

Notification Hub for Cross Platform mobile apps

Abdul Rasheed Feroz Khan

Technical Evangelist

fb.com/abdulrasheed92

© 2015 Microsoft Corporation. All rights reserved.

Table of Contents

[Introduction 3](#_Toc414295682)

[Let’s get Started - Prerequisites 3](#_Toc414295683)

[Enable Google Cloud Messaging 4](#_Toc414295684)

[Configure your Notification Hub 7](#_Toc414295685)

[Connecting your app to the Notification Hub 12](#_Toc414295686)

[Create a new project 12](#_Toc414295687)

[Add the required Components to your project 15](#_Toc414295688)

[Setup Notification Hubs in your project 16](#_Toc414295689)

[Run your app in the emulator 20](#_Toc414295690)

[Send notification from your back-end 23](#_Toc414295691)

[Recommended/related sessions 28](#_Toc414295692)

[Tell us what you think 28](#_Toc414295693)

[References 29](#_Toc414295694)

[Azure Support 29](#_Toc414295695)

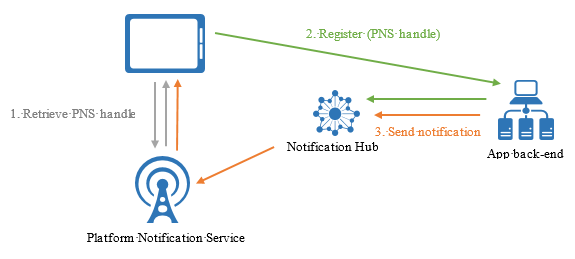
# Introduction

Azure Notification Hubs provide an easy-to-use infrastructure that enables you to send mobile push notifications from any backend (in the cloud or on-premises) to any mobile platform.

With Notification Hubs you can easily send cross-platform, personalized push notifications, abstracting the details of the different platform notification systems (PNSs). With a single API call, you can target individual users or entire audience segments containing millions of users, across all their devices.

You can use Notification Hubs for both enterprise and consumer scenarios. For example:

* Send breaking news notifications to millions with low latency (Notification Hubs powers Bing applications pre-installed on all Windows and Windows Phone devices).
* Send location-based coupons to user segments.
* Send event notifications to users or groups for sports/finance/games applications.
* Notify users of enterprise events like new messages/emails, and sales leads.
* Send one-time-passwords required for multi-factor authentication.



# Let’s get Started - Prerequisites

In this tutorial, you create a blank Xamarin.Android app that receives push notifications using Google Cloud Messaging (GCM). When complete, you will be able to broadcast push notifications to all the devices running your app using your notification hub.

The tutorial walks you through these basic steps to enable push notifications:

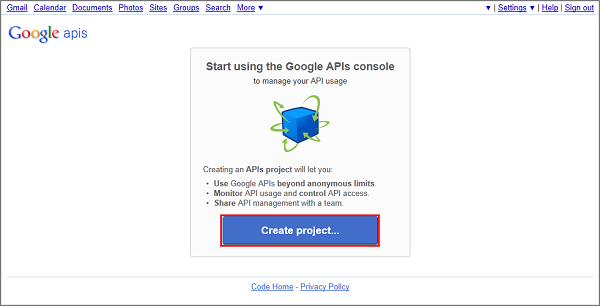
1. Enable Google Cloud Messaging
2. Configure your Notification Hub
3. Connecting your app to the Notification Hub
4. Run your app with the emulator
5. Send notifications from your back-end

This tutorial demonstrates the simple broadcast scenario using Notification Hubs. This tutorial requires the following:

* [Xamarin.Android](http://xamarin.com/download/)
* Active Google account
* [Azure Mobile Services Component](http://components.xamarin.com/view/azure-mobile-services/) – Xamarin Component Store
* [Azure Messaging Component] - Xamarin Component Store
* [Google Cloud Messaging Client Component] - Xamarin Component Store

# Enable Google Cloud Messaging

1. Navigate to the [Google apis](http://go.microsoft.com/fwlink/p/?LinkId=268303) website, sign-in with your Google account credentials, and then click **Create project...**.



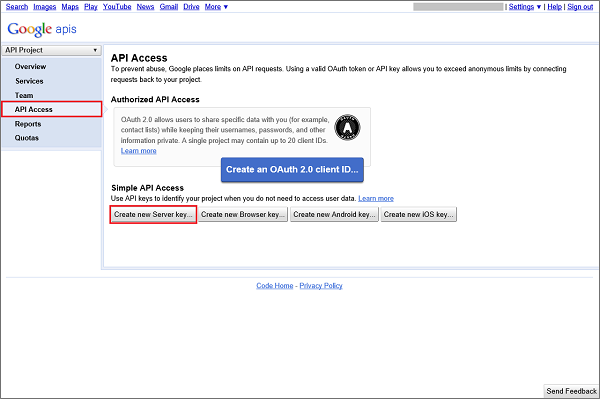
NOTE:

When you already have an existing project, you are directed to the **Dashboard** page after login. To create a new project from the Dashboard, expand **API Project**, click **Create...** under **Other projects**, then enter a project name and click **Create project**.

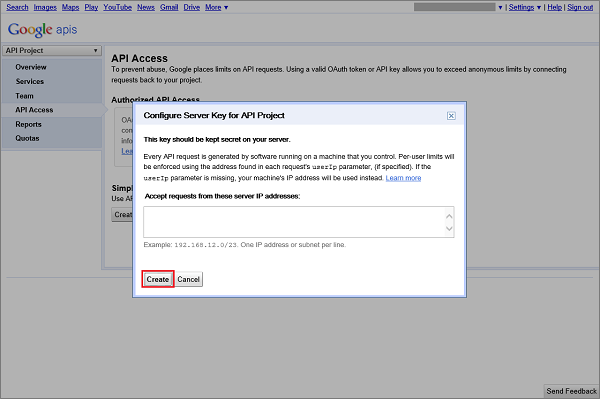
1. Click **Overview** in the left column, and make a note of the project number in the **Dashboard** section.

Later in the tutorial you set this value as the PROJECT\_ID variable in the client.

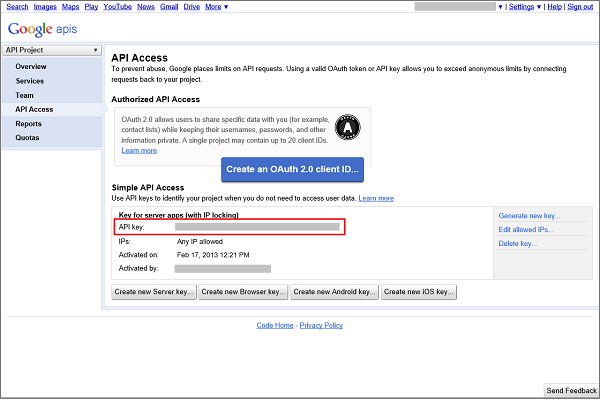
1. On the [Google apis](http://go.microsoft.com/fwlink/p/?LinkId=268303) page, click **Services**, then click the toggle to enable **Google Cloud Messaging for Android** and accept the terms of service.
2. Click **API Access**, and then click **Create new Server key...**



1. In **Configure Server Key for API Project**, click **Create**.



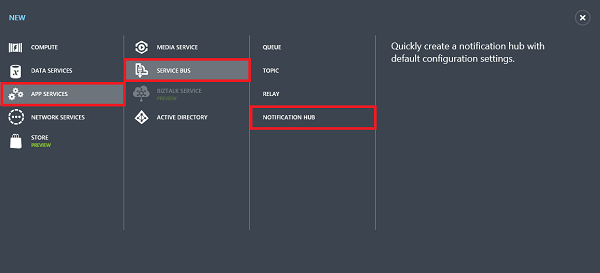
1. Make a note of the **API key** value.



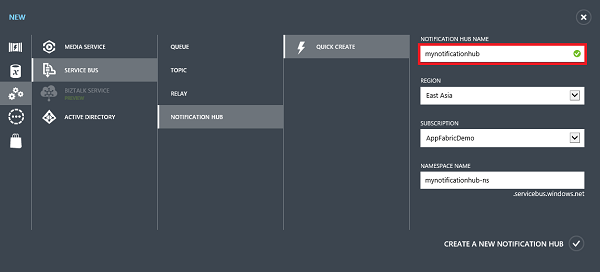
Next, you will use this API key value to enable your notification hub to authenticate with GCM and send push notifications on behalf of your application.

# Configure your Notification Hub

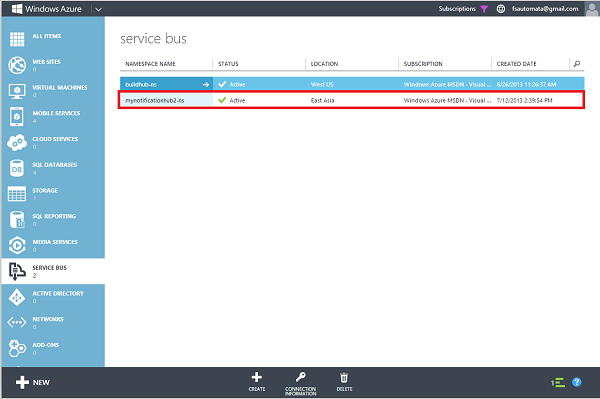
1. Log on to the [Azure Management Portal](https://manage.windowsazure.com/), and then click **+NEW** at the bottom of the screen.
2. Click on **App Services**, then **Service Bus**, then **Notification Hub**, then **Quick Create**.



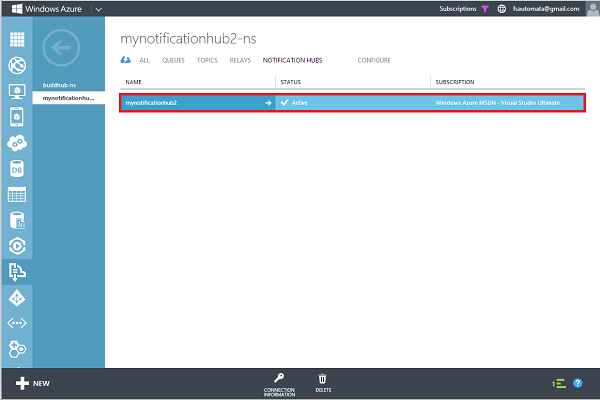
1. Type a name for your notification hub, select your desired region, and then click **Create a new Notification Hub**.



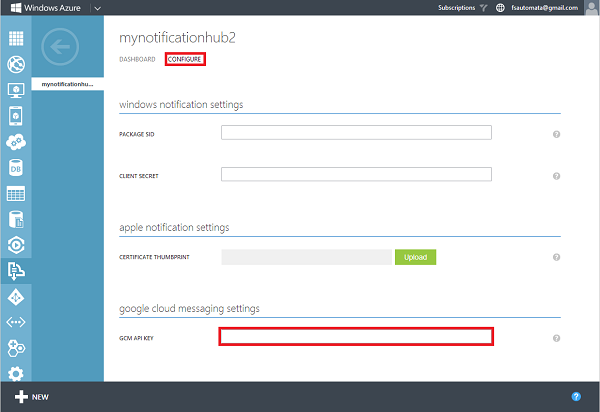
1. Click the namespace you just created (usually ***notification hub name*-ns**), and then click **Configure** at the top.



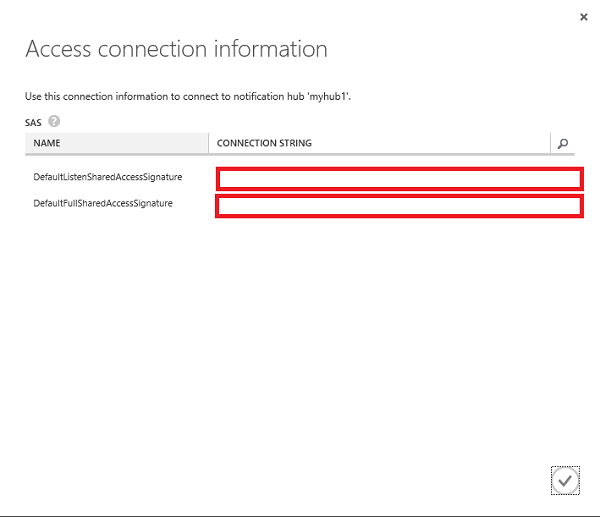
1. Click the **Notification Hubs** tab at the top, and then click on the notification hub you just created.



1. Click the **Configure** tab at the top, enter the **API Key** value you obtained in the previous section, and then click **Save**.



1. Select the **Dashboard** tab at the top, then click **Connection Information**. Take note of the two connection strings.

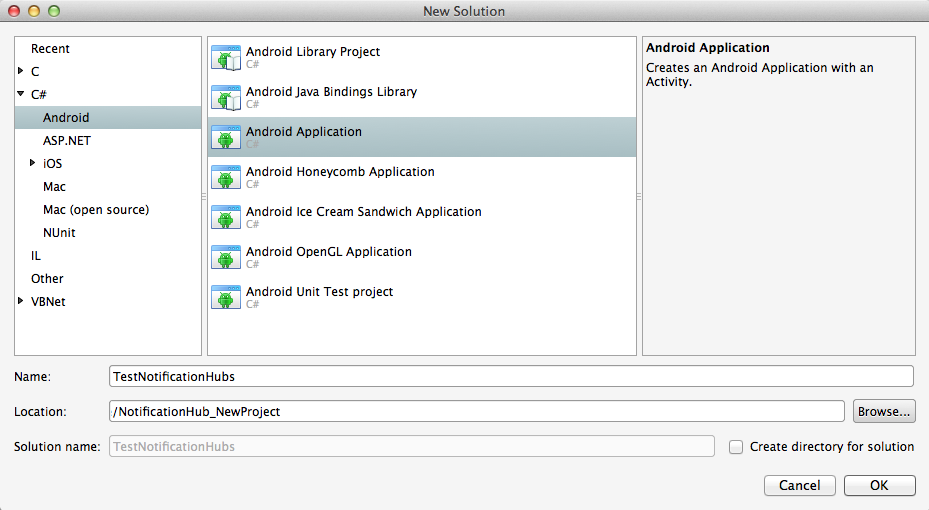
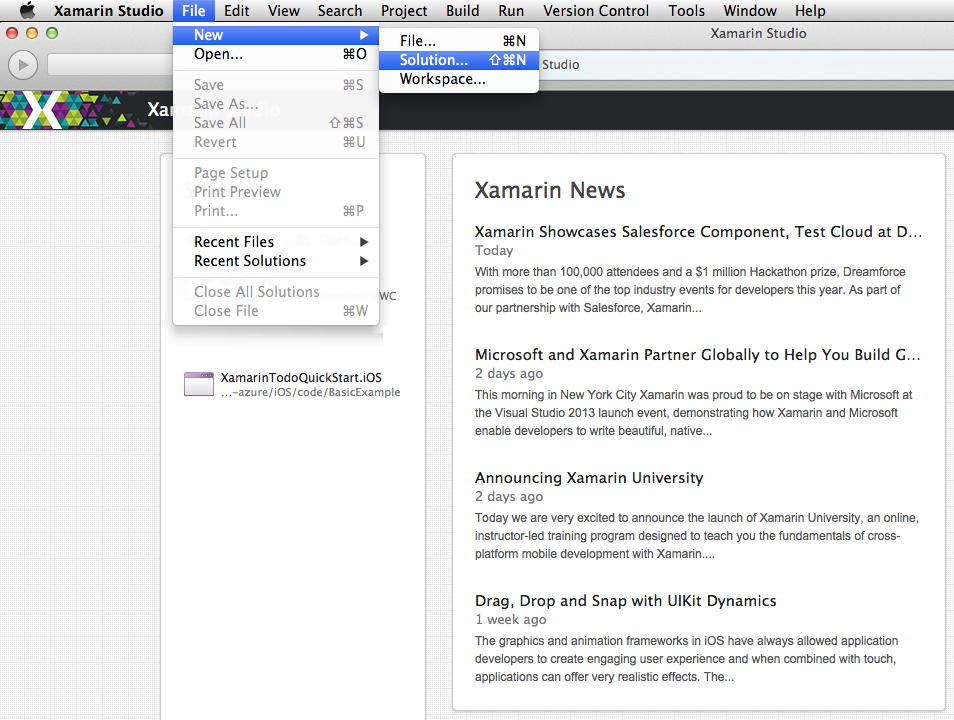


Your notification hub is now configured to work with GCM, and you have the connection strings to register your app and send push notifications.

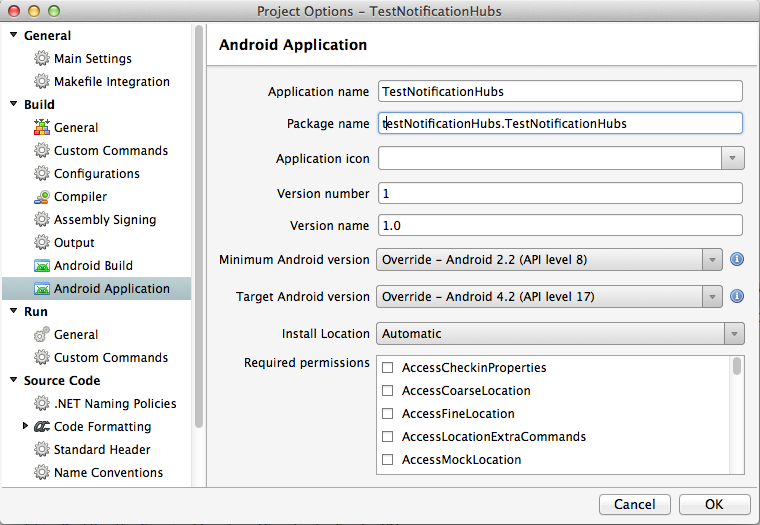
# Connecting your app to the Notification Hub

## Create a new project

1. In Xamarin Studio (or Visual Studio), create a new Android project (File, New, Solution, Android Application).



1. Open the project properties by right clicking on your new project in the Solution view, and choosing **Options**. Select the **Android Application** item in the **Build** section.



1. Set the **Minimum Android version** to API Level 8.
2. Set the **Target Android version** to the API version you would like to target (must be API level 8 or higher).
3. Ensure that the first letter of your **Package name** is lowercase.

IMPORTANT:

The first letter of the package name must be lowercase. Otherwise you will receive application manifest errors when registering your**BroadcastReceiver** and **IntentFilter**s for push notifications below.

## Add the required Components to your project

The Google Cloud Messaging Client available on the Xamarin Component Store simplifies the process of supporting push notifications in Xamarin.Android.

1. Right-click the Components folder in Xamarin.Android app and choose **Get More Components...**
2. Search for the **Azure Mobile Services** component and add it to the project.
3. Search for the **Azure Messaging** component and add it to the project.
4. Search for the **Google Cloud Messaging Client** component and add it to the project.

## Setup Notification Hubs in your project

1. Create a **Constants.cs** class and define the following constant values (replacing placeholders with values):

public const string SenderID = "<GoogleProjectNumber>"; // Google API Project Number

// Azure app specific connection string and hub path

public const string ConnectionString = "<Azure connection string>";

public const string NotificationHubPath = "<hub path>";

1. Add the following using statements to **MainActivity.cs**:

using Microsoft.WindowsAzure.MobileServices;

using Gcm.Client;

1. Create the following method in the **MainActivity** class:

private void RegisterWithGCM()

{

// Check to ensure everything's setup right

GcmClient.CheckDevice(this);

GcmClient.CheckManifest(this);

// Register for push notifications

System.Diagnostics.Debug.WriteLine("Registering...");

GcmClient.Register(this, Constants.SenderID);  
}

1. Create a new class **MyBroadcastReceiver**.

NOTE:

We will walk through creating a **BroadcastReceiver** from scratch below. However, a quick alternative to manually creating a**MyBroadcastReceiver.cs** is to refer to the **GcmService.cs** file found in the sample Xamarin.Android project on GitHub. Duplicating the **GcmService.cs** and changing class names can be a great place to start as well.

1. Add the following using statements to **MyBroadcastReceiver.cs** (referring to the component and assembly added earlier):

using WindowsAzure.Messaging;

using Gcm.Client;

1. Add the following permission requests between the **using** statements and the **namespace** declaration:

[assembly: Permission(Name = "@PACKAGE\_NAME@.permission.C2D\_MESSAGE")]

[assembly: UsesPermission(Name = "@PACKAGE\_NAME@.permission.C2D\_MESSAGE")]

[assembly: UsesPermission(Name = "com.google.android.c2dm.permission.RECEIVE")]

//GET\_ACCOUNTS is only needed for android versions 4.0.3 and below

[assembly: UsesPermission(Name = "android.permission.GET\_ACCOUNTS")]

[assembly: UsesPermission(Name = "android.permission.INTERNET")]  
[assembly: UsesPermission(Name = "android.permission.WAKE\_LOCK")]

1. In **MyBroadcastReceiver.cs** change the **MyBroadcastReceiver** class to match the following:

[BroadcastReceiver(Permission=Gcm.Client.Constants.PERMISSION\_GCM\_INTENTS)]

[IntentFilter(new string[] { Gcm.Client.Constants.INTENT\_FROM\_GCM\_MESSAGE }, Categories = new string[] { "@PACKAGE\_NAME@" })]

[IntentFilter(new string[] { Gcm.Client.Constants.INTENT\_FROM\_GCM\_REGISTRATION\_CALLBACK }, Categories = new string[] { "@PACKAGE\_NAME@" })]

[IntentFilter(new string[] { Gcm.Client.Constants.INTENT\_FROM\_GCM\_LIBRARY\_RETRY }, Categories = new string[] { "@PACKAGE\_NAME@" })]

public class MyBroadcastReceiver : GcmBroadcastReceiverBase<GcmService>

{

public static string[] SENDER\_IDS = new string[] { Constants.SenderID };

public const string TAG = "MyBroadcastReceiver-GCM";  
}

1. Add another class in **MyBroadcastReceiver.cs** named **PushHandlerService** which derives from **PushHandlerServiceBase**. Make sure to use the **Service**directive on the class:

[Service] //Must use the service tag

public class GcmService : GcmServiceBase

{

public static string RegistrationID { get; private set; }

private NotificationHub Hub { get; set; }

public GcmService() : base(Constants.SenderID)

{

Log.Info(MyBroadcastReceiver.TAG, "GcmService() constructor");

}  
}

1. **GcmServiceBase** implements methods **OnRegistered()**, **OnUnRegistered()**, **OnMessage()**, **OnRecoverableError()**, and **OnError()**. Our implementation class **GcmService** must override these methods, and these methods will fire in response to interacting with the notification hub.
2. Override the **OnRegistered()** method in **PushHandlerService** with the following code:

protected override void OnRegistered(Context context, string registrationId)

{

Log.Verbose(MyBroadcastReceiver.TAG, "GCM Registered: " + registrationId);

RegistrationID = registrationId;

createNotification("GcmService-GCM Registered...", "The device has been Registered, Tap to View!");

Hub = new NotificationHub (Constants.NotificationHubPath, Constants.ConnectionString);

try

{

Hub.UnregisterAll(registrationId);

}

catch (Exception ex)

{

Debug.WriteLine(ex.Message);

Debugger.Break();

}

var tags = new List<string>() { "falcons" }; // create tags if you want

try

{

var hubRegistration = Hub.RegisterNative (registrationId, tags);

}

catch (Exception ex)

{

Debug.WriteLine(ex.Message);

Debugger.Break();

}  
}

NOTE:

In the **OnRegistered()** code above you should note the ability to specify tags to register for specific messaging channels.

1. Override the **OnMessage** method in **PushHandlerService** with the following code:

protected override void OnMessage(Context context, Intent intent)

{

Log.Info(MyBroadcastReceiver.TAG, "GCM Message Received!");

var msg = new StringBuilder();

if (intent != null && intent.Extras != null)

{

foreach (var key in intent.Extras.KeySet())

msg.AppendLine(key + "=" + intent.Extras.Get(key).ToString());

}

string messageText = intent.Extras.GetString("msg");

if (!string.IsNullOrEmpty(messageText))

{

createNotification("New hub message!", messageText);

return;

}

createNotification("Unknown message details", msg.ToString());

}

1. Add the following **createNotification** method to **PushHandlerService** for notifying users as used above:

void createNotification(string title, string desc)

{

//Create notification

var notificationManager = GetSystemService(Context.NotificationService) as NotificationManager;

//Create an intent to show ui

var uiIntent = new Intent(this, typeof(MainActivity));

//Create the notification

var notification = new Notification(Android.Resource.Drawable.SymActionEmail, title);

//Auto cancel will remove the notification once the user touches it

notification.Flags = NotificationFlags.AutoCancel;

//Set the notification info

//we use the pending intent, passing our ui intent over which will get called

//when the notification is tapped.

notification.SetLatestEventInfo(this, title, desc, PendingIntent.GetActivity(this, 0, uiIntent, 0));

//Show the notification

notificationManager.Notify(1, notification);

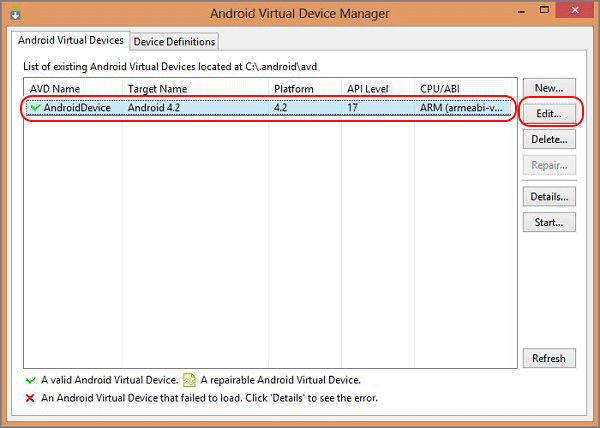
}

1. Override abstract members **OnUnRegistered()**, **OnRecoverableError()**, and **OnError()** so that your code compiles.

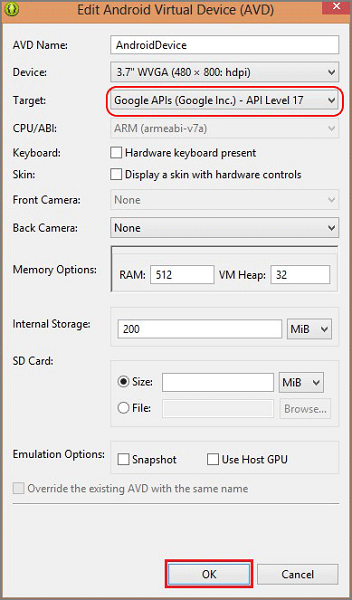
## Run your app in the emulator

When you run this app in the emulator, make sure that you use an Android Virtual Device (AVD) that supports Google APIs.

1. From **Tools**, click **Open Android Emulator Manager**, select your device, and then click **Edit**.



1. Select **Google APIs** in **Target**, then click **OK**.



1. On the top toolbar, click **Run**, and then select your app. This starts the emulator and run the app.
2. The app retrieves the *registrationId* from GCM and registers with the Notification Hub.

IMPORTANT:

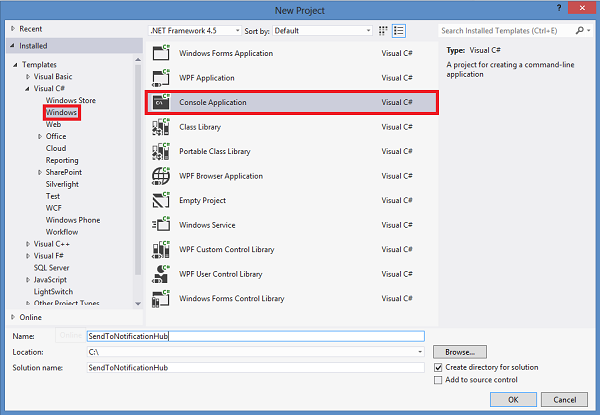
In order to receive push notifications, you must set up a Google account on your Android Virtual Device (in the emulator, navigate to **Settings** and click **Add Account**). Also, make sure that the emulator is connected to the Internet.

# Send notification from your back-end

You can send notifications using Notification Hubs from any back-end using the [REST interface](http://msdn.microsoft.com/library/windowsazure/dn223264.aspx). In this tutorial we will send notifications with a .NET console app, and with a Mobile Service using a node script.

To send notifications using a .NET app:

1. Create a new Visual C# console application:



1. Add a reference to the Azure Service Bus SDK with the [WindowsAzure.ServiceBus NuGet package](http://nuget.org/packages/WindowsAzure.ServiceBus/). In the Visual Studio main menu, click **Tools**, then**Library Package Manager**, then **Package Manager Console**. Then, in the console window type:

Install-Package WindowsAzure.ServiceBus

and press Enter.

1. Open the file Program.cs and add the following using statement:

using Microsoft.ServiceBus.Notifications;

1. In your Program class add the following method:

private static async void SendNotificationAsync()

{

NotificationHubClient hub = NotificationHubClient.CreateClientFromConnectionString("<connection string with full access>", "<hub name>");

await hub.SendGcmNativeNotificationAsync("{ \"data\" : {\"msg\":\"Hello from Azure!\"}}");

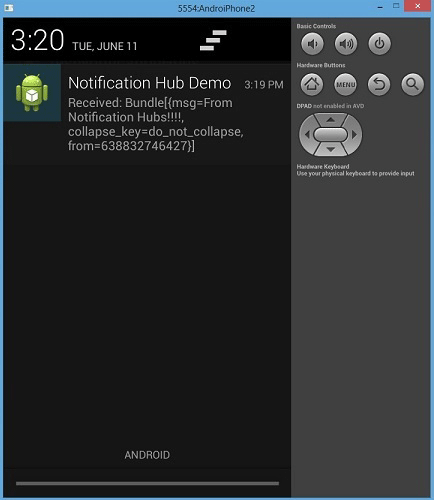
}

1. Then add the following lines in your Main method:

SendNotificationAsync();

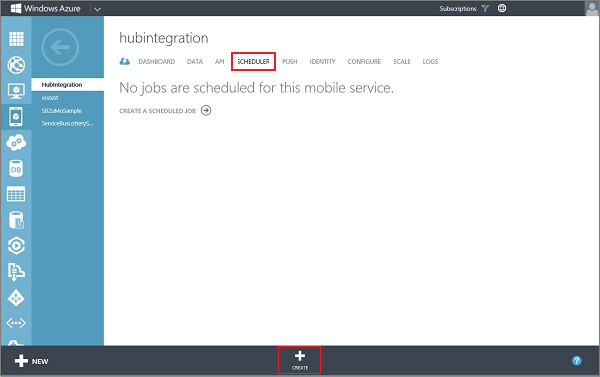
Console.ReadLine();

1. Press the F5 key to run the app. You should receive a toast notification.

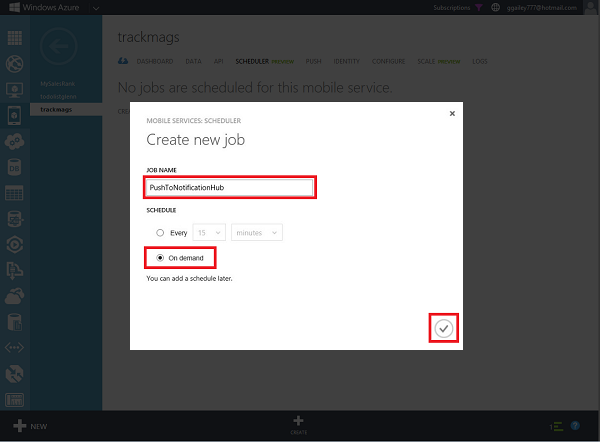


To send a notification using a Mobile Service, follow [Get started with Mobile Services](http://azure.microsoft.com/develop/mobile/tutorials/get-started-xamarin-android/#create-new-service), then:

1. Log on to the [Azure Management Portal](https://manage.windowsazure.com/), and select your Mobile Service.
2. Select the tab **Scheduler** on the top.



1. Create a new scheduled job, insert a name, and select **On demand**.



1. When the job is created, click the job name. Then click the tab **Script** in the top bar.
2. Insert the following script inside your scheduler function. Make sure to replace the placeholders with your notification hub name and the connection string for DefaultFullSharedAccessSignature that you obtained earlier. Click **Save**.

var azure = require('azure');

var notificationHubService = azure.createNotificationHubService('<hub name>', '<connection string>');

notificationHubService.gcm.send(null,'{"data":{"msg" : "Hello from Mobile Services!"}}',

function (error)

{

if (!error) {

console.warn("Notification successful");

}

else

{

console.warn("Notification failed" + error);

}

}

);

1. Click **Run Once** on the bottom bar. You should receive a toast notification.

# Closure

Please delete all services created on the Azure account you were using before you leave so that the next person is not inconvenienced. Thanks.

# Recommended/related sessions

1. Session Name - Track, Hall, Time
2. Session Name - Track, Hall, Time
3. Session Name - Track, Hall, Time

# Tell us what you think

Help us shape future events by sharing your valuable feedback.

Scan the QR code to evaluate *this* session or click the link: \_\_\_\_\_\_\_\_\_\_

< QR Code will be given 2 days before the Conference >



# References

Related references for you to expand your knowledge on the subject:

|  |  |
| --- | --- |
|  | Azure Portal: <http://azure.microsoft.com> |
|  | Azure Updates: <http://azure.microsoft.com/blog/> |
|  | Microsoft Virtual Academy: <http://aka.ms/mva/> |
|  | Developer Network: <http://msdn.microsoft.com/> |

# Azure Support

Must know resources to get online help for Azure:

1. **Azure Support Options** <http://azure.microsoft.com/en-us/support/options/>
2. **Azure Support Plans** <http://azure.microsoft.com/en-us/support/plans/>
3. **Ask questions, & get answers** ****Post questions in the Azure forums Tag questions with the keyword Azure.