Chapter 19: Push Notifications

Section 19.1: Push notifications for Android with Azure

Implementation on Android is a bit more work and requires a specific `Service` to be implemented.

First lets check if our device is capable of receiving push notifications, and if so, register it with Google. This can be done with this code in our `MainActivity.cs` file.

```
protected override void OnCreate(Bundle bundle)
{
    base.OnCreate(bundle);

    global::Xamarin.Forms.Forms.Init(this, bundle);

    // Check to ensure everything's setup right for push
    GcmClient.CheckDevice(this);
    GcmClient.CheckManifest(this);
    GcmClient.Register(this, NotificationsBroadcastReceiver.SenderIDs);

    LoadApplication(new App());
}
```

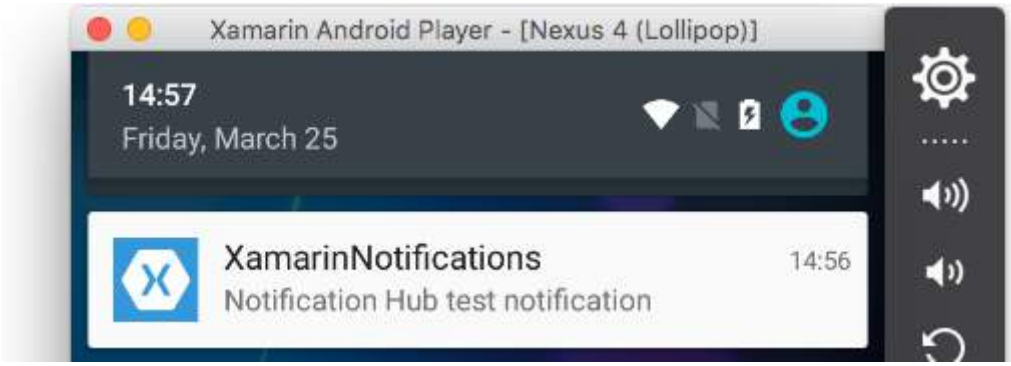
The SenderIDs can be found in the code underneath and is the project number that you get from the Google developer dashboard in order to be able to send push messages.

```
using Android.App;
using Android.Content;
using Gcm.Client;
using Java.Lang;
using System;
using WindowsAzure.Messaging;
using XamarinNotifications.Helpers;

// These attributes are to register the right permissions for our app concerning push messages
[assembly: Permission(Name = "com.versluisit.xamarinnotifications.permission.C2D_MESSAGE")]
[assembly: UsesPermission(Name = "com.versluisit.xamarinnotifications.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")]

namespace XamarinNotifications.Droid.PlatformSpecifics
{
    // These attributes belong to the BroadcastReceiver, they register for the right intents
    [BroadcastReceiver(Permission = Constants.PERMISSION_GCM_INTENTS)]
    [IntentFilter(new[] { Constants.INTENT_FROM_GCM_MESSAGE },
    Categories = new[] { "com.versluisit.xamarinnotifications" })]
    [IntentFilter(new[] { Constants.INTENT_FROM_GCM_REGISTRATION_CALLBACK },
    Categories = new[] { "com.versluisit.xamarinnotifications" })]
    [IntentFilter(new[] { Constants.INTENT_FROM_GCM_LIBRARY_RETRY },
    Categories = new[] { "com.versluisit.xamarinnotifications" })]

    // This is the broadcast receiver
    public class NotificationsBroadcastReceiver : GcmBroadcastReceiverBase<PushHandlerService>
    {
        // TODO add your project number here
```

```
        public static string[] SenderIDs = { "96688------" };
    }

    [Service] // Don't forget this one! This tells Xamarin that this class is a Android Service
    public class PushHandlerService : GcmServiceBase
    {
        // TODO add your own access key
        private string _connectionString =
ConnectionString.CreateUsingSharedAccessKeyWithListenAccess(
            new Java.Net.URI("sb://xamarinnotifications-ns.servicebus.windows.net/"), "<your key
here>");

        // TODO add your own hub name
        private string _hubName = "xamarinnotifications";

        public static string RegistrationID { get; private set; }

        public PushHandlerService() : base(NotificationsBroadcastReceiver.SenderIDs)
        {
        }

        // This is the entry point for when a notification is received
        protected override void OnMessage(Context context, Intent intent)
        {
            var title = "XamarinNotifications";

            if (intent.Extras.ContainsKey("title"))
                title = intent.Extras.GetString("title");

            var messageText = intent.Extras.GetString("message");

            if (!string.IsNullOrEmpty(messageText))
                CreateNotification(title, messageText);
        }

        // The method we use to compose our notification
        private void CreateNotification(string title, string desc)
        {
            // First we make sure our app will start when the notification is pressed
            const int pendingIntentId = 0;
            const int notificationId = 0;

            var startupIntent = new Intent(this, typeof(MainActivity));
            var stackBuilder = TaskStackBuilder.Create(this);

            stackBuilder.AddParentStack(Class.FromType(typeof(MainActivity)));
            stackBuilder.AddNextIntent(startupIntent);

            var pendingIntent =
                stackBuilder.GetPendingIntent(pendingIntentId, PendingIntentFlags.OneShot);

            // Here we start building our actual notification, this has some more
            // interesting customization options!
            var builder = new Notification.Builder(this)
                .SetContentIntent(pendingIntent)
                .SetContentTitle(title)
                .SetContentText(desc)
                .SetSmallIcon(Resource.Drawable.icon);

            // Build the notification
            var notification = builder.Build();
            notification.Flags = NotificationFlags.AutoCancel;
```

```
            // Get the notification manager
            var notificationManager =
                GetSystemService(NotificationService) as NotificationManager;

            // Publish the notification to the notification manager
            notificationManager.Notify(notificationId, notification);
        }

        // Whenever an error occurs in regard to push registering, this fires
        protected override void OnError(Context context, string errorId)
        {
            Console.Out.WriteLine(errorId);
        }

        // This handles the successful registration of our device to Google
        // We need to register with Azure here ourselves
        protected override void OnRegistered(Context context, string registrationId)
        {
            var hub = new NotificationHub(_hubName, _connectionString, context);

            Settings.DeviceToken = registrationId;

            // TODO set some tags here if you want and supply them to the Register method
            var tags = new string[] { };

            hub.Register(registrationId, tags);
        }

        // This handles when our device unregisters at Google
        // We need to unregister with Azure
        protected override void OnUnRegistered(Context context, string registrationId)
        {
            var hub = new NotificationHub(_hubName, _connectionString, context);

            hub.UnregisterAll(registrationId);
        }
    }
}
```

A sample notification on Android looks like this.



Section 19.2: Push notifications for iOS with Azure

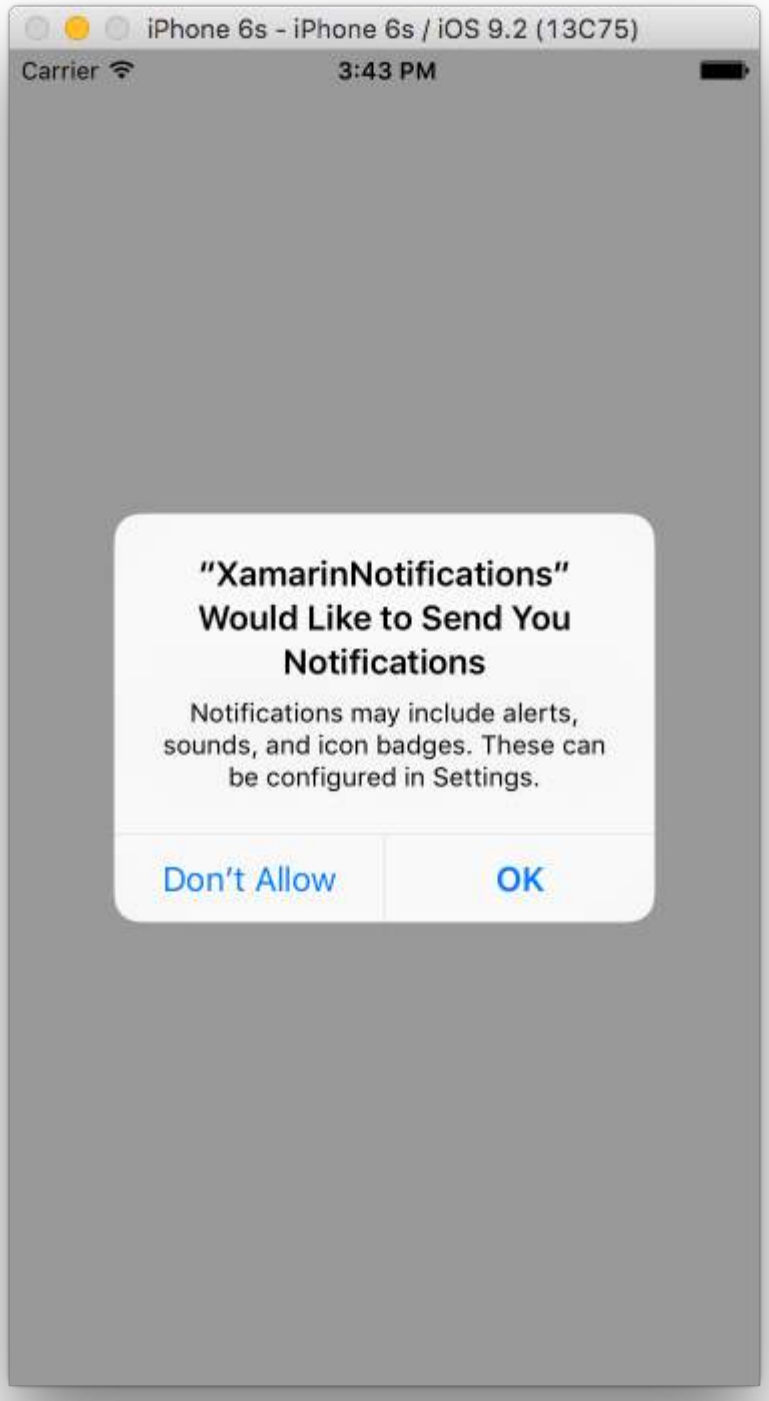
To start the registration for push notifications you need to execute the below code.

```
// registers for push
var settings = UIUserNotificationSettings.GetSettingsForTypes(
    UIUserNotificationType.Alert
    | UIUserNotificationType.Badge
```

```
    | UIUserNotificationType.Sound,
    new NSSet());

UIApplication.SharedApplication.RegisterUserNotificationSettings(settings);
UIApplication.SharedApplication.RegisterForRemoteNotifications();
```

This code can either be ran directly when the app starts up in the `FinishedLaunching` in the `AppDelegate.cs` file. Or you can do it whenever a user decides that they want to enable push notifications.

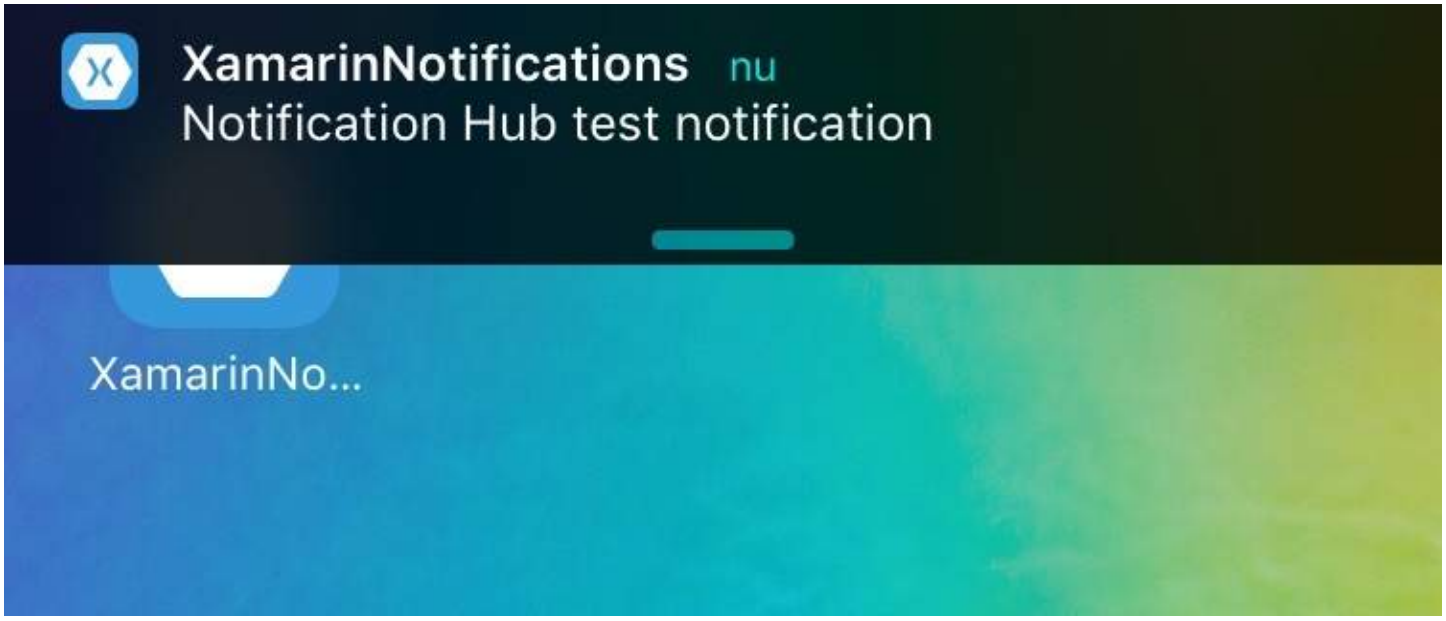
Running this code will trigger an alert to prompt the user if they will accept that the app can send them notifications. So also implement a scenario where the user denies that!



These are the events that need implementation for implementing push notifications on iOS. You can find them in the `AppDelegate.cs` file.

```
// We've successfully registered with the Apple notification service, or in our case Azure
public override void RegisteredForRemoteNotifications(UIApplication application, NSData
deviceToken)
{
   // Modify device token for compatibility Azure
    var token = deviceToken.Description;
    token = token.Trim('<', '>').Replace(" ", "");

    // You need the Settings plugin for this!
    Settings.DeviceToken = token;

    var hub = new SBNotificationHub("Endpoint=sb://xamarinnotifications-
ns.servicebus.windows.net/;SharedAccessKeyName=DefaultListenSharedAccessSignature;SharedAccessKey=<
your own key>", "xamarinnotifications");

    NSSet tags = null; // create tags if you want, not covered for now
    hub.RegisterNativeAsync(deviceToken, tags, (errorCallback) =>
    {
        if (errorCallback != null)
        {
            var alert = new UIAlertView("ERROR!", errorCallback.ToString(), null, "OK", null);
            alert.Show();
        }
    });
}

// We've received a notification, yay!
public override void ReceivedRemoteNotification(UIApplication application, NSDictionary userInfo)
{
    NSObject inAppMessage;

    var success = userInfo.TryGetValue(new NSString("inAppMessage"), out inAppMessage);

    if (success)
    {
        var alert = new UIAlertView("Notification!", inAppMessage.ToString(), null, "OK", null);
        alert.Show();
    }
}

// Something went wrong while registering!
public override void FailedToRegisterForRemoteNotifications(UIApplication application, NSError
error)
{
   var alert = new UIAlertView("Computer says no", "Notification registration failed! Try again!",
null, "OK", null);

   alert.Show();
}
```

When a notification is received this is what it looks like.



Section 19.3: iOS Example

1. You will need a development device
2. Go to your Apple Developer Account and create a provisioning profile with Push Notifications enabled
3. You will need some sort of way to notify your phone (AWS, Azure..etc) **We will use AWS here**

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    global::Xamarin.Forms.Forms.Init();

   //after typical Xamarin.Forms Init Stuff

   //variable to set-up the style of notifications you want, iOS supports 3 types

   var pushSettings = UIUserNotificationSettings.GetSettingsForTypes(
              UIUserNotificationType.Alert |
              UIUserNotificationType.Badge |
              UIUserNotificationType.Sound,
              null );


        //both of these methods are in iOS, we have to override them and set them up
        //to allow push notifications

        app.RegisterUserNotificationSettings(pushSettings);  //pass the supported push
notifications settings to register app in settings page


}

public override async void RegisteredForRemoteNotifications(UIApplication application, NSData
token)
    {
        AmazonSimpleNotificationServiceClient snsClient = new
AmazonSimpleNotificationServiceClient("your AWS credentials here");

        // This contains the registered push notification token stored on the phone.
        var deviceToken = token.Description.Replace("<", "").Replace(">", "").Replace(" ", "");

        if (!string.IsNullOrEmpty(deviceToken))
        {
```

```
            //register with SNS to create an endpoint ARN, this means AWS can message your phone
            var response = await snsClient.CreatePlatformEndpointAsync(
            new CreatePlatformEndpointRequest
            {
                Token = deviceToken,
                PlatformApplicationArn = "yourARNwouldgohere" /* insert your platform application
ARN here */
            });

            var endpoint = response.EndpointArn;

            //AWS lets you create topics, so use subscribe your app to a topic, so you can easily
send out one push notification to all of your users
            var subscribeResponse = await snsClient.SubscribeAsync(new SubscribeRequest
            {
                TopicArn = "YourTopicARN here",
                Endpoint = endpoint,
                Protocol = "application"

            });

        }

    }
```