Touch license
- Server with MySQL and PHP
- Mac with any version of Xcode
- iOS Developer Account
- An iOS device –push notifications don't work on the desktop

The most difficult thing about push notifications (that's how remote notifications are called in the iOS world) is by far setting up the certificates needed. My advice is to take it one step at a time, follow the steps lined up here and you won't be lost in a sea of exceptions and errors later on! If this is not your first iOS app surely some of this will sound very familiar; but if you are –like I was not long ago- a novice in the matter, fear not, just pay close attention and you will be OK.

1) Generate Certificate Signing Request (CSR)

On your Mac, open Keychain Access (you will find it under Applications > Utilities) and click on the option Request a Certificate from a Certificate Authority under Keychain Access > Certificate Assistant menu. You should see a window similar to this one:



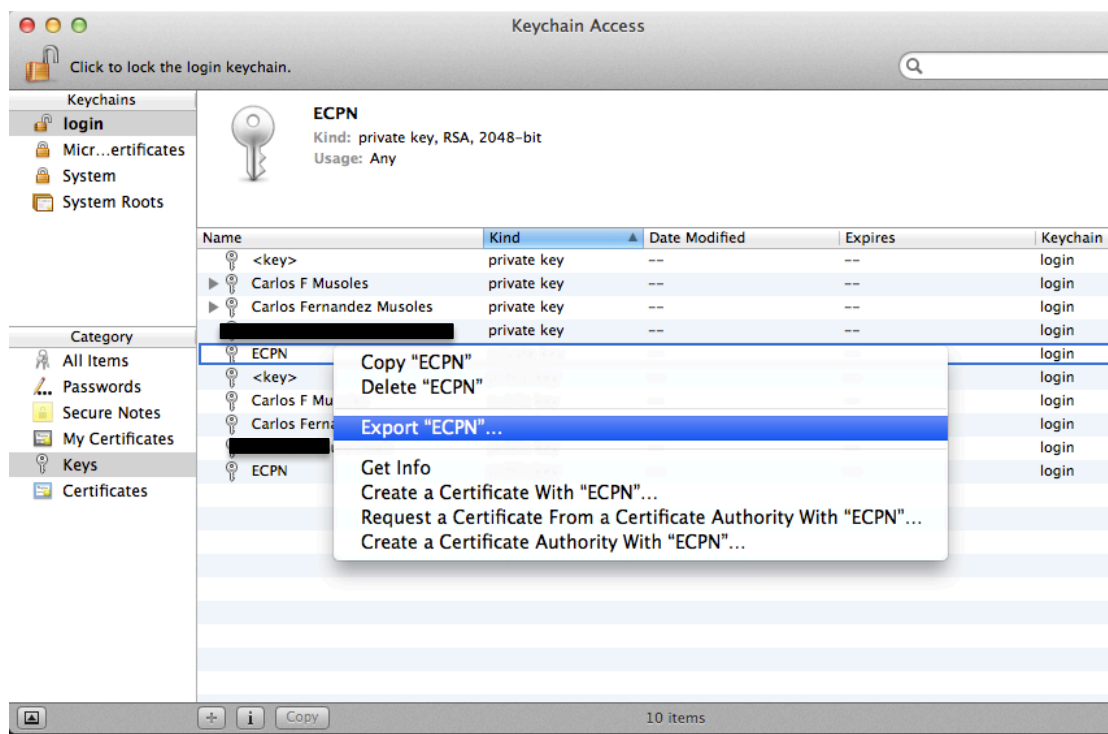
Enter the following:

- *User Email Address*: your email address (preferably the one linked to your iOS Developer account, although this doesn't seem to be a requirement).

- *Common name:* Enter ECPN –it can be anything, it will help you to find the private key later.
- *Request is:* Select Saved to disk option.

Once all the options are selected, click on Continue and save the file in a safe place as “ECPN.certSigningRequest”.

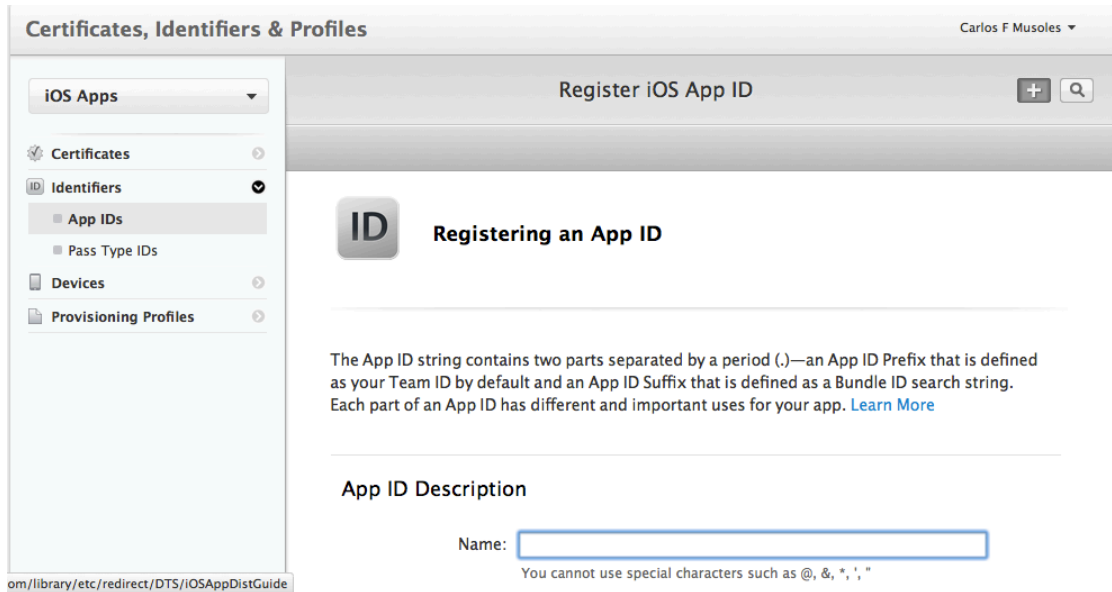
Now go to the Keys section of Keychain Access and find the private key just created (it will appear with the name you entered as common name above). Right-click on it and select Export. Save it in the same folder as the certificate as “ECPNkey.p12”. When you export it it will ask you for a password. Enter whatever you want, but you need to be able to remember it later on!



2) Create App ID

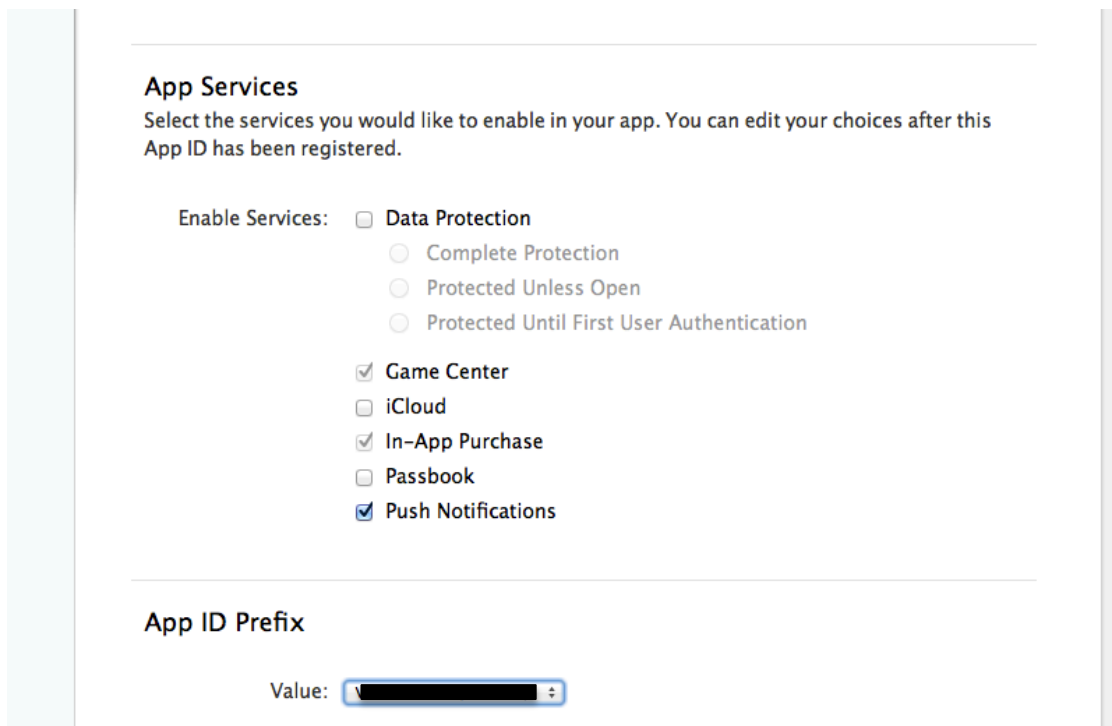
If you are adding Push Notification to your already created project, you won't need this part. However, I leave it for convenience and for those of you starting from scratch.

To create an App ID, log in to the iOS Dev Center and within the Identifiers section select App ID. Click on the “+” button to add a new app, and you should see something like this:



Enter the following information:

- *App ID Description:* ECPN
- *App Services:* Check Push Notifications
- *Explicit App ID:* com.CFM.ECPN –it is **probably best** if you choose here your own bundle name; this will need to coincide with the bundle name given to Unity when building your application. Something like “com.YOURNAME.ECPN”



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:

We recommend using a reverse-domain name style string (i.e., com.domainname.appname). It cannot contain an asterisk (*).

☐ Wildcard App ID

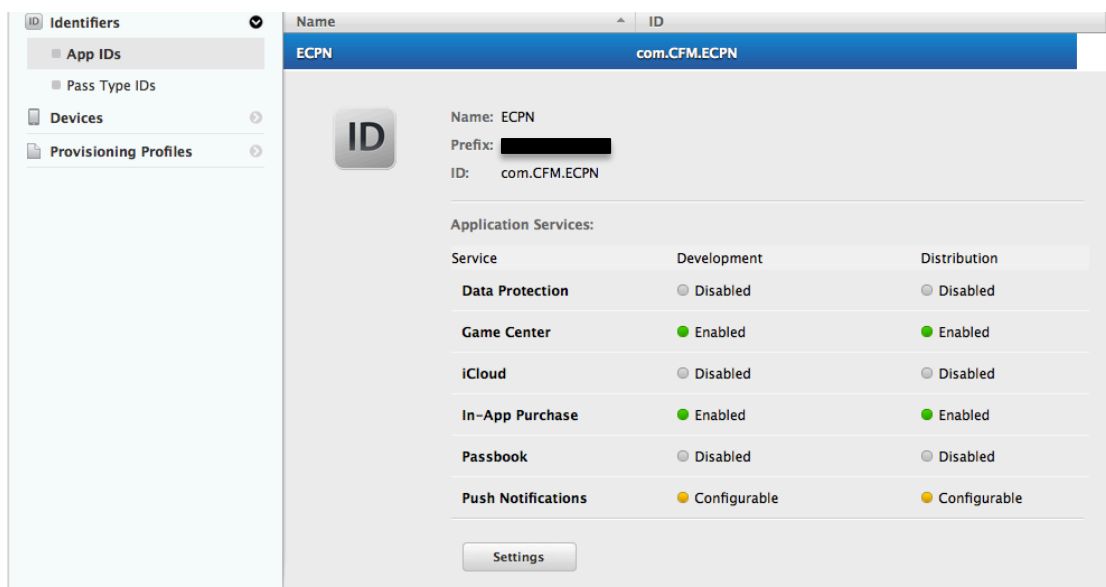
This allows you to use a single App ID to match multiple apps. To create a wildcard App ID, enter an asterisk (*) as the last digit in the Bundle ID field.

Bundle ID:


Example: com.domainname.*


3) Activate Push Notification service

The last certificate you need is the Push Notification certificate. Select ECPN from the list of your App IDs, you should see a screen like this:




Here click Settings and scroll down to Push Notifications and click on Create Certificate under the Development SSL Certificate section of the Push Notifications. This will last for 3 months and you can renew it, but of course if you are publishing your app you should repeat this steps but create a production SSL certificate.

 **Passbook**
● Disabled


 **Push Notifications**
● Enabled

Apple Push Notification service SSL Certificates
To configure push notifications for this iOS App ID, a Client SSL Certificate that allows your notification server to connect to the Apple Push Notification Service is required. Each iOS App ID requires its own Client SSL Certificate. Manage and generate your certificates below.

 **Development SSL Certificate**

Create certificate to use for this App ID.


Create Certificate...

 **Production SSL Certificate**


Create certificate to use for this App ID.

Create Certificate...

To create the SSL cert you will need the original CSR that we created on step 1 –the wizard will ask you for it. Once you have it created, download the SSL by clicking on Download and save it in the folder where you are placing all the files as “aps_development.cer”.

 **Push Notifications**
● Enabled

Apple Push Notification service SSL Certificates
To configure push notifications for this iOS App ID, a Client SSL Certificate that allows your notification server to connect to the Apple Push Notification Service is required. Each iOS App ID requires its own Client SSL Certificate. Manage and generate your certificates below.

 **Development SSL Certificate**

Name: Apple Development iOS Push Services: com.CFM.E...

Type: APNs Development iOS


Expires: May 23, 2014

Revoke

Download

Create an additional certificate to use for this App ID.

Create Certificate...

 **Production SSL Certificate**

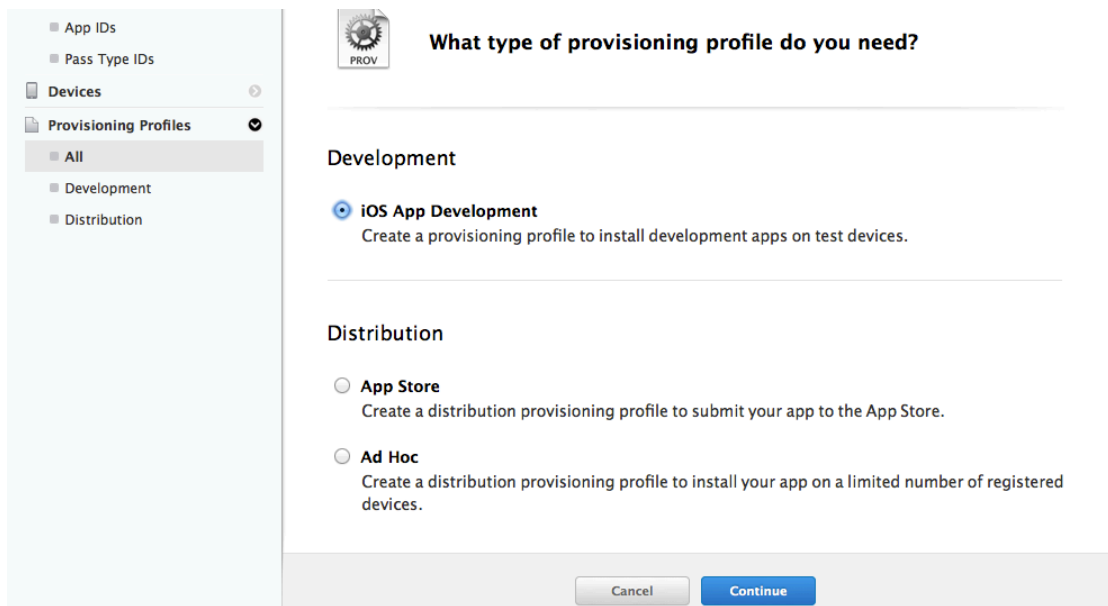
Create certificate to use for this App ID.

Create Certificate...

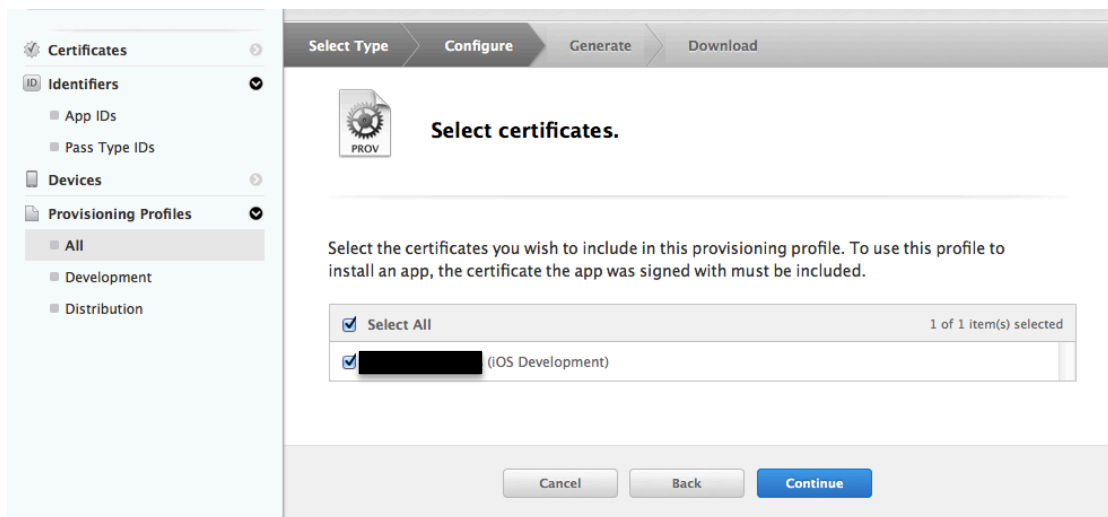
Download.action?displayId=T57XFC9RF7&type=BKLRAVXMGM

4) Making the provisioning file

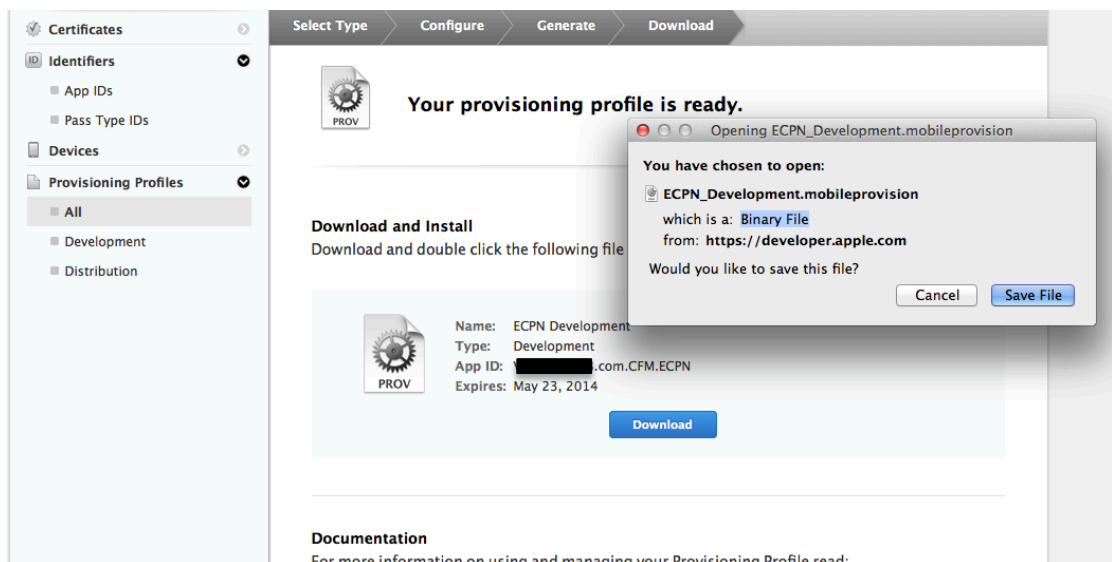
This is to validate your device and for it to be able to use the Push Notification service. On the iOS Portal click on Provisioning Profiles and create a new one. Select to create an iOS App Development profile.



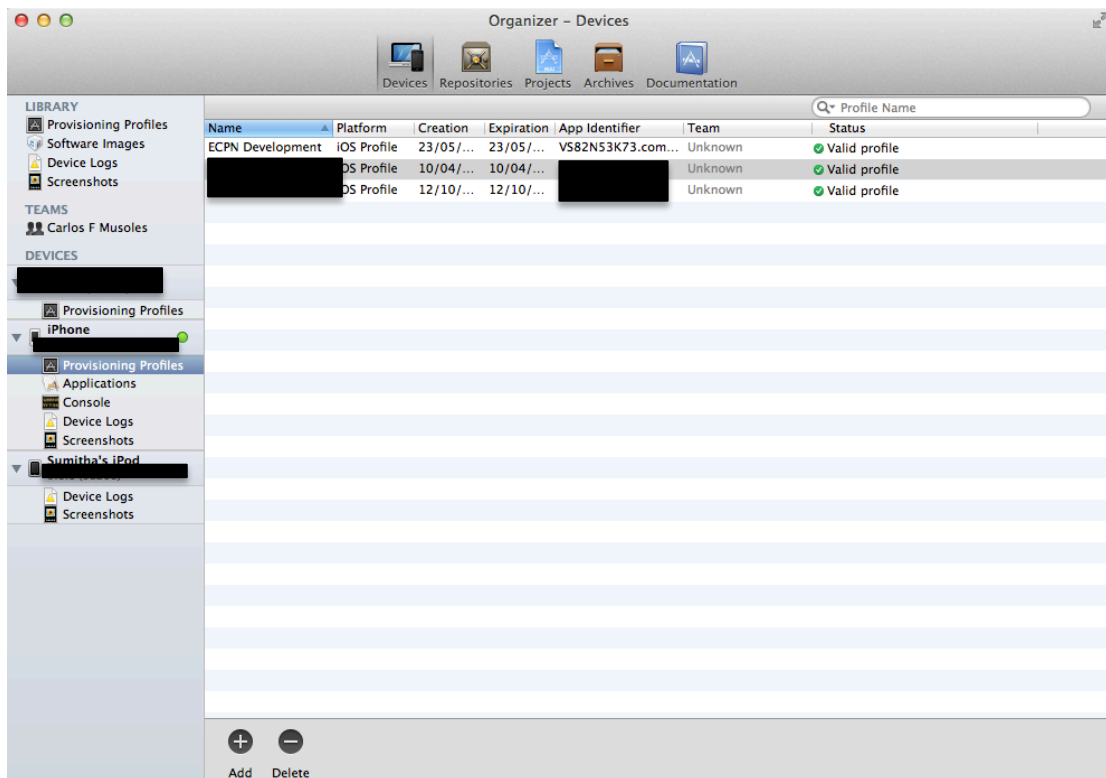
Configure it for your App ID and your personal iOS Development certificate – it should appear all on the wizard).



Select also your device here –if this is the first time you are using your device, you will need to add it; you can do it on the Devices section under the iOS Portal.



Once it is created, download it to a local folder as “aps_development.cer”. You must then add it to Xcode by double clicking on the cert file. To avoid any issues I like to add the certificate to my iOS device manually; you can do this easily! Just plug your device in and go to the Organiser within Xcode. Go to the Provisioning Profiles section and drag and drop the certificate to your device.



5) Making a PEM file with the three files (CSR, p12 and SSL)

We are almost there! All that is left is to join all the files in a PEM file that will be used by your server to certify each message it sends.

In a console window go to the directory where all the files we created are located. In order, enter the following commands:

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

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

(this will ask for an import password, which is the one you entered when exporting the private key in step 1; it also asks for a PEM pass phrase and a repeat. Please set a memorable one as we will need it when setting up our server).

```
cat ECPNcert.pem ECPNkey.pem > ck.pem
```

To test that everything has gone well, test the following command

```
openssl s_client -connect gateway.sandbox.push.apple.com:2195 -cert ECPNcert.pem -key ECPNkey.pem
```

(it will ask for the pass phrase you set just now)

If you see a CONNECTED message followed by a lot of information, it means it has gone OK. You should be able to type a few things and then you will be disconnected.

```

Certificates — bash — 80x24

Carlos-Fernandez-Musoless-MacBook-Pro:Certificates carlos$ openssl x509 -in aps_
development.cer -inform der -out ECPNcert.pem
Carlos-Fernandez-Musoless-MacBook-Pro:Certificates carlos$ openssl pkcs12 -nocer
ts -out ECPNkey.pem -in ECPN.p12
Enter Import Password:
MAC verified OK
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
Carlos-Fernandez-Musoless-MacBook-Pro:Certificates carlos$ cat ECPNcert.pem ECPN
key.pem > ck.pem
Carlos-Fernandez-Musoless-MacBook-Pro:Certificates carlos$ telnet gateway.sandbo
x.push.apple.com 2195
Trying 17.172.233.66...
Connected to gateway.sandbox.push-apple.com.akadns.net.
Escape character is '^]'.
^C
^CConnection closed by foreign host.
Carlos-Fernandez-Musoless-MacBook-Pro:Certificates carlos$ openssl s_client -con
nect gateway.sandbox.push.apple.com:2195 -cert ECPNcert.pem -key ECPNkey.pem
Enter pass phrase for ECPNkey.pem:
CONNECTED(00000003)
```

6) Upload ck.pem to your server and modify php files

The last step is to upload the ck.pem file we created directly in the remote folder on your server where you have located all the PHP files.

You will have to modify the server version of SendECPNmessageToAll.php file and alter line 82 –you need to add your own pass phrase, the one you enter on step 5.

```

69 $result = curl_exec($ch);
70 if ($result == FALSE) {
71     die('Curl failed: ' . curl_error($ch));
72 }
73 //echo $result;
74
75 // Close connection
76 curl_close($ch);
77 }
78
79 function send_ios_notification($deviceIDs,$message) {
80     $ctx = stream_context_create();
81     stream_context_set_option($ctx, 'ssl', 'local_cert', 'ck.pem');
82     stream_context_set_option($ctx, 'ssl', 'passphrase', ' ');
83
84     foreach($deviceIDs as $deviceToken) {
85         // Open a connection to the APNS server
86         $fp = stream_socket_client(
87             'ssl://gateway.sandbox.push.apple.com:2195', $err,
88             $errstr, 60, STREAM_CLIENT_CONNECT|STREAM_CLIENT_PERSISTENT, $ctx);
89
90         if (!$fp)
91             exit("Failed to connect: $err $errstr" . PHP_EOL);
92
93         echo 'Connected to APNS' . PHP_EOL;
94
95         // Create the payload body
96         $body['aps'] = array(
97             'alert' => $message,
```

Please remember to check that you have modified the server version of the file and not just the local one.

7) Creating a sample project in Unity

Once you have all the certificates in place and the server up and running, here comes the easy bit! Open Unity and create a new project. Import the files that came with this package and open SampleScene.

You should be able to see an ECPNManager object. In the editor, click on the ECPNManager and enter the URL for your php files on your server (where you uploaded the PHP files).

You do not need to tweak anything else in the scene. All you have to do is go to Player Settings and enter a few options under the iOS category:

- Product name: ECPN
- Other settings > Bundle identifier: whatever you chose when creating your App ID; for me, this is com.CFM.ECPN

8) Build and enjoy!

Well, that's it, all that is left is to plug in your device, build the Sample scene and run it! For more information about how the plugin works in Unity, see the section Dissecting ECPN Plugin.

Google Cloud Messaging

The android counterpart of the Push Notifications is called Google Cloud Message, but other than the name it behaves in much the same way as the notifications in the apple world.

On the up side, they are much easier to set up, as they do not require certificate files or provisioning profiles, just having a google project ID and the right permissions set up in your manifest file. We will see how to do this in a minute (promise!)

On the down side, because the permissions need to be set for a specific package name, you will have to recompile the java plugin with the appropriate package name and file path if you wish to use the system on your own project. But don't worry! It is very simple to adapt, and we'll cover it all in this document.

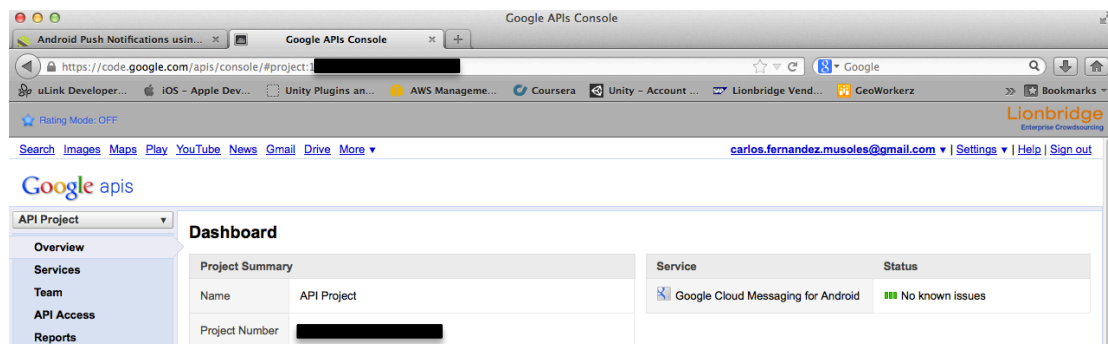
What you need

- Unity android
- Server with MySQL and PHP
- Java IDE (Eclipse, Dr Java) to recompile jar file, Google account –not to run the example, but if you want to incorporate the functionality to your own project)
- Android device (min sdk version=8, target sdk version = 16)

1) Register Google Project

First thing you need to do is register your project in Google with the following link: <https://code.google.com/apis/console/>

Once you have created, copy the project ID number for reference later (it appears in the URL at the end after '#project' and can also be found under the Dashboard as Project Number).



2) Activate Google Cloud Messaging for Android

Go to the Services section and set Google Cloud Messaging for Android to ON.

 Google Civic Information API 	<input type="checkbox"/> OFF	Courtesy limit: 25,000 requests/day
 Google Cloud Datastore API 	<input type="checkbox"/> OFF	Courtesy limit: 10,000,000 requests/day
 Google Cloud Messaging for Android 	<input checked="" type="checkbox"/> ON	
 Google Cloud Messaging for Chrome 	<input type="checkbox"/> OFF	Courtesy limit: 10,000 requests/day

3) Get API key

Under API Access you will find a screen showing something similar to this:

Simple API Access

Use API keys to identify your project when you do not need to access user data. [Learn more](#)

Key for browser apps (with referers)

API key: [REDACTED]

Referers: Any referer allowed

Activated on: Mar 24, 2013 9:08 PM

Activated by: carlos.fernandez.musoles@gmail.com – **you**

[Generate new key...](#)
[Edit allowed referers...](#)
[Delete key...](#)

[Create new Server key...](#)
[Create new Browser key...](#)
[Create new Android key...](#)
[Create new iOS key...](#)

Take a note of the API Key shown here –you may need to create an authorized one if you want to remove the limitations imposed by Google.

4) Modify the PHP files

Edit your settings.php file (included in this package within the PHP folder); on line 14 you should substitute the current key for your API key (created in step 3).

```

1  <?
2  // This is the url your database is located at.
3  // Consult webhost if you don't know it (should be of the form mysqlXXXXX.XXX:PORT)
4  $loginURL = [REDACTED];
5  // Username with access to read / write database
6  $username = [REDACTED];
7  // The password associated with $username
8  $password = [REDACTED];
9  //the name of the database that contains your data
10 $database = [REDACTED];
11 /*
12  * Google API Key
13  */
14 define("GOOGLE_API_KEY", [REDACTED]); // Place your Google API Key
15
16
17 ?>

```

5) Set up Unity project for android

Create a new android project in Unity and import all the files included in this package –including the files within the folder Plugins > Android, namely the androidmanifest and both jar files (gcm.jar and GCMJava.jar). You should be able to see a scene called SampleScene. Open the scene and find the object called ECPNManager to enter a few parameters:

- *Google Cloud Message Project ID*: This is your project ID that you got in step 1.
- *PHP files location*: this is the remote folder address on your server where you placed all the PHP files on the Server Set Up section.

Just to try the example out, go to Player settings and enter these parameters:

- Product name: ECPN
- Bundle identifier: com.CFM.ECPN

Please **note** that to use the system in another project of your own you need to tweak a few more things! We will cover these things in the section Importing ECPN to your Existing Project.

The avid user would have noticed that if you want to use the SampleScene in both android and iOS builds to test cross-platform notifications I asked you to set two different Bundle identifiers! For iOS we said you should set it to your own explicit bundle name, but for android I was very specific: com.CFM.ECPN.

The reason behind this is again the nature of how permissions work in android – they are bound to the package name. So in order to run the sample scene, just do as I said, when you build your iOS sample project, use your own bundle ID, but when building the android project, use com.CFM.ECPN. Of course this problem won't exist when you are using the package in your own project (you will be using only your own bundle ID for both versions!).

6) Build and enjoy!

Well, that's it, all that is left is to plug in your device, build the Sample scene and run it! For more information about how the plugin works in Unity, see the section Disecting ECPN Plugin.

Important note! If when you compile the project later on you get a **dx error** pointing at some of the classes within a '.jar' file please **remove** the android.jar file from your Assets folder (when importing the package, it is located within the Plugins>Android folder path.. There seems to be an incompatibility in between its classes and some of Unity's. This file is not important for your project, it is only used when you recompile your GCMJava.jar –see section below *Importing ECPN to your existing project – Android.*

How to use the SampleScene

After all the set up is complete, you will be dying to see a notification actually firing on your device! Well, this is it. Once you have built the SampleScene on

your devices (and this is common no matter what platform you are in), all you have to do is run it. You will see 4 buttons in a rather simplistic GUI:

- Register Device: sends a request to APNS / Google to register and retrieve the device ID.
- Unregister Device: if the device was registered, sends a request to unregister the device with APNS / Google.
- Send Message: sends a message to all previously registered devices. Remember that users will receive the notification whether or not they are currently running your app. For android this includes the device sending the message too!
- Exit: Well, it... quits the app.

As we mentioned before, the plugin only works on actual devices, but this restriction only applies to registrations. That means you can send messages from the Unity Editor (or anything else you can build your project in!) as long as you have some devices already registered in your database. This is handy for those of you with only one device at hand –register on the device, test on the editor!

Remember that notification messages may take several minutes to be delivered to your device.

Dissecting ECPN Plugin

The Unity plugin itself (ECPNManager.cs) is very simple and the public functions are platform-agnostic, so you don't have to check where your code is running, it does it for you! There are three functions that can be publicly called –and should be the ones that you will use to register your devices, send messages, etc.:

- void RequestDeviceToken(): Sends a request to APNS / Google to register the current device –to start receiving notifications.
- void RequestUnregisterDevice(): Sends a request to APNS / Google to unregister the current device –no longer receive notifications.
- void SendMessageToAll(): It sends a request to your server to broadcast a message to all registered devices.

Under the hood

The inner workings of the iOS code are very simple since unity has a built in class to handle the registration and handling of notification services called –quite obviously- NotificationServices. To register the device we use NotificationServices.RegisterForRemoteNotificationTypes(...) and poll for the deviceToken (which will be received asynchronously from APNS, hence the polling). Unregistration is done with the function UnregisterForRemoteNotifications() –what else??

The android part uses an external java plugin (GCMJava.jar, included in the package) to communicate with Google server. In a nutshell, when we call RequestDeviceToken() it sends a call to the java class, which in turn registers the

device (and your app) for notification services. As with iOS this is an asynchronous operation, so the plugin acts as a listener and waits in the background for a confirmation message from Google. Once it arrives, it sends the deviceToken to Unity.

No matter what platform is used, registration –and unregistration too- is not completed until you store the deviceToken information in your server database. This is how your server knows who to send notifications to later on. Once your deviceToken is received, the plugin automatically sends the necessary information to the PHP files on your server to register / unregister your user's device. The files that handle it are: RegisterDeviceIDtoDB.php and UnregisterDeviceIDfromDB.php. and they are responsible for storing deviceTokens in your database.

When a user calls ECPN.SendMessageToAll() it sends a request to your server which is handled by SendECPNmessageToAll.php. This file simply gets all deviceTokens previously stored in the DB and sends a notification message. It handles automatically how to send it –via APNS or Google- so you don't have to worry about determining which users own an android and which ones an iOS device.

Importing ECPN to your Existing Project

Naturally, after testing it with the sample scene, you will be anxious to import the ECPN system into your existing –or future- projects. If you have successfully managed to receive notifications on an android or iOS device, this should be the next step. If you haven't yet, I recommend you to follow all the steps for the iOS or android (or both, whatever suits you!) until you get the basic notifications right. You can of course jump ahead and try it on directly on your project, but due to the heavy dependencies on the permissions / certificates, it may be worth it to run the example first and then build from there. The choice is yours.

So I will assume you have the SampleScene working at this point. To incorporate ECPN functionality to your current project you will need a couple of tweaks, but nothing that should scare you after all those certificates and permissions!

iOS

Once you have the certificates bit done and the server set up (see sections above), it is as simple as dragging and dropping ECPNManager prefab included in the package and using the public functions within the ECPNManager.cs –as we do in the example MainScene.cs, but you can include them however you see fit. You need to set the PHP files Location parameter within ECPNManager object to point at the folder in your server where you have the PHP files.

Remember to set the Bundle identifier in Unity to match the one used in the iOS Developer Portal when creating the App ID. You are good to go!

Android

After setting up the server and completing the Google Project registration (see sections above) you will need to modify all package name references within the plugin to match your own bundle identifier (something like com.YOURNAME.YOURAPP). This can get a little tricky, so follow the steps set here and you will be receiving notifications in no time!

Prior to any steps, you need to import all the files in the package, which include the following:

- Plugin > Android > AndroidManifest (see details below), GCMJava.jar, gcm.jar (**note: you should not keep** android.jar in your Assets folder, it will be used later when recompiling GCMJava.jar, but should not appear within your project or it will throw dx compilation errors).
- ECPNManager prefab.
- PHP files (for convenience only)

1) ECPNManager prefab

First of all, drag and drop ECPNManager prefab into your project hierarchy and set the *Google CloudMessage Project ID* and *PHP Files Location* properties with the values you set in the sections above.

Set the *Package Name* property of the ECPNManager object to your own package name (for now on I will refer to com.YOUR.APP to your bundle ID; when you are doing this for yourself, please use your own bundle ID).

2) The AndroidManifest.xml file

If your project does not have an android manifest already, just go with the one included in the package and follow the instructions set in step 3 to modify it. If you do have one already, you will need to incorporate a few lines to it. Here are the important things you need to include:

- within the application block, add the following:

```
<!-- GCM java -->
<service android:enabled="true"
android:name="com.CFM.ECPN.GCMJava" /> <!-- android:name must
coincide with GCMJava package name + .GCMJava -->

    <receiver
        android:name="com.google.android.gcm.GCMBroadcastReceiver"
        android:permission="com.google.android.c2dm.permission.SEND" >
        <intent-filter>

            <!-- Receives the actual messages. -->
            <action android:name="com.google.android.c2dm.intent.RECEIVE" />
            <!-- Receives the registration id. -->
            <action android:name="com.google.android.c2dm.intent.REGISTRATION" />

            <category android:name="com.YOUR.APP" /> <!-- android:name must
coincide with GCMJava package name -->
        </intent-filter>
    </receiver>

    <service android:name="com.YOUR.APP.GCMIntentService" /> <!-- android:name
must coincide with GCMJava package name + .GCMIntentService-->
    <!-- end -->
```

- add the following permissions

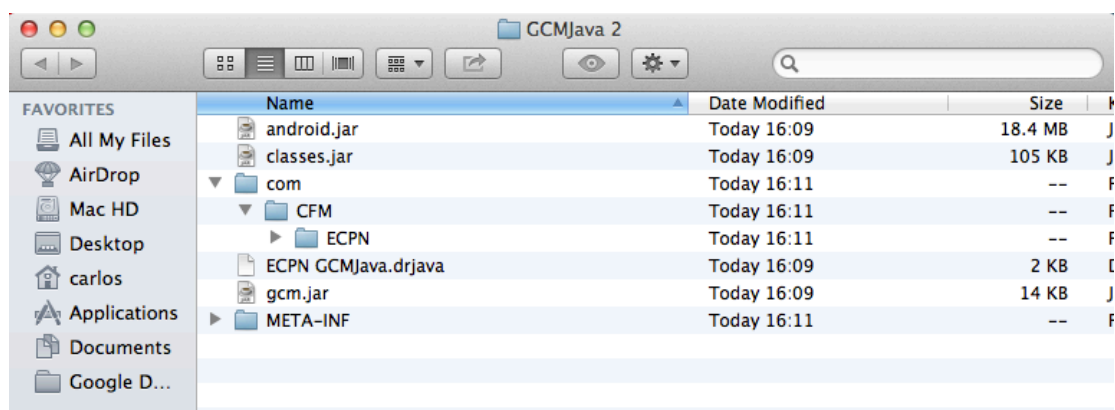
```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<!-- GCM requires a Google account. -->
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
<!-- Keeps the processor from sleeping when a message is received. -->
<uses-permission android:name="android.permission.WAKE_LOCK" />
<!-- Creates a custom permission so only this app can receive its messages. -->
<permission
    android:name="com.YOUR.APP.permission.C2D_MESSAGE"
    android:protectionLevel="signature" /> <!-- android:name must be of the form
"GCMJava.package.name" + ".permission.C2D_MESSAGE"-->
<uses-permission android:name="com.YOUR.APP.permission.C2D_MESSAGE" />
<!-- This app has permission to register and receive data message. -->
<uses-permission android:name="com.google.android.c2dm.permission.RECEIVE" />
<uses-sdk android:minSdkVersion="8" android:targetSdkVersion="16" />
```

3) Recompile GCMJava.jar to match your bundle ID.

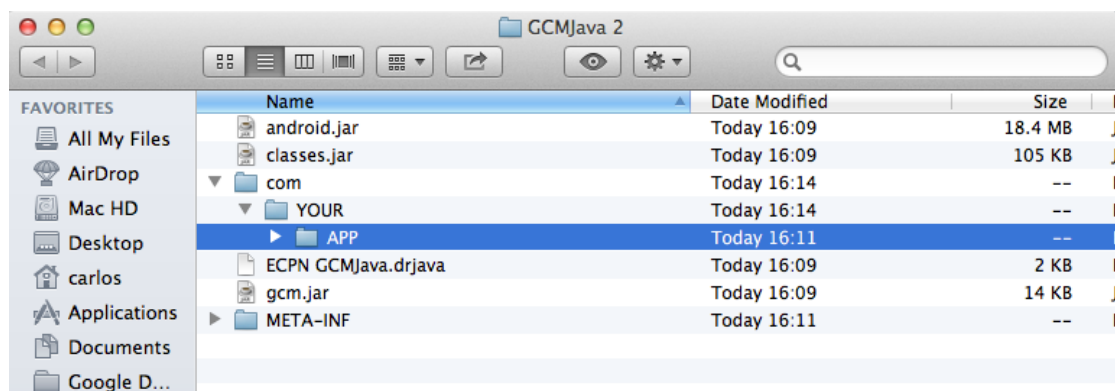
As we have seen in step 2, certain permissions have been set against your particular bundle identifier (in particular the ones that handle the reception of messages from Google). This is the reason we need to recompile the GCMJava.jar plugin with the appropriate package name, otherwise there would be a mismatch in between who is able to listen to the messages sent by Google (GCMJava) and who has permissions to listen to them (your app).

In order to recompile the GCMJava.jar project, uncompress it first. Jar files are just like zip files, so use your favourite software to unzip it (I personally like to rename the file to *.zip and let my operating system deal with it by double-clicking).

You should be able to see a folder structure like this one:



Do you see the folder structure com > CFM > ECPN? Well that is exactly matching your package name and bundle ID. To change it, first change the folder names to match your package name -in our example this was com.YOUR.APP, so the folder structure should look like:

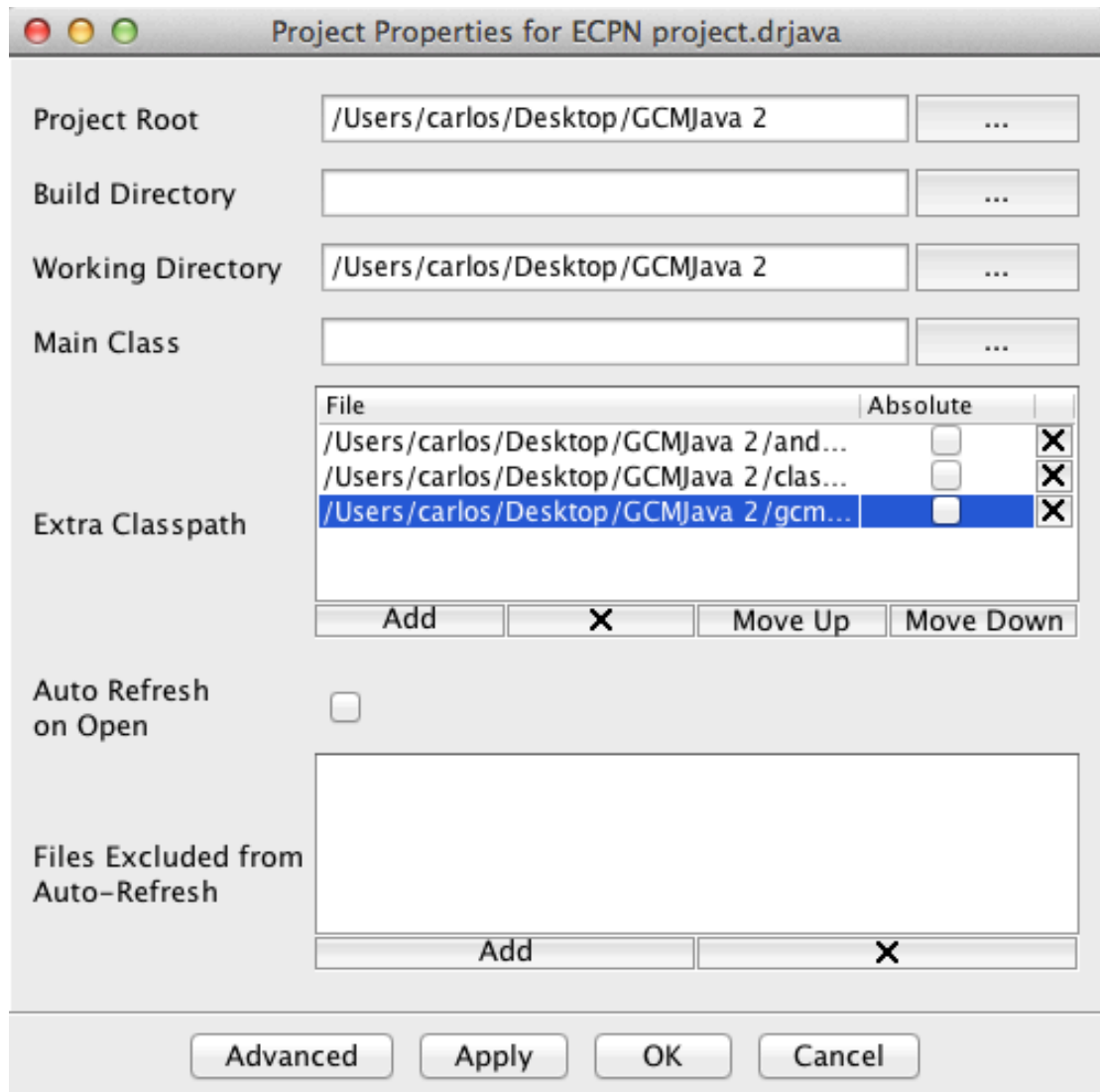


With your IDE of choice (I like [Dr Java](#) for its simplicity), create a new project with the package name com.YOUR.APP and import all the java files within the APP folder. In Dr. Java this is as simple as to open the application and creating a project; once opened, just open all the java files contained in the directory com>YOUR>APP and that's it, all imported!

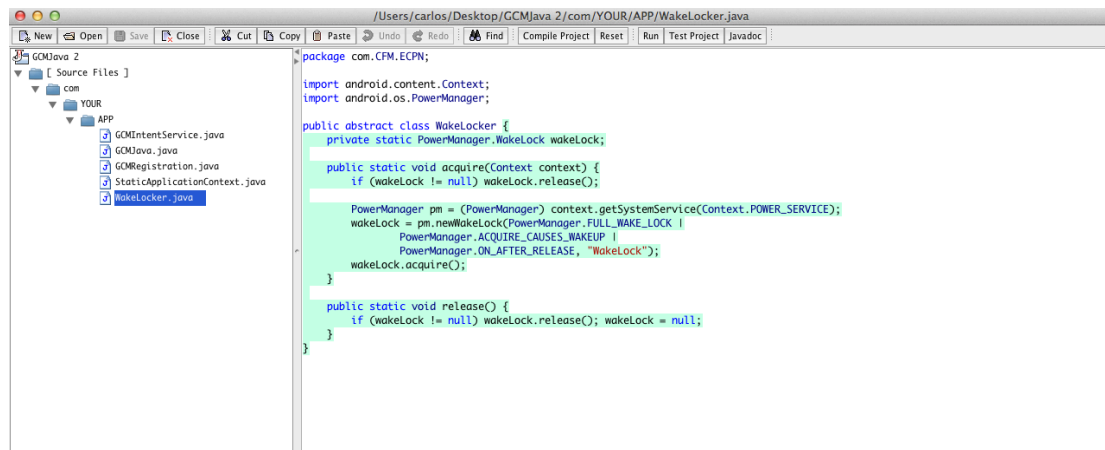
To compile the project you will need to include three files in the class path. Again this is very simple in Dr. Java, when you are creating the new project (Project>New) add the following files to the Extra Classpath:

- gcm.jar
- android.jar
- classes.jar

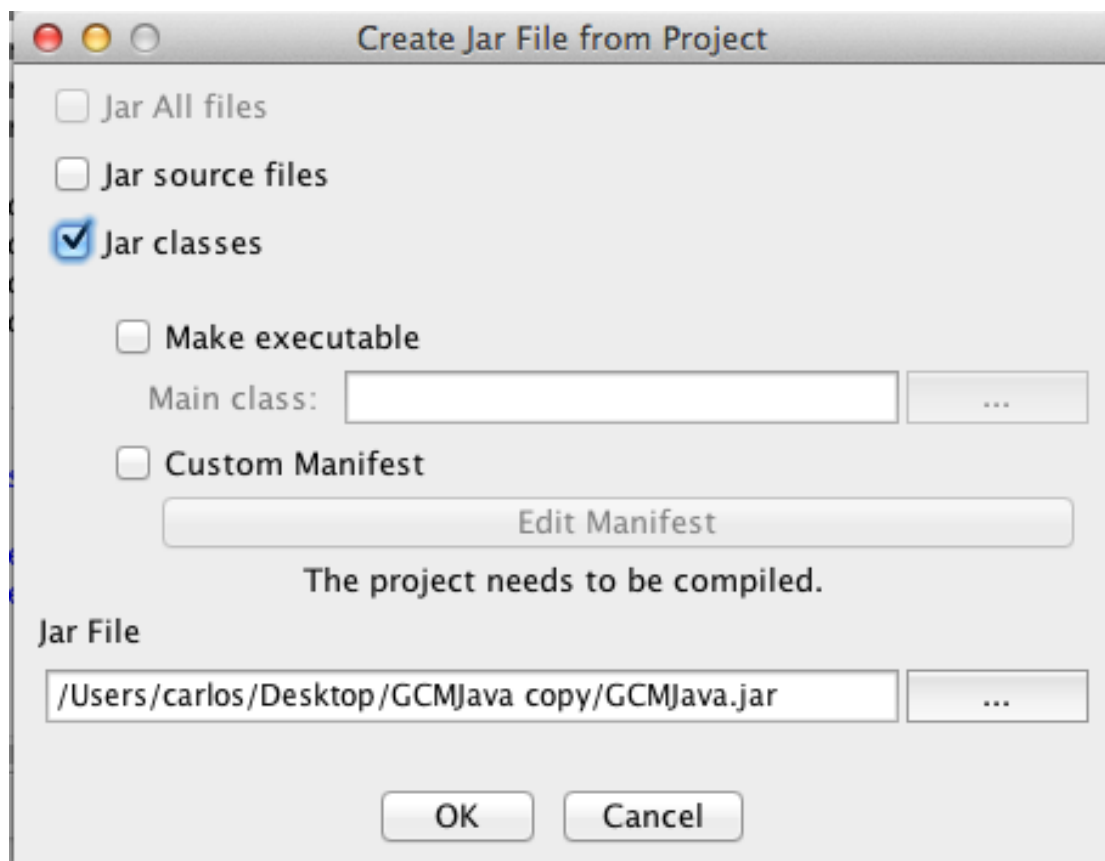
All the files can be found within the parent folder you have unzipped (note: android.jar is included within the unity package in Plugins>Android folder).



To import the java files in Dr. Java just open them with the newly created project! You should see the project structure:



Lastly, you need to change the package name on each of the java files to match your own (com.YOUR.APP); there is a package * statement on line 1 on all files. Save the files and compile the project (clicking on Compile Project button on top). To recreate the jar file, go to the menu Project > Create Jar File From Project. Select the option *Jar classes*. Set the Jar File name as GCMJava.jar –you can place it wherever you want, but remember where it is!).



If you have done it all correctly, you should see a success message and your new GCMJava.jar file should be created. Place it in your Unity project, under the Plugins>Android folder –substitute the old one if it is still there.

Now you are ready to use the ECPNManager functions within your new project to register, unregister and send messages! See section Dissection ECPN Plugin to learn how to use them.

Important note! It is important that you do **NOT** include the libraries (android.jar mainly) in the build. If you are using Dr. Java this is done by checking the 'Jar classes' checkbox only.

Room for improvement

Wow! You made it to the end, congratulations! Now you are able to set basic remote notifications in any project. But of course this service is not limited to sending a rigid message to all the devices using your app. There are infinite uses for notifications, and this plugin allows you to easily tweak its functions to increase its functionality. Here is a list of a few things I can think of that can be implemented nearly effortlessly:

Change notification icon

This is only for android, as iOS uses the default app icon to be shown by notifications. If you want another icon to be shown in your android devices, just replace the png file under the folder Plugins>Android>res>drawable. Place your new icon there and name it 'ic_launcher.png'. Done!

Change notification title

Also for android, you can define the notification title by changing a single line of code within the GCMJava.jar package! Go to GCMIntentService.java class and change the following property to match your fancy:

```
private static final String NOTIFICATION_TITLE = "ECPN";
```

Send messages to subsets of users

By altering the PHP file SendECPNmessageToAll.php, instead of pulling all users from the database at the beginning to populate the array \$contacts, you can be clever and only get a subset –how? That's up to you, maybe you send the list from Unity, or you have a secondary table with friend list... Possibilities are endless!

```

1  <?php
2  include("settings.php");
3
4  mysql_connect($loginURL,$username,$password);
5  @mysql_select_db($database) or die( "9");
6
7  $message = strip_tags($_POST["user"]) . " says hi";
8
9  // Get all device ids from table
10 $sql = "SELECT * FROM ECPN_table";
11 $result = mysql_query($sql);
12
13 while($row = mysql_fetch_array($result)) {
14     $contacts[] = $row['unityID'];
15 }
16

```

Reacting to notifications within your app

The current implementation of ECPN generates a local notification when receiving a remote message. In android it does this irrespective of whether the application is running or not –on iOS messages are ignored if the notification arrives while running the app.

Should you want your app to do something different upon reception of a message:

- On android: implement method protected void onMessage(Context context, Intent intent) within GCMIntentService.java class (which is part of the GCMJava.jar, see Importing ECPN to your existing Project section for more info). This method is called whenever the user device gets a message. Maybe you could send a message to your unity project by using the static method UnityPlayer.UnitySendMessage(...) –which will work only when the app is running.
- On iOS: if you want to know whether you get any notifications while the app is running you have to poll the following method [NotificationServices.GetRemoteNotification\(...\)](#), which will return a RemoteNotification object –if exists.

Making sure your messages get delivered

This is only for pros! If you want to ensure your messages get to the recipients you cannot rely on APNS / Google servers. Sometimes they are busy, and the messages may get lost in the virtual cyberspace. So if deliverance is a must for your project, why not create a table no your database that stores each message before being sent (for instance, within SendECPNmessageToAll.php routine). Then, when a message is actually received, send a message to your server to

remove the message entry. You could have a script to check every few minutes the new table to see if an old message is still waiting to be confirmed, and fire it again!

Version control

Changes on version 1.04

- Fixed blackout issue when using plugin. Changes to:
 - o AndroidManifest: instead of declaring GCMJava as an <activity>, now it is declared as a <service> (as described in this guide)
 - o GCMJava.java and GCMRegistration.java changed
- Changes to the guide about recompiling the GCMJava.jar –please read this section if you are recompiling the jar package and are having issues using the plugin.

Changes on version 1.02

- Added support for MS SQL (thanks to [Jacob Junker Larsen](#), a.k.a. **hrlarsen!**)
- Changes to SendMessageToAll.php (initialization of arrays)
- Change notification title info
- ECPNManager.cs #if UNITY_ANDROID wrapper to avoid compilation error under iOS environment

Changes on version 1.01

- AndroidManifest.xml changes:
 - o Android:name="StaticApplicationContext" no longer required
 - o Removed incompatibility with new devices:
android:targetSdkVersion="16" removed at the bottom
- Changes to the guide:
 - o Added warning about android dx compilation error and how to solve it
- Changes to ECPNManager.cs:
 - o Added code to gather application context from unity activity