

# UTNotifications Manual

Version 1.6

[Introduction](#)

[Getting Started](#)

[Creating Local Notifications](#)

[Custom User Data & Handling Notifications](#)

[Using Notification Profiles \(Sounds & Icons Settings\)](#)

[Image Notifications \(Android\)](#)

[Push Notifications Overview](#)

[What You Need for Push Notifications](#)

[General](#)

[iOS: Apple Push Notification Service \(APNS\)](#)

[Android: Google Cloud Messaging \(GCM\)](#)

[Android: Amazon Device Messaging \(ADM\)](#)

[Windows Store: Windows Push Notification Services \(WNS\)](#)

[Push Notifications Payload Format](#)

[Configuring the Apple Push Notification Service \(APNS\)](#)

[Provisioning Profiles and Certificates.](#)

[Generating the Certificate Signing Request \(CSR\)](#)

[Making the App ID and SSL Certificate](#)

[Converting the aps\\_development.cer File](#)

[Making the Provisioning Profile](#)

[Apply Credentials and Test](#)

[Configuring the Google Cloud Messaging \(GCM\)](#)

[Enable Google Cloud Messaging](#)

[Apply Credentials and Test](#)

[Configuring the Amazon Device Messaging \(ADM\)](#)

[Getting Your OAuth Credentials and API Key](#)

[Apply Credentials and Test](#)

[Configuring the Windows Push Notification Services \(WNS\)](#)

[Register your app with the Dashboard](#)

[Obtain the identity values for your app](#)

[Apply Credentials and Test](#)

[Unicode Support](#)

[Contacts](#)

## Introduction

[API Reference](#) | [Forum](#) | [Support Email](#) | [Issue Tracking](#)

UTNotifications is an advanced and professional Unity extension that is yet very convenient and easy to use. It provides a convenient cross-platform API for posting and handling local, scheduled (including those appearing once and those repeating) and push notifications. Currently it fully supports iOS, Android (Google Play and Amazon Kindle Android devices) and Windows Store (Windows Phone 8.1, Windows 8.1/10, Universal 8.1, Universal 10).

### Features:

- Immediate local notifications.
- Scheduled (those appearing once and those repeating) local notifications with automated restoring on device reboot.
- Push notifications.
- 2 Android push notifications services: Google Cloud Messaging (GCM) & Amazon Device Messaging (ADM) in a single build.
- Completely cross-platform API.
- The full source code is provided as well as the code of the native plugins so one can change and adjust anything one likes.
- A demo push notifications provider web server with the source code is included.
- Default or custom notifications sounds and icons.
- A detailed manual and an API Reference docs are included.
- Convenient Unity editor extension for configuring.
- Android & Windows Store manifest files automated patching.
- Notifications enabling/disabling API for all supported platforms allows one to add notifications toggle to the game options.
- API for handling clicked & received notifications of any type both local and push.
- One can attach custom data to the notification of any type and access it while handling the received notification.
- Hiding or cancelling a specific notification or all of them.
- Application icon badge number management API for iOS and Android.
- Android grouped notifications.
- Image notifications on Android.
- A sample & test scene.

UTNotifications consists of two main parts: Unity client extension and a demo server which shows you how to send push (remote) notifications. For the production version of your project you'll have to use your own game server or a dedicated notifications server but you can use the provided demo server source code as you like. There is also a number of third-party solutions for the push notification servers, such as free and open source [Uniqush](#), and services, such as [OneSignal](#), which are compatible with UTNotifications as it uses plain iOS, Google Play, Amazon & Windows push notifications services. You can also use the UTNotifications for local/scheduled notifications only, in that case you don't need any server. Unity client extension uses native plugins so you have to use any edition of Unity 5

(works fine even with Personal edition) or Unity 4.x Pro. Neither iOS Pro nor Android Pro are required. Minimal supported version of Unity is 4.6.

## Getting Started

Once you installed the UTNotifications asset into your project, you're able to open its settings from Unity menu: *Edit -> Project Settings -> UTNotifications* (Unity restart may be required first time to see this menu item).

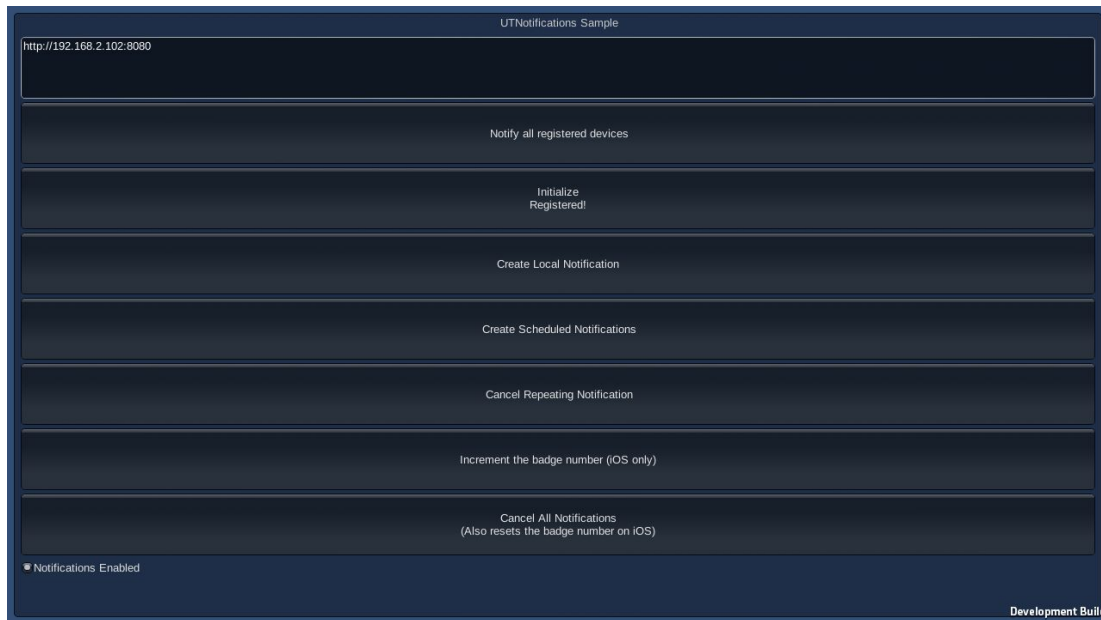
The image shows the 'Inspector' window in Unity, displaying the 'UTNotificationsSettings' asset. The window is organized into several sections with expandable/collapsible headers. The 'Help' section contains links to Manual, API Reference, Forum, Report Issue, Feedback, and Support Email. The 'Notification Profiles (Sounds & Icons)' section shows a 'default' profile and a '+' button to add more. The 'iOS' section has checkboxes for 'Push Notifications' and 'Restore Notifications On Reboot', a text field for 'Bundle Identifier (Package Id)', and a dropdown for 'APNS Registration Id encoding' set to 'HEX'. The 'Android' section includes a dropdown for 'Show Notifications' set to 'WHEN\_CLOSED\_OR\_IN\_BACKGROUND', checkboxes for 'Push Notifications' and 'Restore Notifications On Reboot', a dropdown for 'Grouping Mode' set to 'BY\_NOTIFICATION\_PROFILES', a 'More info...' button, an informational message about group summaries, and a checkbox for 'Show Only Latest Notification'. The 'Google Cloud Messaging' section has a checkbox for 'Push Notifications' and a 'Load google-services.json' button. The 'Amazon Device Messaging' section has a checkbox for 'Push Notifications' and text fields for 'Bundle Identifier (Package Id)', 'Android debug signature MD5', and 'Amazon Debug API Key'. The 'Windows Store' section has a checkbox for 'Push Notifications', a text field for 'Certificate', a text field for 'Identity Name', and a checked checkbox for 'Notify only when app is closed or hidden'. The 'Advanced' section is expanded, showing 'Push Payload Format (GCM, ADM, WNS)' with a table of settings: Title (data/title), Text (data/text), User data parent (opt) (data/), Notification profile (opt) (data/notification\_profile), Id (opt) (data/id), and Badge (opt) (data/badge\_number).

Push Payload Format (GCM, ADM, WNS)	
Title	data/title
Text	data/text
User data parent (opt)	data/
Notification profile (opt)	data/notification_profile
Id (opt)	data/id
Badge (opt)	data/badge_number

Local notifications doesn't require any additional setting up. Configuring of push notifications services is [described below](#).

There is an example scene:

`Assets/UTNotifications/Sample/UTNotificationsSample.unity` which you can use to get familiar with most of the UTNotifications features and how to use them. It also helps you to check if the configuration is correct.



You can also add the `UTNotifications.UTNotificationsSample` script (`Assets/UTNotifications/Sample/UTNotificationsSample.cs`) to any `GameObject` in your own scene to access this test menu. Please note that notifications are not available in some device emulators and in the Unity editor, so please deploy to a device in order test or debug notifications related functionality.

You can find an API Reference in UTNotifications Unity Settings and [here](#).

## Creating Local Notifications

Local notifications are notifications, shown by request of the client application itself. With UTNotifications you can show immediate, scheduled and repeated scheduled local notifications.

First thing you need to know, is that entire UTNotifications API is placed in a namespace `UTNotifications`. So you may want to add a using statement to easily access that namespace:

```
using UTNotifications;
```

Now let's Initialize `UTNotifications.Manager`. It should be done before accessing any UTNotifications methods. Some `MonoBehaviour's Awake()` or `Start()` method is a good place for it:

```
public void Start()
{
    UTNotifications.Manager.Instance.Initialize(false);
}
```

Note, that `UTNotifications.Manager.Instance` (or just `Manager.Instance` if you added `using UTNotifications`) is a main access point to all `UTNotifications` methods. It returns a singleton instance of the `UTNotifications.Manager` class. We provided `false` here as a value of the argument `willHandleReceivedNotifications` of `UTNotifications.Manager.Initialize` as we don't have an intention to handle shown notifications right now. For more info on handling notifications, please see the [appropriate section](#).

Now you can start creating local notifications. F.e.:

```
UTNotifications.Manager.Instance.PostLocalNotification("Title", "Text", 1);
```

It creates an immediate local notification with title "Title", text "Text" and id = 1. Notification ids are used to identify each notification. F.e. a new notification with the same id as an old one replaces that old notification instead of creating second separate notification. id is also used to hide or cancel a specific notification (see an API Reference for details).

**Note** that with default settings, you will not see or hear any immediate notifications on any platforms (because by default notifications are not shown while an application is running). You can configure that behaviour in `UTNotifications Settings`: **Common Android Settings** -> **Show Notifications & Windows Store Settings** -> **Notify only when app is closed or hidden**. Unfortunately, iOS doesn't allow to control this behaviour - you will never see notifications while an app is running on iOS.

Let's now schedule a local notification:

```
UTNotifications.Manager.Instance.ScheduleNotification(15, "Title", "Text", 2);
```

It will create a local notification with title "Title", text "Text" and id = 2, which will be triggered in 15 seconds after that code is executed. You can also specify a `System.DateTime` value as a first argument. It will be a date and time to trigger a notification.

Similarly, you can create a repeated scheduled notification:

```
UTNotifications.Manager.Instance.ScheduleNotificationRepeating(5, 25, "Title", "Text", 3);
```

This notification with title "Title", text "Text" and id = 3 will be shown first time in 5 seconds after that code is executed and then will be repeated every 25 seconds. There is also a `System.DateTime` version of this method.

**Note** that the repeating times are approximate, and may differ, especially on iOS, where only fixed options like every minute, every day, every week and so on are available. So the provided interval value will be approximated by one of the available options.

On Android, there is a way to show a notification, containing an image:

```
Manager.Instance.ScheduleNotification(10, "Image Notification", "Image notification text",
    4, new Dictionary<string, string>
    {
        {"image_url", "http://thecatapi.com/api/images/get?format=src&type=png&size=med"}
    });
```

For more details on image notifications see [Image Notifications \(Android\)](#).

You can also configure notifications icons and sounds. For more details see [Using Notification Profiles \(Sounds & Icons Settings\)](#).

## Custom User Data & Handling Notifications

UTNotifications provide a way to handle a list of all notifications shown before or when an app was running, and also a notification which was clicked by user. Besides, each of notifications (local and push) can contain some custom data, which can be read when handling a clicked or received notification.

For that you can subscribe on UTNotifications.[Manager](#) events `OnNotificationClicked` & `OnNotificationsReceived` before initializing UTNotifications. F.e.:

```
UTNotifications.Manager notificationsManager = UTNotifications.Manager.Instance;
```

```
notificationsManager.OnNotificationClicked += (notification) =>
```

```
{
    Debug.Log(notification.text + " clicked");
};
```

```
notificationsManager.OnNotificationsReceived += (receivedNotifications) =>
```

```
{
    foreach (var notification in receivedNotifications)
    {
        Debug.Log(notification.text + " received/triggered");
    }
};
```

```
notificationsManager.Initialize(true);
```

Here we provided `true` as a value of the argument `willHandleReceivedNotifications` of `UTNotifications.Manager.Initialize`, as we want to handle received notifications with `OnNotificationsReceived`. Never set it to `true` if you don't want to handle received notifications, as it can be heavy for an app performance. Handling clicked notifications doesn't require `true` for `willHandleReceivedNotifications` argument.

**Note** that iOS doesn't provide a list of all notifications shown when an app wasn't running in foreground. Received notifications list will contain only a notification, which was clicked and all notifications shown when an app is running in foreground. On other platforms, you'll receive a list of all shown notifications, even ones shown when an app was closed.

You can provide a `<string, string>` dictionary, containing any custom data, which can then be accessed when handling clicked or received notifications as `ReceivedNotification.userData`. Each of methods for creating local notifications can accept an optional value `userData`. Push notifications' payload is used to get a value of the `userData` when handling them.

F.e. with local notifications:

```
Dictionary<string, string> userData = new Dictionary<string, string>();  
userData.Add("event_type", "DAILY_GIFT_RECEIVED");
```

```
Manager.Instance.ScheduleNotificationRepeating(DateTime.Now.AddDays(1),  
    TimeUtils.DaysToSeconds(1), "A gift for you!", "Start the game to receive your gift", 5,  
    userData);
```

F.e. with push notifications (ADM payload format):

```
{  
    "data":  
    {  
        <...>,  
        "event_type": "DAILY_GIFT_RECEIVED"  
    }  
}
```

And then let's handle it:

```
// Should be subscribed before initializing UNotifications.Manager  
UNotifications.Manager.Instance.OnNotificationClicked += (notification) =>  
{  
    if (notification.userData != null && notification.userData.ContainsKey("event_type"))  
    {  
        string eventType = notification.userData["event_type"];  
        switch (eventType)  
        {  
            case "DAILY_GIFT_RECEIVED":  
                ShowDailyGiftDialog();  
                break;  
  
            default:  
                Debug.LogWarning("Unexpected event_type: " + eventType);  
                break;  
        }  
    }  
};
```

## Using Notification Profiles (Sounds & Icons Settings)

By default any notification will be posted with a default system notification sound and the application icon. UNotifications allow to define custom sounds and icons for notifications (custom notification icons are not supported by iOS, no customization is currently supported

on Windows Store). What sound and icon is used for a specific notification is defined by a **notification profile** - named set of the notification options.

For example, the game has two kinds of notifications - when a player receives a gift and when some in-game research is complete. You then can define two notification profiles: "gift" & "research\_complete". First one will use a gift box icon when shown and some specific sound, and second one will have a bulb icon and another sound.

You can create and edit the notification profiles in UTNotifications editor: *Edit -> Project Settings -> UTNotifications -> Notification Profiles (Sounds & Icons)*.

Each of functions `UTNotifications.Manager.Instance.PostLocalNotification` and

`UTNotifications.Manager.Instance.ScheduleNotification` has the optional

argument **string notificationProfile** which defines a name of a notification profile used for this notification.

For the push notifications you can also specify a notification profile.

- **iOS (APNS).**

Notification profile name is specified as a sound in the APNS json payload:

```
{
  "aps":
  {
    <...>
    "sound" : "Data/Raw/<NOTIFICATION PROFILE NAME>"
  }
}
```

Note that **<NOTIFICATION PROFILE NAME>** should not contain any file extension.

- **Android.**

Notification profile name is stored in the "data" node of the notification json.

**GCM:**

```
{
  "registration_ids":<...>,
  "data":
  {
    <...>,
    "notification_profile": "<NOTIFICATION PROFILE NAME>"
  }
}
```

**ADM:**

```
{
  "data":
  {
    <...>,
    "notification_profile": "<NOTIFICATION PROFILE NAME>"
  }
}
```

- **Windows Store (WNS):**

Notification profile name is stored in the payload json root node:



```
{
  <...>,
  "notification_profile": "<NOTIFICATION PROFILE NAME>"
}
```

Push notifications sent from the `UTNotificationsSample` (Notify all registered devices) use the notification profile `"demo_notification_profile"`. You can try configuring a profile with that name to see how the feature works. If the requested notification profile is not found, the default sound and icons will be used.

There is a predefined profile `"default"`, which is used on Android when no notification profile is specified for a notification. It's **important** to configure at least its Small Icon (Android 5.0+): Android, starting with a version 5.0, ignores any color information of small notification icons: the icons are considered to be completely white and only alpha channel of the icons is applied (so icons can be only white & transparent). So any non-transparent icons turn into just white squares when using as small notification icons.

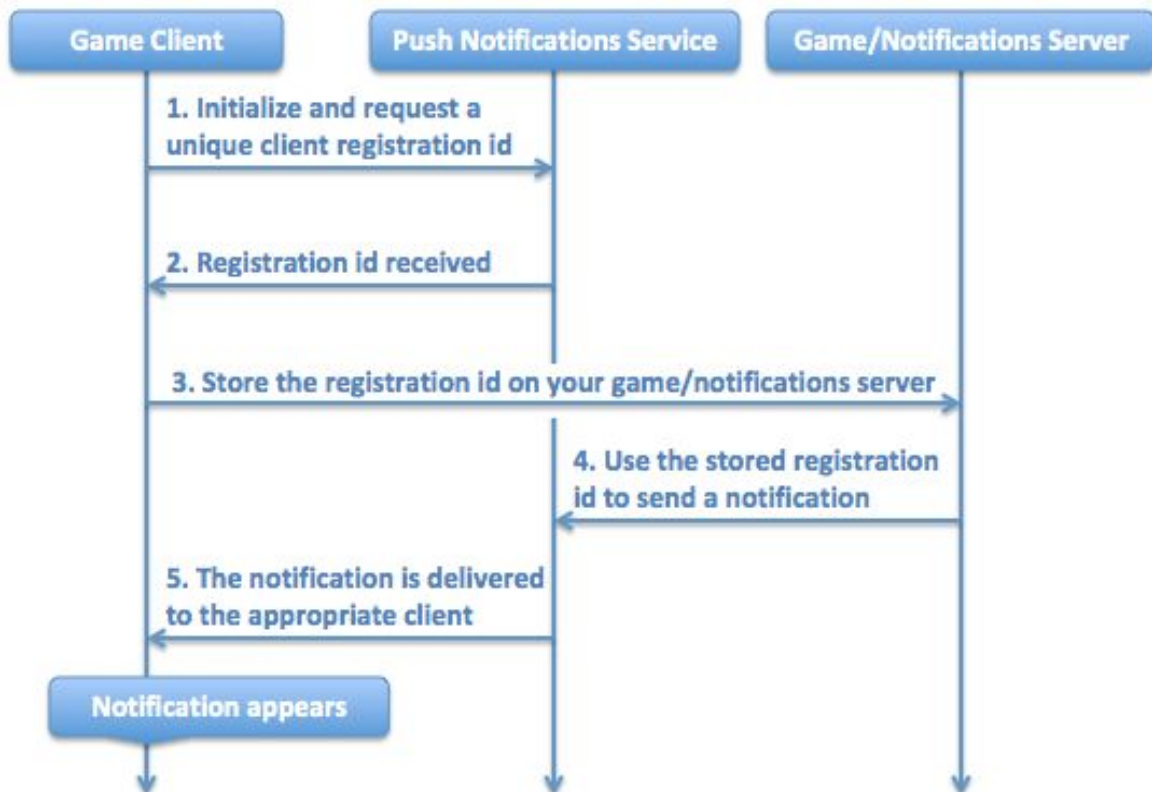
## Image Notifications (Android)

With `UTNotifications` you can create image notifications, i.e. notifications containing large images. It's supported with both local and push notifications. In order to create an image notification [add a user data argument](#) `"image_url"` with a string value, containing an URL of a picture to use. `"image_url"` value may be a normal `http://` or `https://` URL, or an Android file system URL: `file:///<full path to a picture file>`.

## Push Notifications Overview

Push notifications, also known as server notifications or remote notifications, are the notifications to a device without a specific request from the client. Unlike local notifications, which don't include any server part, push notifications always originate from a server. Different devices rely on different methods to deliver push notifications. Apple, for example, uses the Apple Push Notification Service. Android doesn't have a common system, but different Android devices provide different push notifications services. **Google Play** featured ones (i.e. most of Android devices) use Google Cloud Messaging (GCM) and Firebase Cloud Messaging (FCM) (`UTNotifications` are going to use FCM on Android instead of GCM starting with version 1.7). Amazon Android devices (entire **Kindle Fire** series) don't support GCM and have their own Amazon Device Messaging (ADM) API. Windows 8.1+ and Windows Phones use Windows Push Notification Services (WNS). `UTNotifications` rely on OS specific push notifications systems internally, but externally provides the common for all the supported services client side API.

No matter what OS and service is used, the general scheme is the same:



1. **Initialize and request a unique registration id.** The client application using a push notifications service (“PNS”: one of APNS, GCM, ADM and WNS) API requests a unique identifier for that specific PNS of that specific application on that specific device. Please note that in general it should be done on every start of the app because this identifier can get out of date and the application would receive a new one. With `UTNotifications` it’s done by calling `UTNotifications.Manager.Instance.Initialize(...)` function.
2. **Registration id received.** The application (game client) receives the id from PNS API asynchronously or synchronously. In order to receive it you will subscribe to `UTNotifications.Manager.Instance.OnSendRegistrationIdEvent` (please subscribe before calling the `Initialize` function because in some cases receiving the registration id may be done synchronously).
3. **Store the registration id on your game/notifications server.** You send the received id to your own server which will later send push notifications. You do it in the delegate subscribed to the `OnSendRegistrationIdEvent`.
4. **Use the stored registration id to send a notification.** Your server requests the server side of PNS API to send (i.e. “push”) custom notification to one or more clients using their registration ids which were previously stored. Please see `DemoServer.PushNotificator` class source code (`Assets/UTNotifications/Editor/DemoServer/src/DemoServer/PushNotificator.java`) for an example.

5. **The notification is delivered to the appropriate client.** PNS delivers the notification to the client with specified registration id. You don't have to do anything on this stage with UTNotifications (cause it takes care of everything with both Android PNSes and there is nothing to be done on iOS). A click on the notification will open your application: it's being started if has't been and goes foreground if it was in a background. If you would like to handle incoming notifications please see API Reference for `UTNotifications.Manager.OnNotificationsReceived` event and `UTNotifications.Manager.Initialize(...)` function.

Please note that every push notification service requires some configuring. This is described in the sections below.

## What You Need for Push Notifications

### General

- A server that is connected to the internet. Push notifications are always sent by a server. For development you can use your computer as the server but for production use, you need at least something like a VPS (Virtual Private Server). A cheap shared hosting account is not good enough in most cases. You need to be able to run a background process on the server and be able to make outgoing TLS connections on certain ports.

### iOS: Apple Push Notification Service (APNS)

- An iPhone or iPad. 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 (this is described below).
- An OS X computer.

### Android: Google Cloud Messaging (GCM)

- Any Google Play featured device with Android 2.3.3+.

### Android: Amazon Device Messaging (ADM)

- Any Amazon Kindle Fire device (tablet or phone) except the 1st generation of Kindle Fire tablets which don't support push notifications.

### Windows Store: Windows Push Notification Services (WNS)

- Any Windows Phone 8.1 or Windows 8.1/10 device.

## Push Notifications Payload Format

APNS requires any push notifications sent by your server to have a specific format. It is describe [in this document](#).

Unlike it, GCM, ADM & WNS don't have one common format of the message payload. Each of them accepts a JSON data payload, which then is interpreted by the client application. The client application itself is responsible for creating notifications based on the payload received from the appropriate service. Fortunately, UTNotifications does this nasty job for

you. This is why it requires the JSON payload to be in a specific format, which though you can configure. The default format looks like:

**GCM:**

```
{
  "registration_ids":["<id1>", ...], <or "to":"id1",>
  "data":
  {
    "title":"<Title>",
    "text":"<Text>",
    ["id":<int id>],
    ["badge_number":<int badge>],
    ["notification_profile":"<profile name>"],
    ["image_url":"<picture URL>"],
    ["<User data key 1>":"<User data value 1>",
    ...]
  }
}
```

**ADM:**

```
{
  "data":
  {
    "title":"<Title>",
    "text":"<Text>",
    ["id":<int id>],
    ["badge_number":<int badge>],
    ["notification_profile":"<profile name>"],
    ["image_url":"<picture URL>"],
    ["<User data key 1>":"<User data value 1>",
    ...]
  }
}
```

**WNS:**

```
{
  "title":"<Title>",
  "text":"<Text>",
  ["id":<int id>],
  ["badge_number":<int badge>],
  ["notification_profile":"<profile name>"],
  ["image_url":"<picture URL>"],
  ["<User data key 1>":"<User data value 1>",
  ...]
}
```

If push server you're going to use sends push messages in a different format, you can configure it in the UTNotifications Unity settings: Edit -> Project Settings -> UTNotifications -> Advanced -> Push Payload Format (GCM, ADM, WNS) . "data/" prefix is always added to each of the field names (but it's ignored on WNS).

## Configuring the Apple Push Notification Service (APNS)

This section was created using this article:

<http://www.raywenderlich.com/32960/apple-push-notification-services-in-ios-6-tutorial-part-1>

Here is described configuring the APNS for an application called PushChat, you'll replace it with your project name.

### Provisioning Profiles and Certificates.

To enable push notifications in the iOS version of 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.



**Apple Development Push S**  
Issued by: Apple Worldwide Devel  
Expires: Sunday, August 7, 2011 1  
This certificate is valid

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 manual, you won't bother with the distribution profiles and certificates and just use the ones for development. But steps for the production certificates are very similar (you just choose "production" instead of "development" on every step that contains this option). Of course there is no need to create an extra App ID for it, you just need to setup its "production" part (find more about it below).

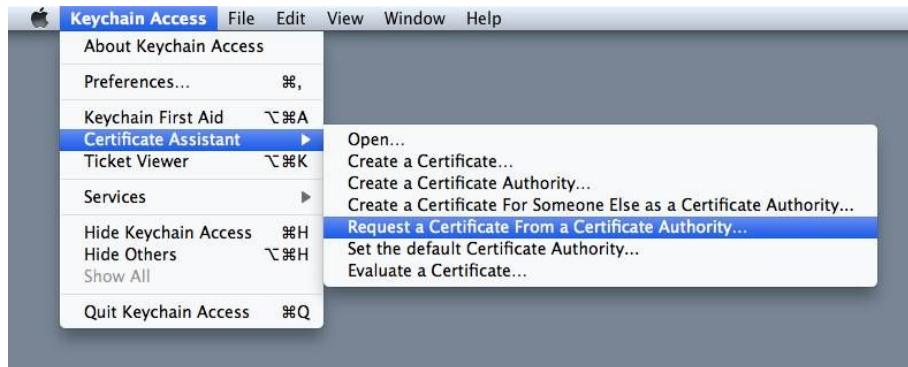
### Generating the Certificate Signing Request (CSR)

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 Certificate](#) first. 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.

You can type anything you want for **Common Name**, but choose something descriptive. This allows us to easily find the private key later. Let’s type **PushChat** here.

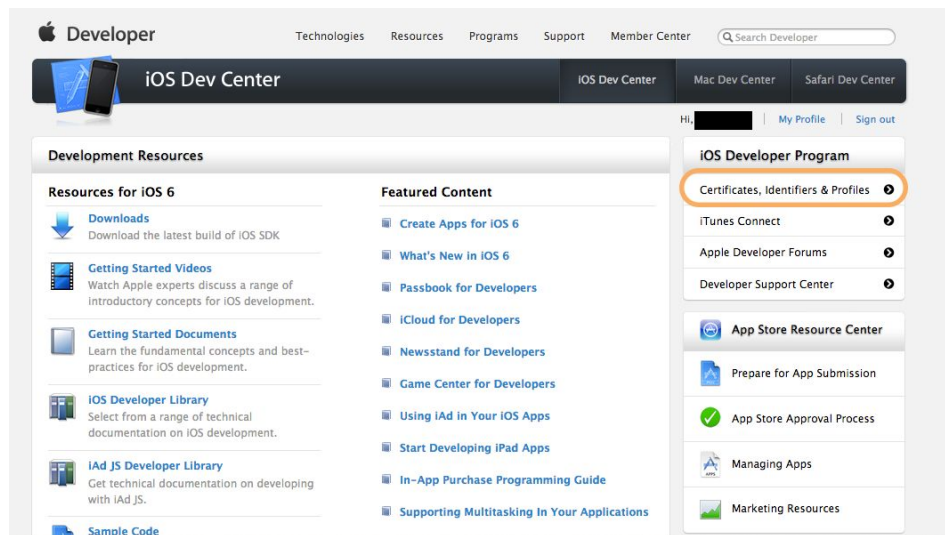
Check **Saved to disk** and click **Continue**. Save the file as

“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.

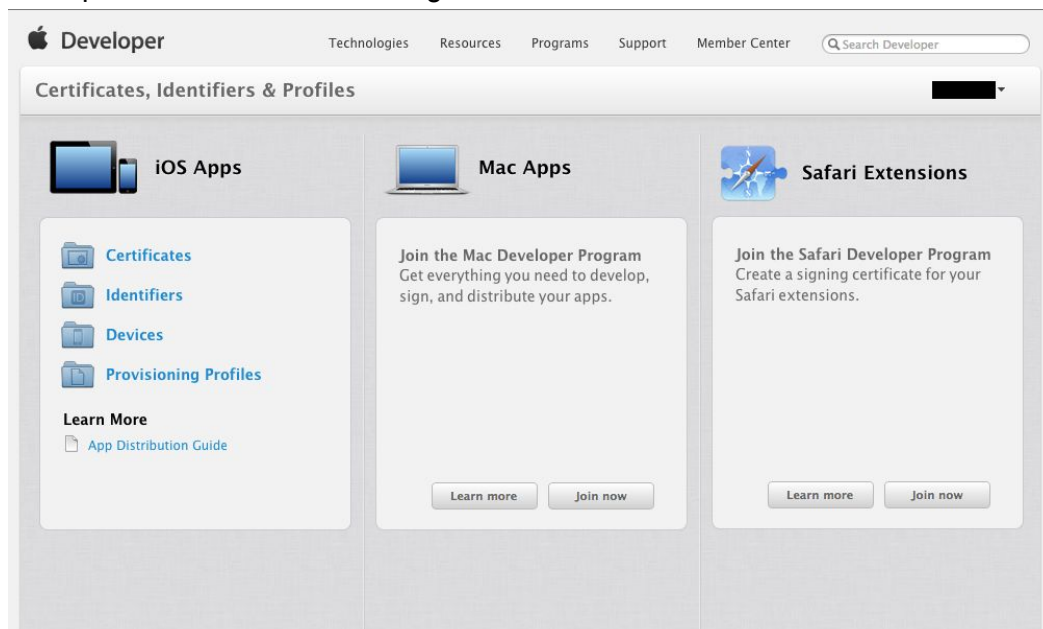
## Making the App ID and SSL Certificate

Log in to the [iOS Dev Center](#) and “**Select the Certificates, Identifiers and Profiles**” from the right panel.



**Note:** App ID creation doesn't work with some browsers (f.e. I had issues with Google Chrome) so please do it in **Safari**.

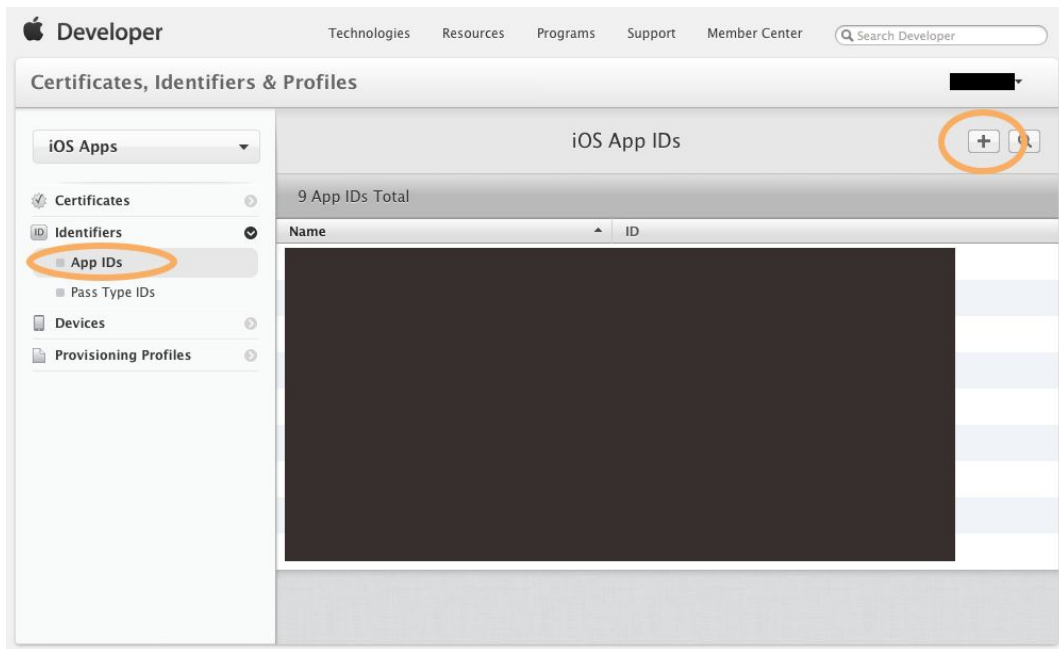
You will be presented with the following screen:



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.



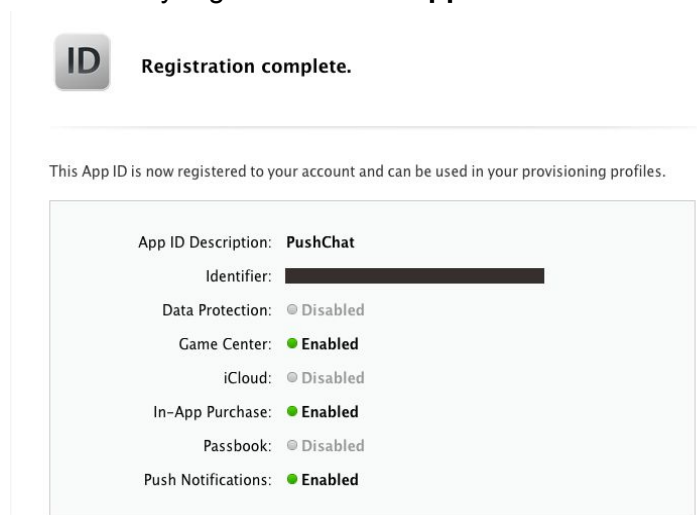


Fill the following details:

- App ID Description: PushChat.
- **Explicit App ID:** f.e. com.uttest.pushchat. You'll have to type your own **Bundle Identifier** here – the same as set in [Unity Player Settings for iOS](#) – instead of using this example one. **Note** that if the Bundle Identifier in Unity Player Settings for iOS is "com.Company.ProductName" you have to replace it by your own first!
- App Services Check the **Push Notifications Checkbox**.

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:






Name: PushChat  
 Prefix:   
 ID: .pushchat

Application Services:		
Service	Development	Distribution
App Group	<input type="radio"/> Disabled	<input type="radio"/> Disabled
Associated Domains	<input type="radio"/> Disabled	<input type="radio"/> Disabled
Data Protection	<input type="radio"/> Disabled	<input type="radio"/> Disabled
Game Center	<input checked="" type="radio"/> Enabled	<input checked="" type="radio"/> Enabled
HealthKit	<input type="radio"/> Disabled	<input type="radio"/> Disabled
HomeKit	<input type="radio"/> Disabled	<input type="radio"/> Disabled
Wireless Accessory Configuration	<input type="radio"/> Disabled	<input type="radio"/> Disabled
iCloud	<input type="radio"/> Disabled	<input type="radio"/> Disabled
In-App Purchase	<input checked="" type="radio"/> Enabled	<input checked="" type="radio"/> Enabled
Inter-App Audio	<input type="radio"/> Disabled	<input type="radio"/> Disabled
Apple Pay	<input type="radio"/> Disabled	<input type="radio"/> Disabled
Passbook	<input type="radio"/> Disabled	<input type="radio"/> Disabled
Push Notifications	<input checked="" type="radio"/> Configurable	<input checked="" type="radio"/> Configurable
VPN Configuration & Control	<input type="radio"/> Disabled	<input type="radio"/> Disabled


Edit

Notice in the **“Push Notifications”** 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 **Edit** button to configure these settings.

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





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.

 <b>Development SSL Certificate</b>	
Create certificate to use for this App ID.	<div>Create Certificate...</div>
 <b>Production SSL Certificate</b>	
Create certificate to use for this App ID.	<div>Create Certificate...</div>

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.

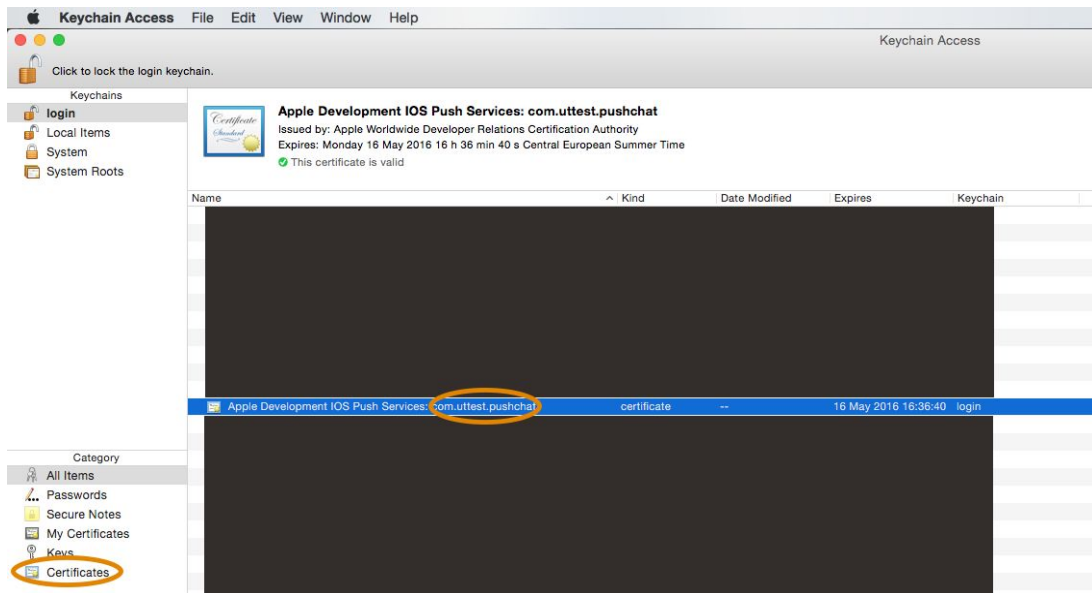
**Note: The production certificate remains valid for a year, but you can renew it before the year is over to ensure there is no downtime for your app.**

### Converting the `aps_development.cer` File

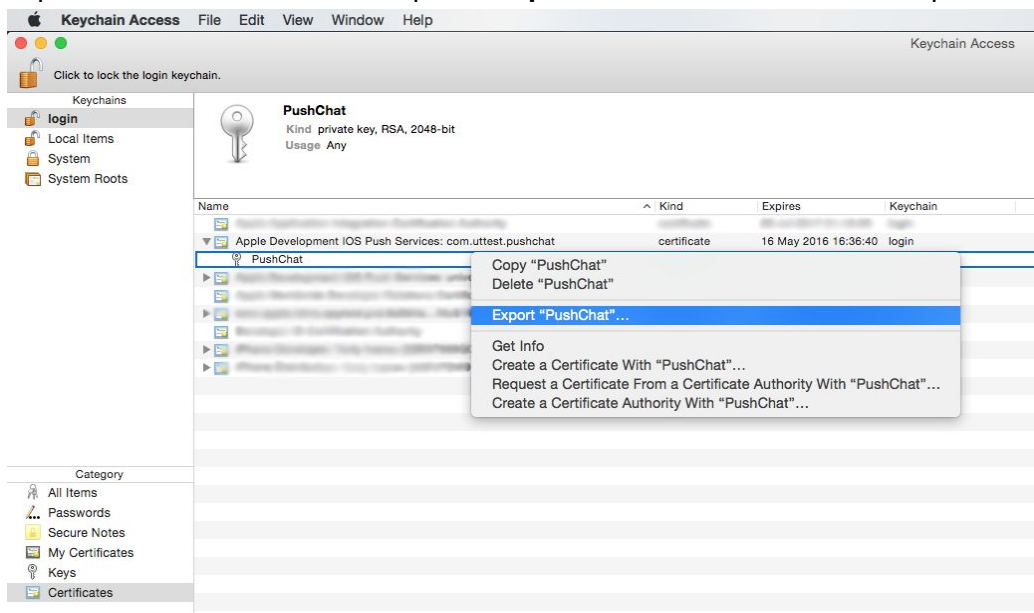
The `aps_development.cer` certificate will be used by the server to send push notifications. However most servers (and the provided **DemoServer**) don't work with `.cer` files directly but support `.p12` file format instead so we'll have to convert it. If your server can work directly with `.cer` files you can just skip this step.

1. Double-click on the `aps_development.cer` file to import it to the Keychain. You'll see a new item like “Apple Development IOS Push Services: `com.uttest.pushchat`” in **All items** list.

2. Select the **Certificates** category:



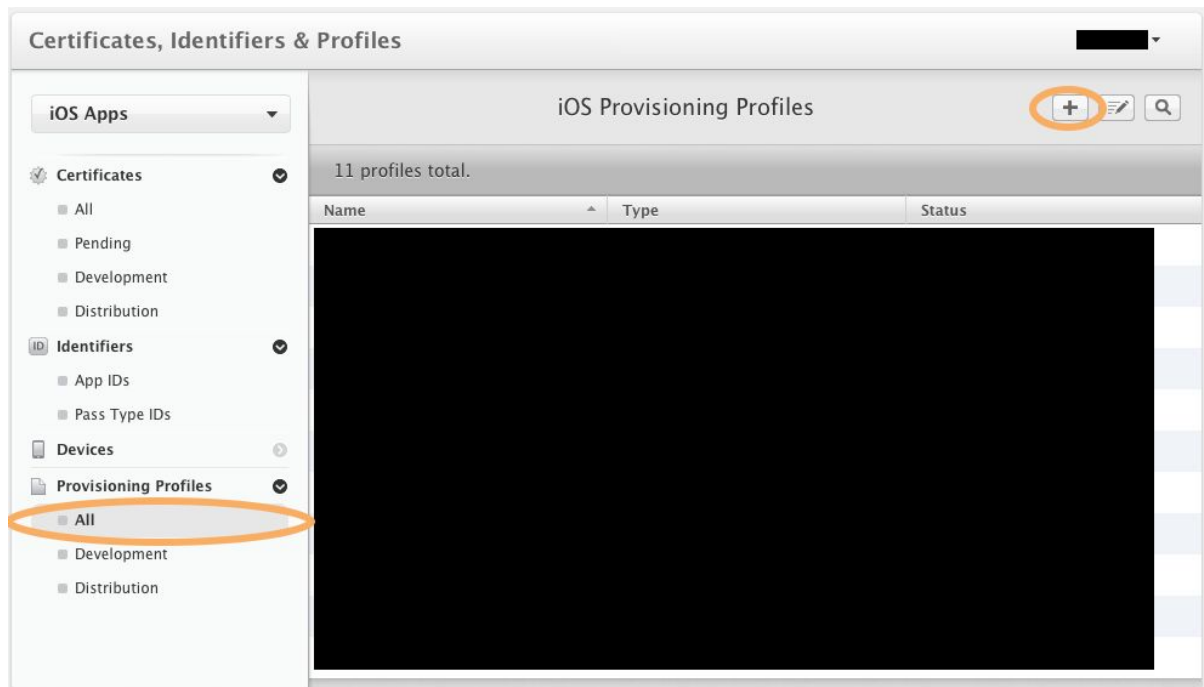
3. Expand the certificate item and press **Export...** in a context menu of its private key:



4. Select the `.p12` file name and location, click **Save** and choose a password. The key file is ready to use.

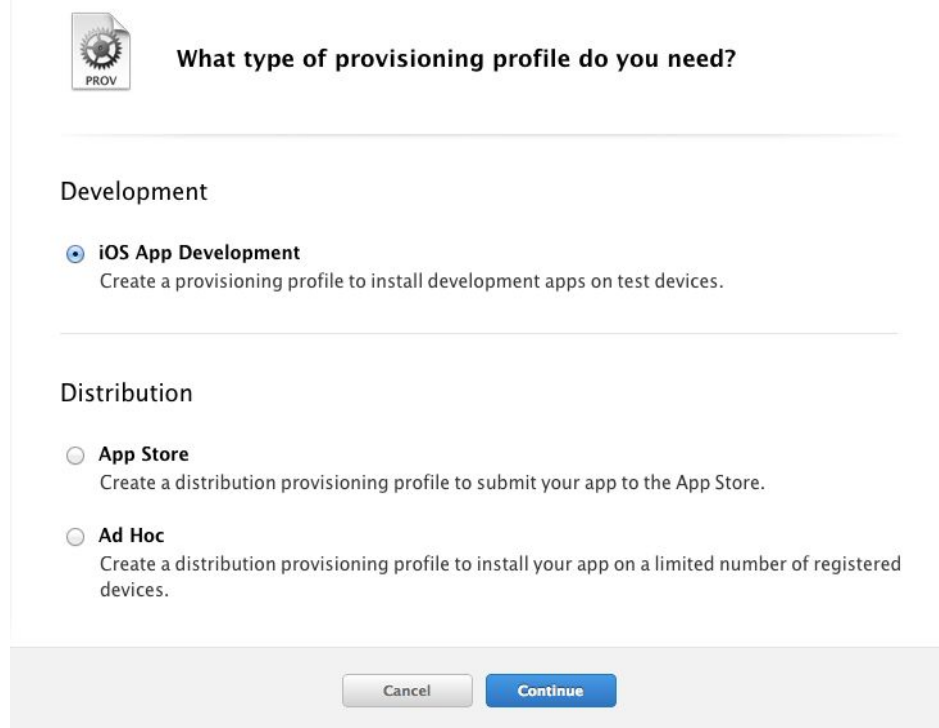
## Making the Provisioning Profile

You're not yet done with the [iOS Dev Center](#). Click the **Provisioning Profiles** button in the sidebar and click the **+** button.

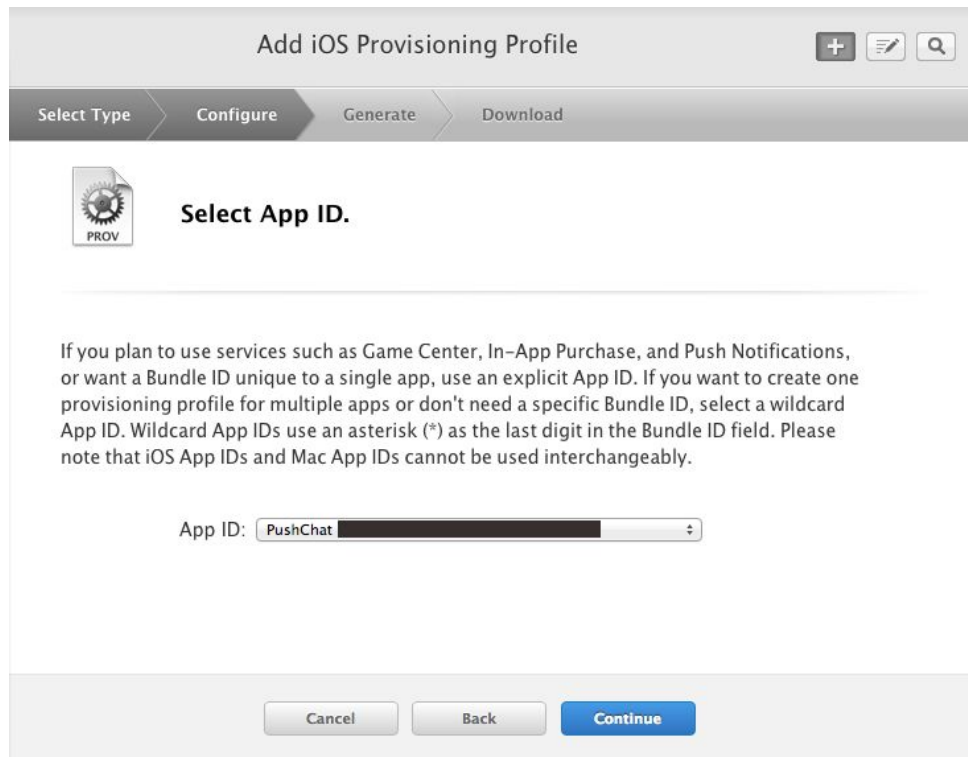


This will open up the iOS provisioning profile wizard.

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



2. 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. Press **Continue**.



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



4. 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

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



### Name this profile and generate.

The name you provide will be used to identify the profile in the portal. You cannot use special characters such as @, &, \*, ', " for your profile name.

Profile Name:

Type: **Development**

App ID: **PushChat**

Certificates: **1 Included**

Devices: **2 Included**

Cancel

Back

Generate

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.

## Apply Credentials and Test

What is left is to configure the **Demo Server**. You can use any Java IDE you like. F.e. in **Eclipse** you can create a new Java project in the

Assets/UTNotifications/Editor/DemoServer folder. All the source files and libraries will be imported into it by default.

1. Open the file

```
Assets/UTNotifications/Editor/DemoServer/src/DemoServer/PushNotification.java
```

2. Find these lines in it:

```
private static final String APN_CERT_PATH = null;
private static final String APN_CERT_PASSWORD = null;
```

3. Replace these nulls by the full path and password of .p12 you created in [Converting the aps\\_development.cerFile](#). The path may also be relative to the

Assets/UTNotifications/Editor/DemoServer folder. For example:

```
private static final String APN_CERT_PATH = "apn_cert.p12";
private static final String APN_CERT_PASSWORD = "test321";
```

4. Build and run the **Demo Server** (Fn + F5 in **Eclipse** in OS X by default).

5. Now let's save the running server address in the m\_webServerAddress variable in file

Assets/UTNotifications/Sample/UTNotificationsSample.cs like:

```
protected string m_webServerAddress = "http://address:port";
```

F.e. I connected an iPad to the same Wi-Fi network as the **Demo Server** so I used the internal network address of the server here:

```
protected string m_webServerAddress = "http://192.168.2.102:8080";
```

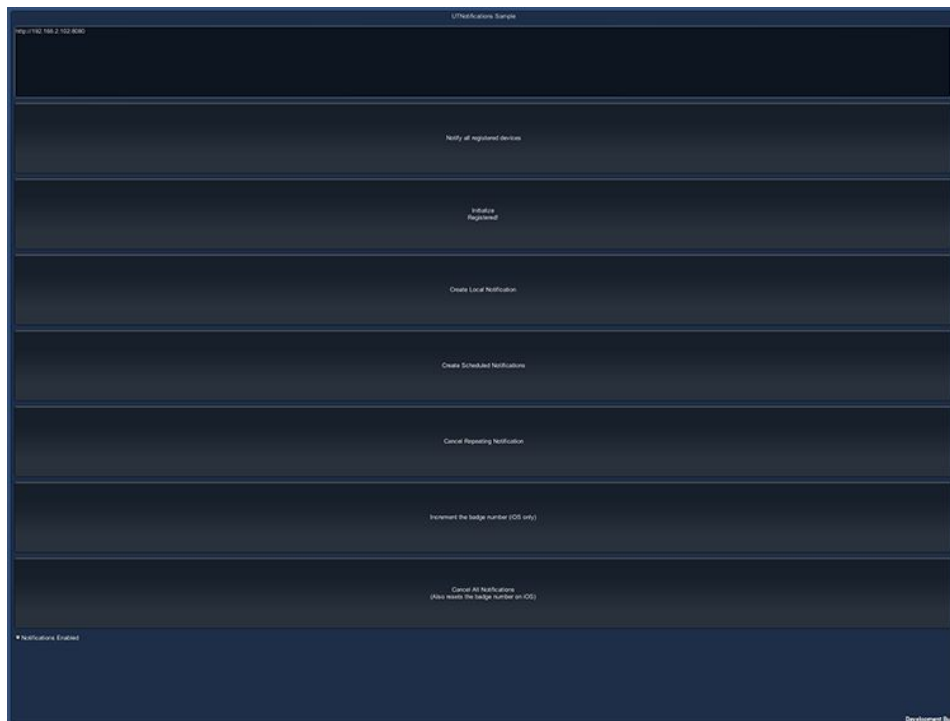
6. In Unity open the UTNotifications Settings in menu: Edit -> Project Settings -> UTNotifications (Unity restart may be required to see this menu item first time) and enable **Push Notifications** toggle in the **iOS Settings**.

7. Setup the UTNotificationsExampleScene

(Assets/UTNotifications/Sample/PushNotificationExampleScene.unity) as the first scene in build in Unity: File -> Build Settings -> Scenes In Build

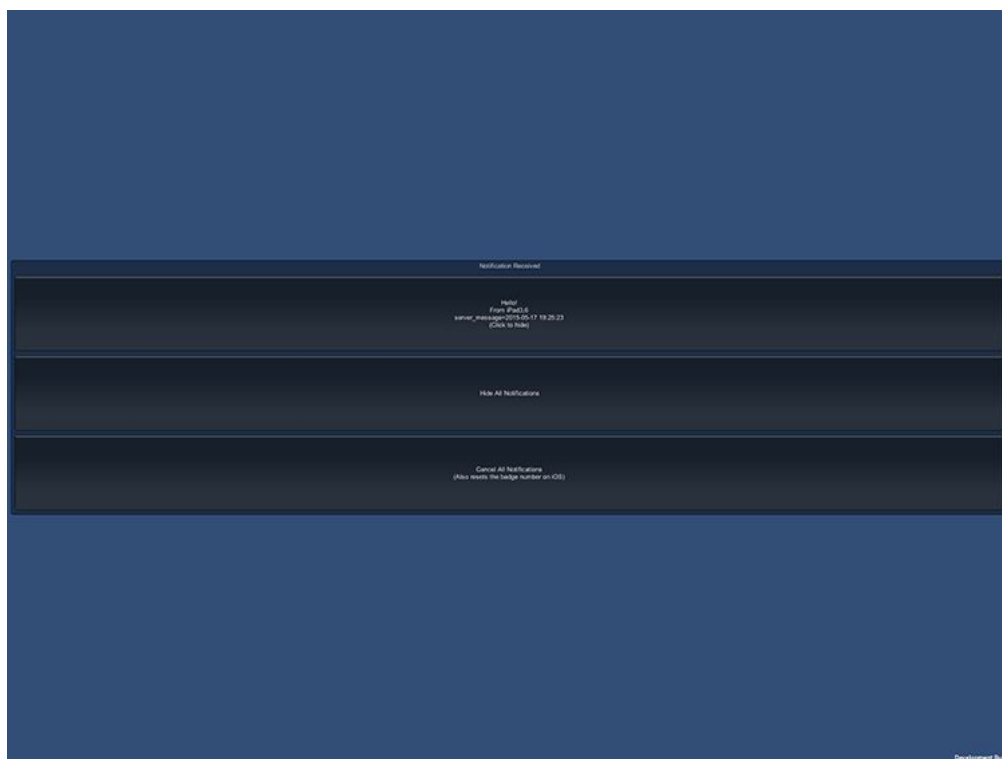
8. Build and deploy the iOS version to a device. Please make sure, that **XCode** uses the same **Code Signing Identity** as was selected on 3rd step of [Making the provisioning profile](#).

9. If you did everything right you should see this:



2nd button text: "Initialize\nRegistered!" means that the server is running, accessible and the **registration id** was successfully received and sent to the **Demo Server**.

10. Press **Notify all registered devices** button to request the **Demo Server** to send a push notification to every registered in it **registration id**. When it's delivered you'll see a screen similar to this:



It means that registering, receiving push notifications and their handling works fine!



## Configuring the Google Cloud Messaging (GCM)

Based on GCM official documentation: <https://developers.google.com/cloud-messaging/>.

### Enable Google Cloud Messaging

1. Open this page:  
<https://developers.google.com/cloud-messaging/android/client#get-config>
2. Press **GET A CONFIGURATION FILE** button.
3. Choose or create an app (use the same Android package name as specified for your project in Unity as **Bundle id**). Press **Choose and configure service** button.

Create or choose an app

App name

Push Notifications ▾

Services will be added to your existing project in the [Google Developers Console](#).

Android package name

universal.tools.notificationsexample ▾


CONTINUE TO  
Choose and configure services →

4. Choose **Cloud Messaging** tab if not selected. Press **ENABLE GOOGLE CLOUD MESSAGING** button.


## Choose and configure services

✓ You are configuring the **Push Test** app with package name **universal.tools.notificationsexample**. ✕


Select which Google services you'd like to add to your app below.



Google Sign-In



Analytics



Cloud Messaging

### Cloud Messaging

Google Cloud Messaging lets you send messages between your server and your users' devices

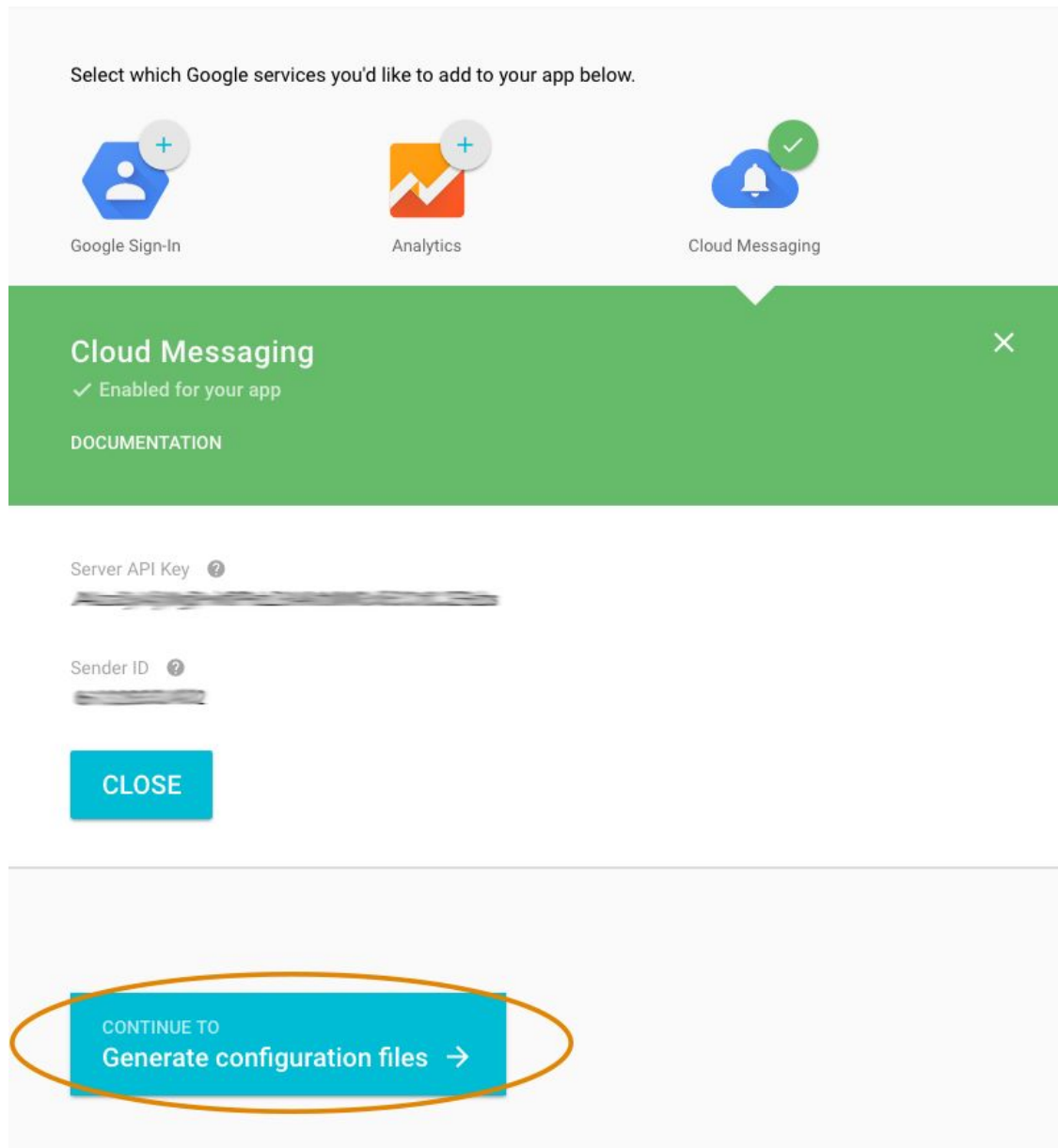
[LEARN MORE](#)

**ENABLE GOOGLE CLOUD MESSAGING**

CONTINUE TO

**Generate configuration files** →

5. Press **Generate configuration files** button.



6. Press **Download google-services.json** button to download the configuration file.

### Apply Credentials and Test

What is left is to apply the configuration file you obtained previously and start the **Demo Server**. You can use any Java IDE you like. F.e. in **Eclipse** you can create a new Java project in the `Assets/UTNotifications/Editor/DemoServer` folder. All the source files and libraries will be imported into it by default.

1. In Unity open the UTNotifications Settings in menu: `Edit -> Project Settings -> UTNotifications` (Unity restart may be required to see this menu item first time) and enable **Push Notifications** toggle in the **Google Cloud Messaging**.

2. In **Google Play Settings** press **Load google-services.json** button to load and apply the configuration file you obtained in 6th step of [Enable Google Cloud Messaging](#). You should see a numeric value of **SenderID** below the button.



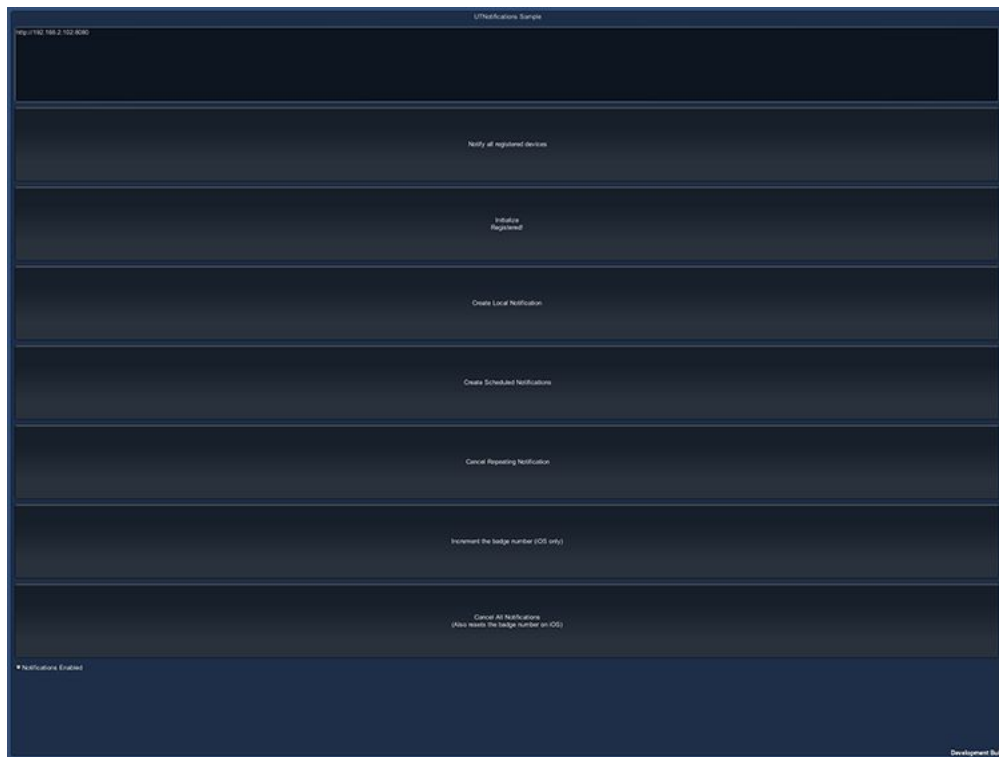
3. Build and run the **Demo Server** (Fn + F5 in **Eclipse** in OS X by default). The DemoServer should have been automatically configured in a previous step.
4. Now let's save the running server address in the `m_webServerAddress` variable in file `Assets/UTNotifications/Sample/UTNotificationsSample.cs` like:  

```
protected string m_webServerAddress = "http://address:port";
```

 F.e. I connected an Android device to the same Wi-Fi network as the **Demo Server** so I used the internal network address of the server here:  

```
protected string m_webServerAddress = "http://192.168.2.102:8080";
```
5. Setup the `UTNotificationsExampleScene` (`Assets/UTNotifications/Sample/PushNotificationExampleScene.unity`) as the first scene in build in Unity: File -> Build Settings -> Scenes In Build
6. Build and deploy the Android version to a device.

7. If you did everything right you should see this:



2nd button text: "Initialize\nRegistered!" means that the server is running, accessible and the **registration id** was successfully received and sent to the **Demo Server**.

8. Press **Notify all registered devices** button to request the **Demo Server** to send a push notification to every registered in it **registration id**. When it's delivered you'll see a screen similar to this:



It means that registering, receiving push notifications and their handling works fine!

## Configuring the Amazon Device Messaging (ADM)

Based on ADM official documentation:

<https://developer.amazon.com/public/apis/engage/device-messaging/tech-docs/02-obtaining-adm-credentials>

### Getting Your OAuth Credentials and API Key

To obtain credentials and enable your app to use ADM:

1. Create an account on the [Amazon Apps & Games Developer Portal](https://developer.amazon.com/public/apis/engage/device-messaging/tech-docs/02-obtaining-adm-credentials) and add your app, if you have not already done so.
2. In **Apps & Services > My Apps**, select the app with which you want to use ADM or create a new one.
3. Click **Device Messaging**.
4. If you have already assigned a security profile to your app, proceed to step 7.
5. To assign a security profile to your app, choose an existing security profile from **Select a Security Profile** or click **Create a New Security Profile**. A security profile provides the OAuth credentials that you use when sending messages with ADM.  
**Note:** You can share the use of a security profile among more than one app. Sharing a profile allows apps to share some types of data. For example, you may have a "My Cat - Free" app and a "My Cat - HD" app. If you apply a single security profile to both apps, data accessed by that profile is available to both apps. For a shared profile, choose a name that applies to both, for example, "My Cat Apps profile".

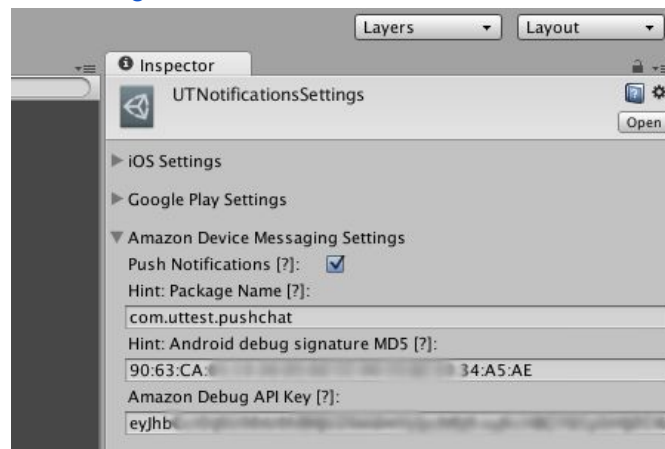
6. If you used an existing security profile, be sure to select **Confirm** to save your changes.
7. Click **View Security Profile**.

8. Store somewhere the **Client ID** and **Client Secret** values.

9. Then click **Android/Kindle Settings**.
10. Create an **API Key**. Your app requires one or more API Keys.
  - **(Required)** For a pre-release or "debug" version of your app. In all cases, you must create an API Key for the debug version of your app, in order to test your app with ADM.
  - **(Optional)** For a release or "production" version of your app. If you sign the release version of your app using your own certificate, you must create an

additional API Key for the release version of your app. If you allow Amazon to sign your app on your behalf, you do not need to create an additional API Key. To create an API Key, you must provide both the package name (for example, `com.mycompany.bestapplication`) for the app and its signature:

- **Debug** application signature for the pre-release version of your app.
  - a. In Unity open the UTNotifications Settings in menu: `Edit -> Project Settings -> UTNotifications` (Unity restart may be required to see this menu item first time) and enable **Push Notifications** toggle in the **Amazon Device Messaging**.
  - b. Copy and paste the **Package Name** and **Android debug signature MD5** hints from **UTNotifications Settings / Amazon Device Messaging Settings** to the Amazon **Security Profile** fields **Package** and **Signature**.  
**Note:** If you don't see the **Android debug signature MD5** hint value please build the Android version at least once successfully. If getting the **Android debug signature MD5** is still failed after that, please see <https://developer.amazon.com/public/apis/engage/device-messaging/tech-docs/02-obtaining-adm-credentials>.



## Security Profile Management

[More Information](#)  
[Login with Amazon](#)  
[GameCircle](#)  
[Device Messaging](#)

**PushChatSecurityProfile - Security Profile**

General **Android/Kindle Settings** iOS Settings

An API Key allows Amazon to verify your app's identity. An API Key is generated based on the values you provide below. If different versions of your app have different signatures or package names, such as for one or more testing versions and a production version, each version requires its own API Key. [Learn More](#)

API Key Name \*   
Identifies the app you will use with this API key.

Package \*   
The package name of your Android project. For example, com.mycompany.bestapp.

Signature \*   
The MD5 signature of the certificate used to sign your app. [Get instructions on obtaining this value](#)

[Generate New Key](#)

- **Release** application signature for the production version of your app. If you sign the release version of your app using your own certificate, provide the MD5 signature for that certificate to create an additional API Key (more details at <https://developer.amazon.com/public/apis/engage/device-messaging/tech-docs/02-obtaining-adm-credentials>). If you allow Amazon to sign your app on your behalf, you do not need to obtain an API Key for the release signature.



11. Click **Generate New Key**.
  12. Store the retrieved **API Key** somewhere.
- Note:** It shouldn't contain any spaces or newline characters.


#### Security Profile Management

[More Information](#)  
[Login with Amazon](#)  
[GameCircle](#)  
[Device Messaging](#)

**PushChatSecurityProfile - Security Profile**

General **Android/Kindle Settings** iOS Settings

An API Key allows Amazon to verify your app's identity. An API Key is generated based on the values you provide below. If different versions of your app have different signatures or package names, such as for one or more testing versions and a production version, each version requires its own API Key. [Learn More](#)

API Key Name	PushChatDebugAPIKey
Key	
Package	com.uttest.pushchat
Signature	90:s3:CA...:34:A5:AE

[Edit](#)

[Add an API Key](#)

### Apply Credentials and Test

What is left is to configure the **Demo Server**. You can use any Java IDE you like. F.e. in **Eclipse** you can create a new Java project in the `Assets/UTNotifications/Editor/DemoServer` folder. All the source files and libraries will be imported into it by default.

1. Open the file  
`Assets/UTNotifications/Editor/DemoServer/src/DemoServer/PushNotificationr.java`
2. Find these lines in it:  

```
private static final String AMAZON_CLIENT_ID = null;
private static final String AMAZON_CLIENT_SECRET = null;
```
3. Replace the `null`'s by the values you got in 8th step of [Getting Your OAuth Credentials and API Key](#).
4. Build and run the **Demo Server** (Fn + F5 in **Eclipse** in OS X by default).
5. In Unity open the UTNotifications Settings in menu: Edit -> Project Settings -> UTNotifications
6. In **Amazon Device Messaging Settings** write down the **Amazon Debug API Key** value you got in 12th step of [Getting Your OAuth Credentials and API Key](#).

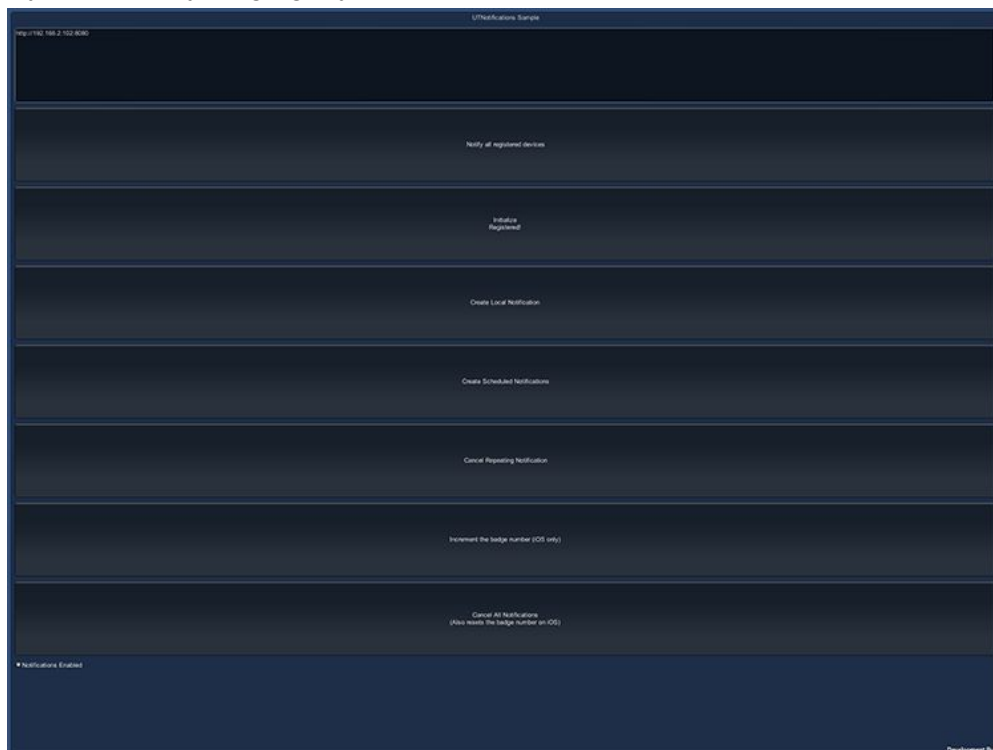


7. Now let's save the running server address in the `m_webServerAddress` variable in file `Assets/UTNotifications/Sample/UTNotificationsSample.cs`, like:  

```
protected string m_webServerAddress = "http://address:port";
```

 F.e. I connected a **Kindle** device to the same Wi-Fi network as the **Demo Server** so I used the internal network address of the server here:  

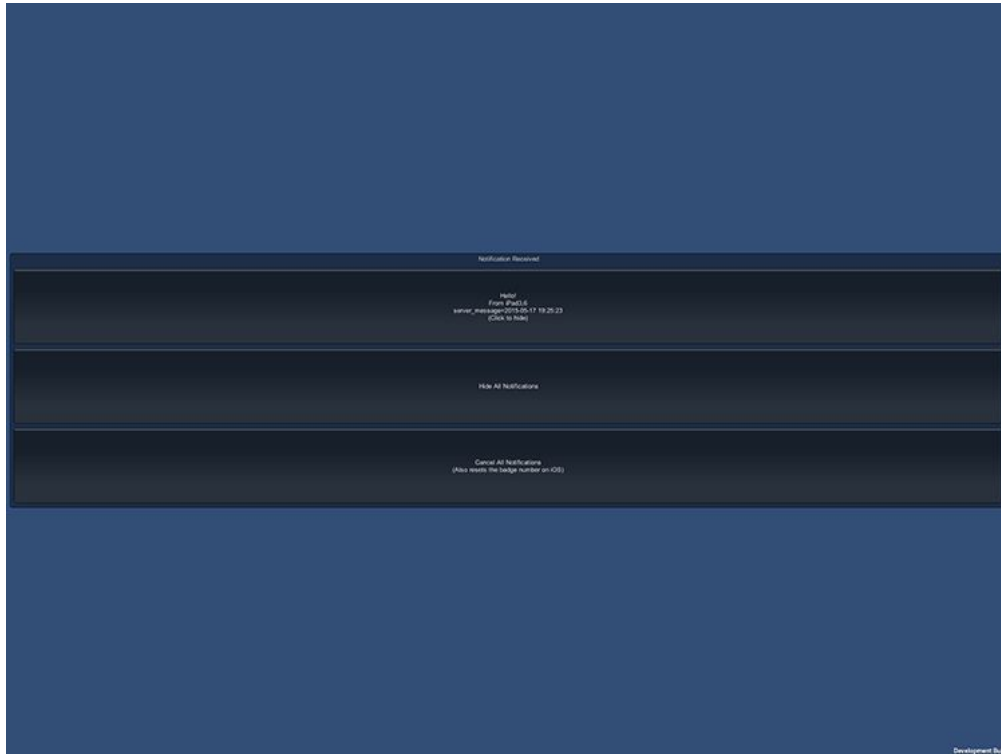
```
protected string m_webServerAddress = "http://192.168.2.102:8080";
```
8. Setup the `UTNotificationsExampleScene` (`Assets/UTNotifications/Sample/PushNotificationsExampleScene.unity`) as the first scene in build in Unity: `File -> Build Settings -> Scenes In Build`
9. Build and deploy the Android version to a device.
10. If you did everything right you should see this:



2nd button text: "Initialize\nRegistered!" means that the server is running,

accessible and the **registration id** was successfully received and sent to the **Demo Server**.

11. Press **Notify all registered devices** button to request the **Demo Server** to send a push notification to every registered in it **registration id**. When it's delivered you'll see a screen similar to this:



It means that registering, receiving push notifications and their handling works fine!

## Configuring the Windows Push Notification Services (WNS)

Based on WNS official documentation:

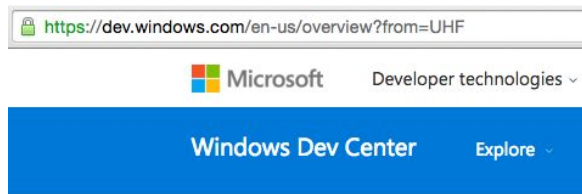
<https://msdn.microsoft.com/en-us/library/windows/apps/hh465407.aspx>.

### Register your app with the Dashboard

Before you can send notifications through WNS, you must register your app. Do so through the [Dashboard](#), the developer portal that enables you to submit, certify, and manage your Windows Store apps. When you register your app through the Dashboard, you are given credentials—a Package security identifier (SID) and a secret key—which your cloud service uses to authenticate itself with WNS.

To register:

1. Go to the [Windows Store apps page](#) of the Windows Dev Center and sign in with your Microsoft account.
2. Once you have signed in, click the [Dashboard](#) link.
3. On the Dashboard, select Submit an app.



## My apps

UTNotifications Test  
In progress

+ Create a new app

Payout summary

Advertising performance

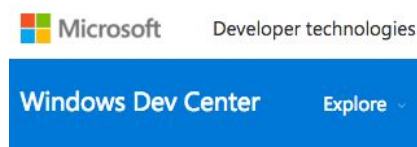
Account settings

4. Choose a name and click “Reserve app name” to register an app.

### Obtain the identity values for your app

When you reserved a name for your app, the Windows Store created your associated credentials. It also assigned associated identity values—name and publisher.

1. Click at **Services** -> **Push Notifications** in the left menu.



UTNotifications Sample

App overview

Analytics ▾

Submissions

IAPs

Monetization ▾

Services ▴

Push notifications

Maps

App management ▾

## 2. Press on a link **Live Services** site.

Microsoft Developer technologies

Windows Dev Center Explore Docs Downloads Samples Community Programs

UTNotifications Sample

# Push notifications

Windows Push Notification Services (WNS) and Microsoft Azure Mobile Services

The Windows Push Notification Services (WNS) enables you to send toast, tile, badge, and raw updates from your own cloud service. [Learn more](#)

If you have an existing WNS solution or need to update your current client secret, visit the [Live Services site](#)

You can also use [Microsoft Azure Mobile Services](#) to send push notifications, authenticate and manage app users, and store app data in the cloud. [Sign in](#) to your Microsoft Azure account or [sign up](#) now to add services to up to ten apps for free.

App overview  
Analytics  
Submissions  
IAPs  
Monetization  
Services  
**Push notifications**  
Maps  
App management

## 3. Save somewhere the following values: **Package SID**, **Client secret**, **Identity Name** & **Publisher**.

### UTNotifications Sample

Settings

Basic Information  
API Settings  
**App Settings**  
Localization

To protect your app's security, Windows Push Notification Services (WNS) and services using Microsoft account use client secrets to authenticate the communications from your server.

Package SID:  
ms-app://  
This is the unique identifier for your Windows Store app.

Link to different app

Application Identity:  
<Identity  
Name="  
Publisher="CN=  
>  
To set your application's identity values manually, open the AppManifest.xml file in a text editor and set these attributes of the <identity> element using the values shown here.

Client ID:  
000000004017474F  
This is a unique identifier for your application.

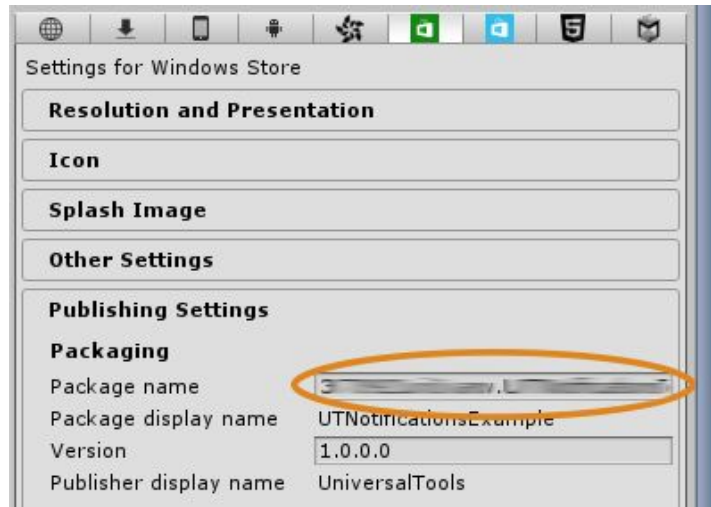
Client secret:  
For security purposes, don't share your client secret with anyone.

If your client secret has been compromised or your organization requires that you periodically change client secrets, create a new client secret here. After you create a new client secret, both the old and the new client secrets will be accepted until you activate the new secret.

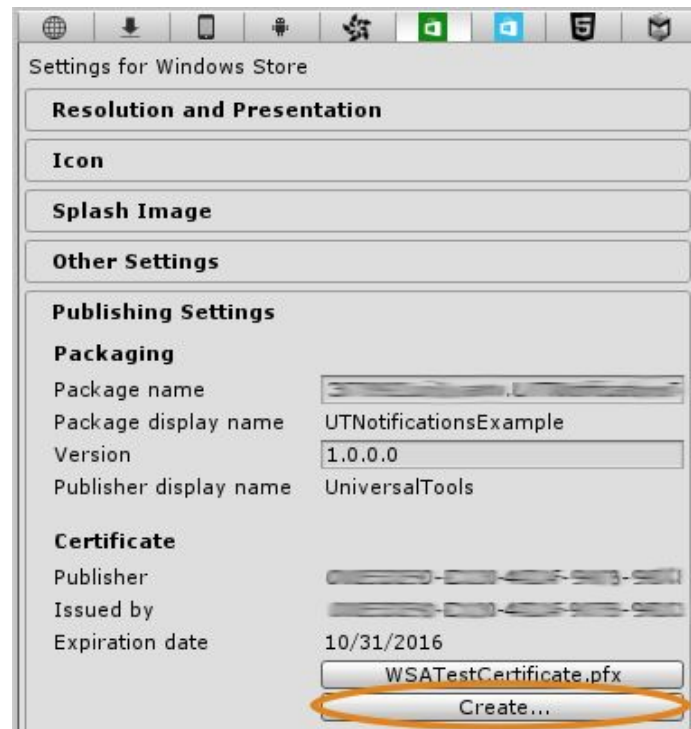
[Create a new client secret](#)

**Note:** Please wait 24 hours before you activate your new client secret, because the old client secret won't work after you activate the new one.

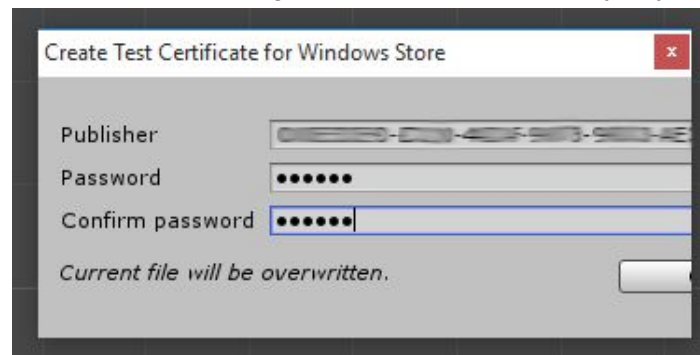
- In Unity open the UTNotifications Settings in menu: Edit -> Project Settings -> UTNotifications (Unity restart may be required to see this menu item first time) and enable **Push Notifications** toggle in the **Windows Store Settings**.
- Open Windows Store player settings: File -> Build Settings... -> Windows Store -> Player Settings.
- Use **Identity Name** value from 3rd step as **Package Name**.



7. Press Create button to create a certificate.



8. Use **Publisher** from 3rd step for **Publisher**. Don't include starting **CN=** to this value, only the rest. Note, that at least in Unity 5.2 the certificate creation dialog is buggy (it's not optimized for such a long values of Publisher). Anyway, it works.



## Apply Credentials and Test

What is left is to configure the **Demo Server**. You can use any Java IDE you like. F.e. in **Eclipse** you can create a new Java project in the

Assets/UTNotifications/Editor/DemoServer folder. All the source files and libraries will be imported into it by default.

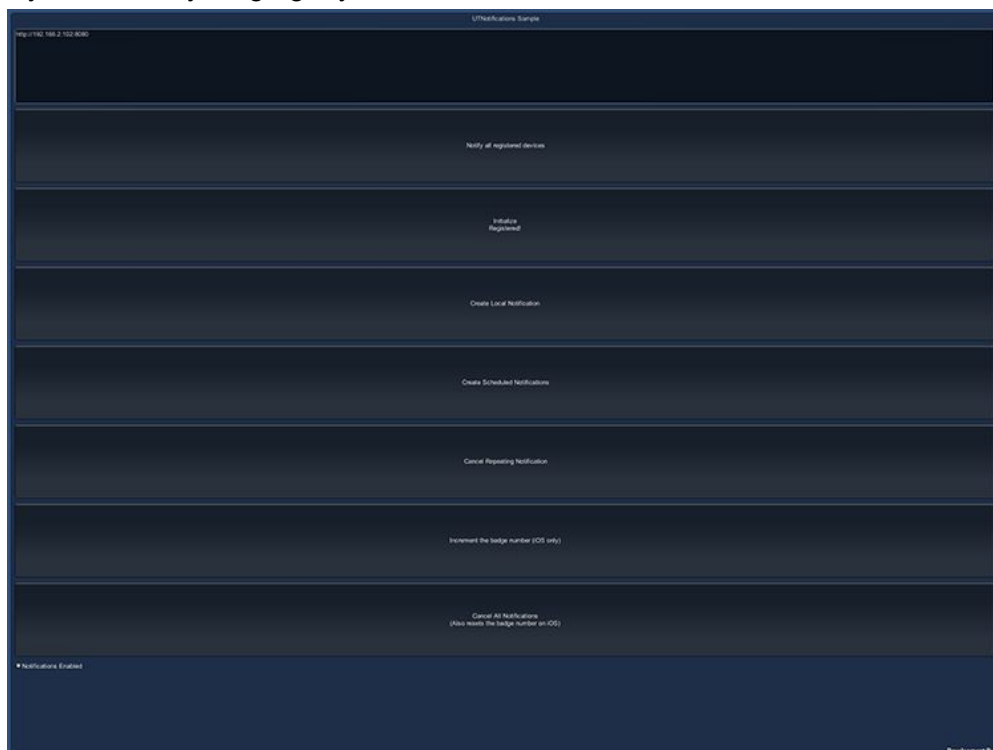
1. Open the file  
Assets/UTNotifications/Editor/DemoServer/src/DemoServer/PushNotification.r.java
2. Find these lines in it:  

```
private static final String WINDOWS_PACKAGE_SID = null;  
private static final String WINDOWS_CLIENT_SECRET = null;
```
3. Replace the `null`'s by the values **Package SID** & **Client secret** you got in **3rd** step of [Obtain the identity values for your app](#) section.
4. Build and run the **Demo Server** (Fn + F5 in **Eclipse** in OS X by default).
5. Now let's save the running server address in the `m_webServerAddress` variable in file Assets/UTNotifications/Sample/UTNotificationsSample.cs like:  

```
protected string m_webServerAddress = "http://address:port";
```

F.e. I connected a **Kindle** device to the same Wi-Fi network as the **Demo Server** so I used the internal network address of the server here:  

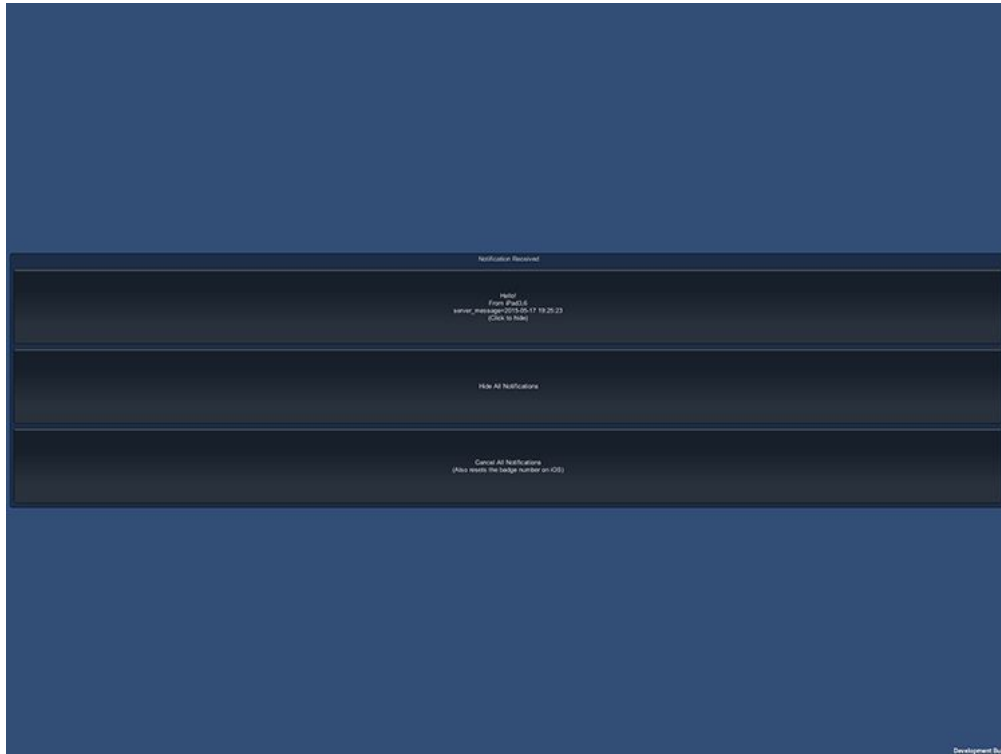
```
protected string m_webServerAddress = "http://192.168.2.102:8080";
```
6. Setup the UTNotificationsExampleScene  
(Assets/UTNotifications/Sample/UTNotificationsExampleScene.unity) as the first scene in build in Unity: File -> Build Settings -> Scenes In Build
7. Build and deploy the Windows Store version to a phone or a local computer.
8. If you did everything right you should see this:



2nd button text: "Initialize\nRegistered!" means that the server is running,

accessible and the **registration id** was successfully received and sent to the **Demo Server**.

9. Press **Notify all registered devices** button to request the **Demo Server** to send a push notification to every registered in it **registration id**. When it's delivered you'll see a screen similar to this:



It means that registering, receiving push notifications and their handling works fine!

## Unicode Support

Please note that in order to support non-English Unicode characters on Android and Windows Store (GCM, ADM & WNS), the **title** and **text** and user data strings of sent push notifications should be [URL-encoded](#). F.e. see

Assets/UTNotifications/Editor/DemoServer/src/DemoServer/PushNotificator.java,  
public static int **notifyGooglePlay** OR public static int **notifyAmazon**:

```
title = java.net.URLEncoder.encode(title);  
text = java.net.URLEncoder.encode(text);
```

No encoding is required for iOS.

## Contacts

If you got any questions please feel free to contact us: [universal.tools.contact@gmail.com](mailto:universal.tools.contact@gmail.com).

You can post bugs and feature requests at

<https://github.com/universal-tools/UTNotificationsFeedback/issues>.

If you liked using UTNotifications, please [rate it](#), but any criticism is also welcome - please help us make the asset better!

Thank you for using UTNotifications!



Your Universal Tools team.