Notification Hub (Part 3)

Documentation for running application in iOS:

Download the application from the repo:
https://github.com/andrewchungxam/XamarinFormsPushNotificationSample

This doesn't work in simulator, so you will need the appropriate provisioning profile (push notifications require specific permissions). The iOS simulator does not support push notifications.

Try to run this application on your phone (device only) - it will likely fail and the error message will complain that you don't have the original author's provisioning profile (this is because the original author did not list your iPhone as one of the acceptable iPhones that his provisioning profile could work with).

So you're going to need to set up your provisioning profile.

PROVISIONING PROFILE:
For iOS, you will need an Apple Developer Program membership and a physical iOS device.

Here are the steps that are needed to created the appropriate Provisioning profile ** (NOTE: there is one step at the end requires some extra work. See note below the following url)
https://docs.microsoft.com/en-us/azure/app-service-mobile/app-service-mobile-xamarine-forms-get-started-push#generate-the-certificate-signing-request-file

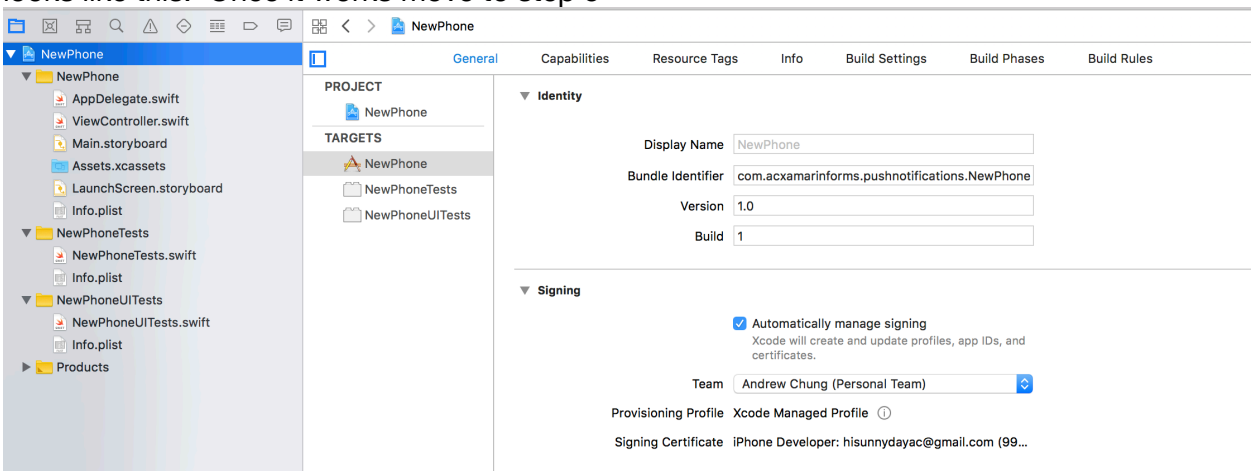
Follow the above directions only up to this line: *This ensures that the project uses the new profile for code signing. For the official Xamarin device provisioning documentation, see Xamarin Device Provisioning.*

The above steps and directions are clear and good - HOWEVER, sometimes at the final steps when you have to upload the SSL certificate up to Azure there could be a small issue in how the keychain on your Mac will show the certificate. You will know you hit an error because you will either not be able to export the certificate in the .p12 format — or if you are able to upload it, Azure will say that it is not in the correct format. Use the solutions here to solve the issue: https://stackoverflow.com/questions/15662377/unable-to-export-apple-production-push-ssl-certificate-in-p12-format/21060610#21060610 (I had to use BOTH the first and the second answer before it worked properly).

RUNNING YOUR APP ON DEVICE:
Here are the official docs: Xamarin Device Provisioning

If you have never run an app on an iPhone - I do the following:
If my summarized version doesn't work please check out the official doc:
1. Close down Xcode / Visual Studio for Mac / Any Simulators.
2. Plug your iPhone (device) to your Mac.
3. You'll be prompted to Trust your computer (If you don't trust it, you won't be able to

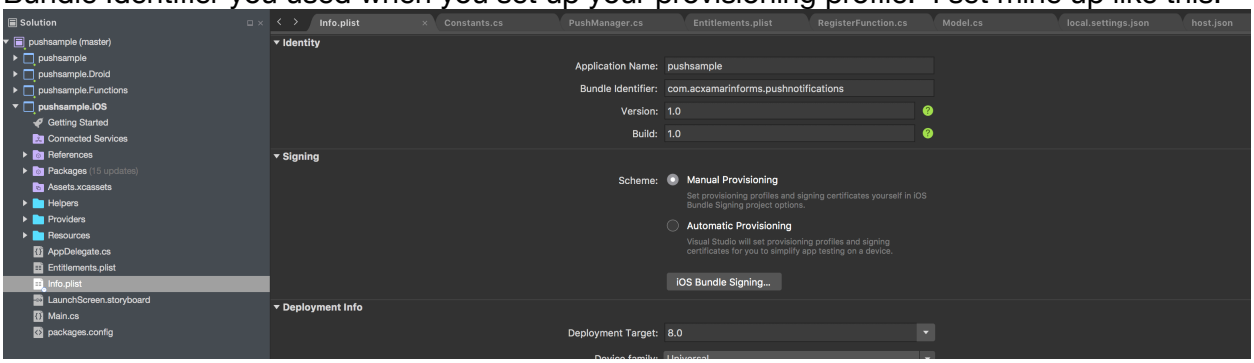
interact with your device from your computer.) (if you don't see that prompt - try opening iTunes)
4. Open Xcode - create a new project - run this project on your iPhone. My signing options looks like this: Once it works move to step 5



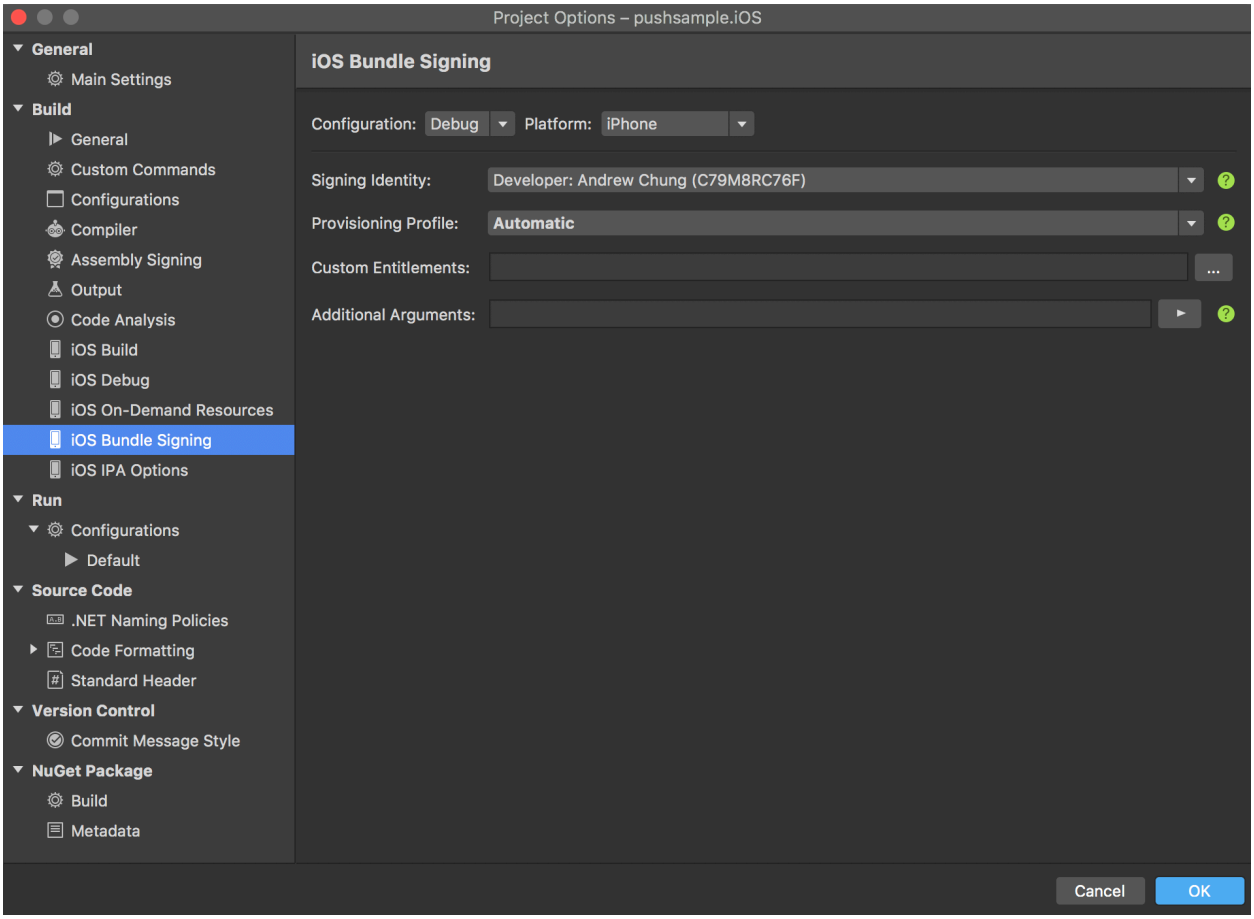
5.Open up Visual Studio for Mac - create a new project. Xamarin.Forms or iOS and then deploy to your device.

USING THE PUSH NOTIFICATIONS PROVISIONING PROFILE:
Now you've got your IDE/Iphone/Provisioning profiles working…as a next step, deploy this app, Xamarin Forms Push Notification Sample (https://github.com/andrewchungxam/Xamarin-Forms-Push-Notifications-Sample), to your device.

This time you want to use the provisioning profile you step up earlier.
In your solution tree > go to the pushSample.iOS - you want to set it up like this - using the Bundle Identifier you used when you set up your provisioning profile. I set mine up like this:

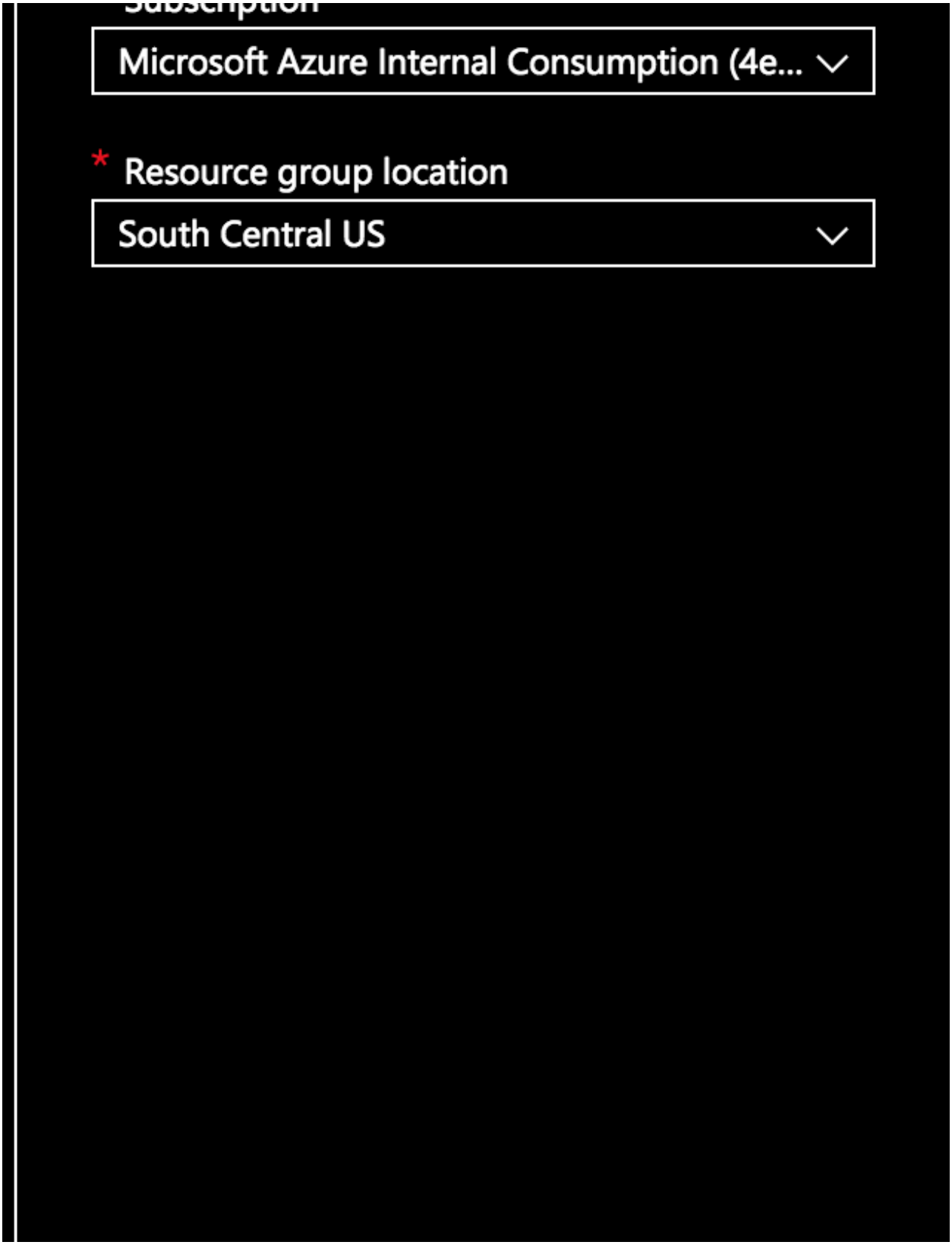


When you click iOS Bundle Singing, you see a screen that looks like this:

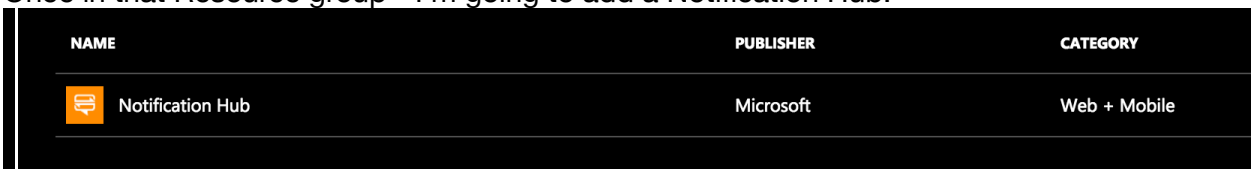


CREATING THE NOTIFICATION HUB:

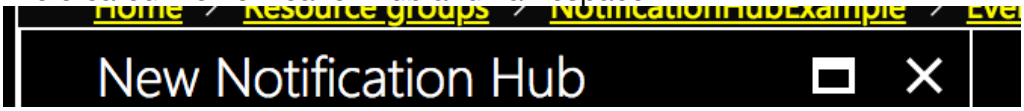
I created a new Resource group in Portal.Azure.com

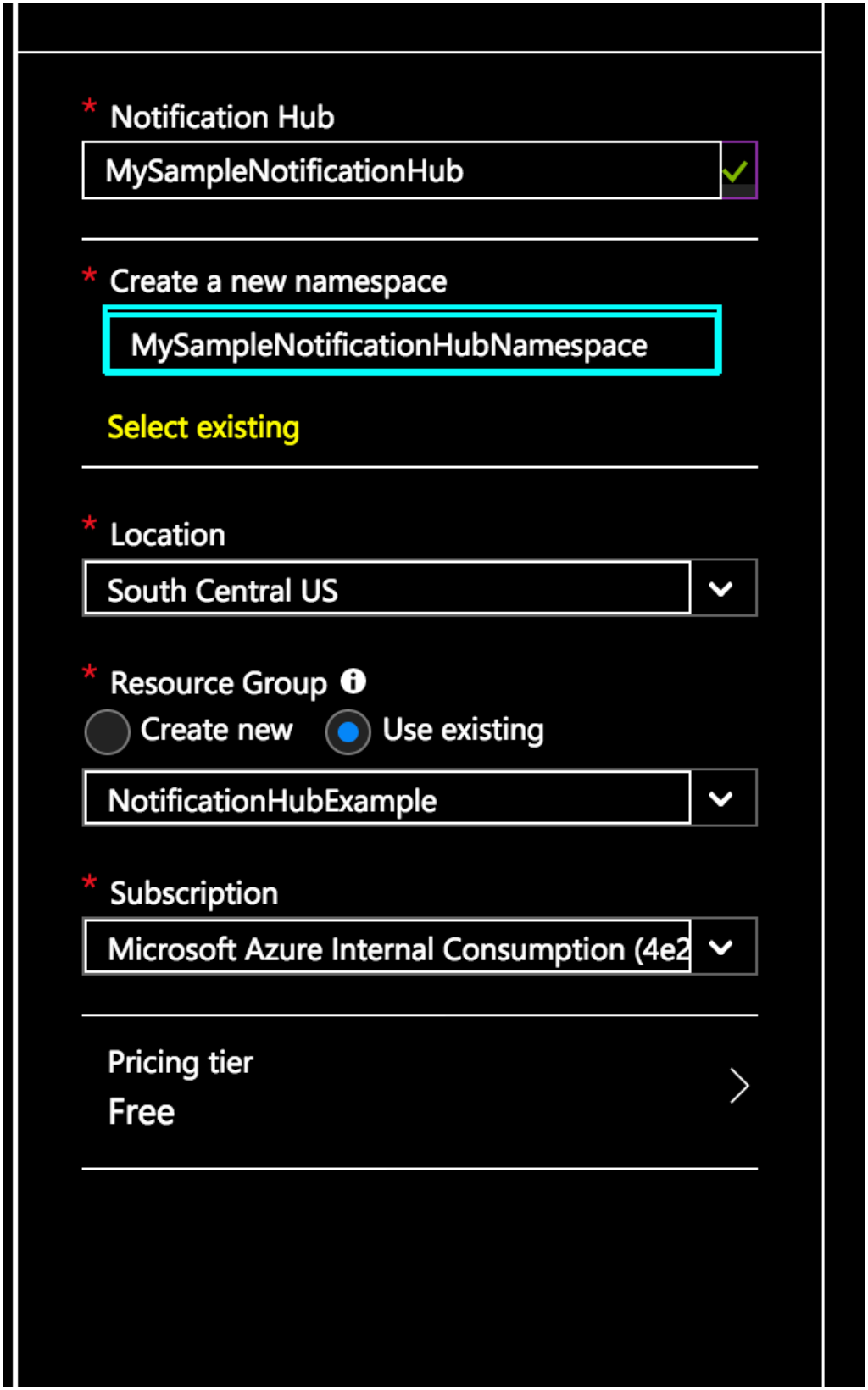

Microsoft Azure Internal Consumption (4e...
* Resource group location
South Central US

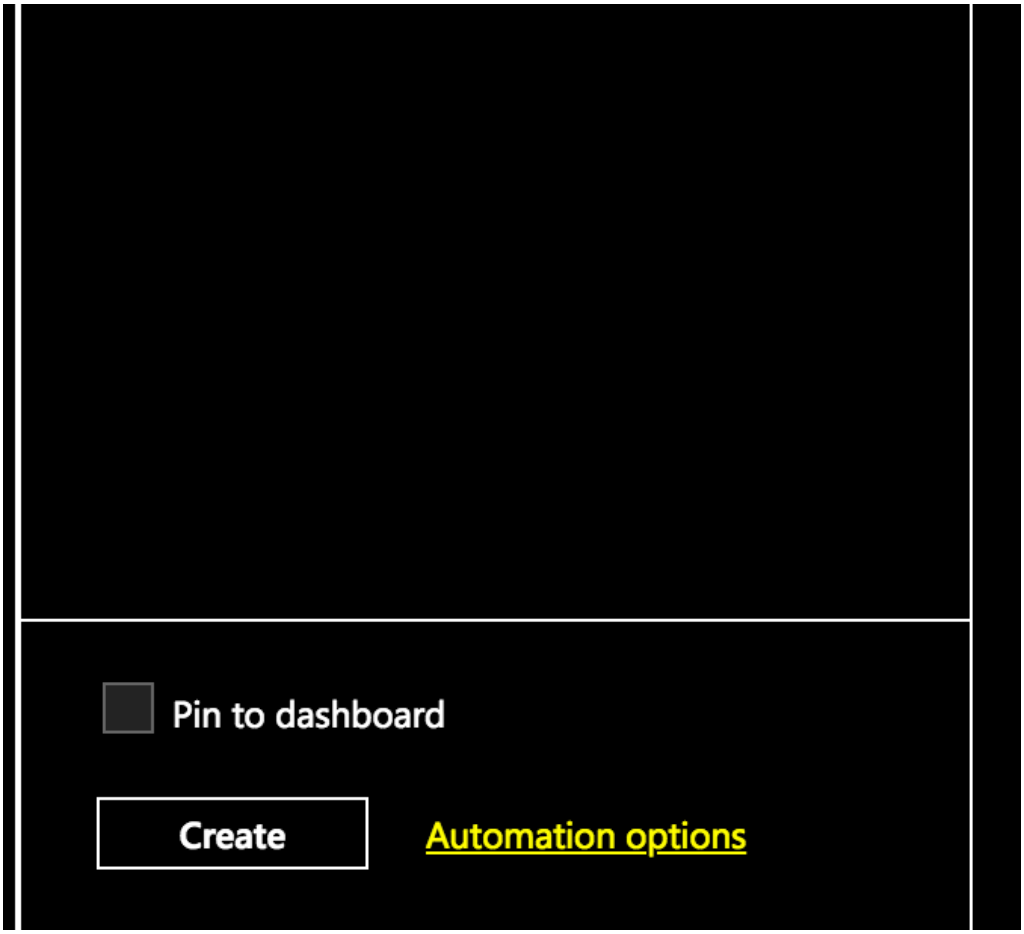
Once in that Resource group - I’m going to add a Notification Hub:



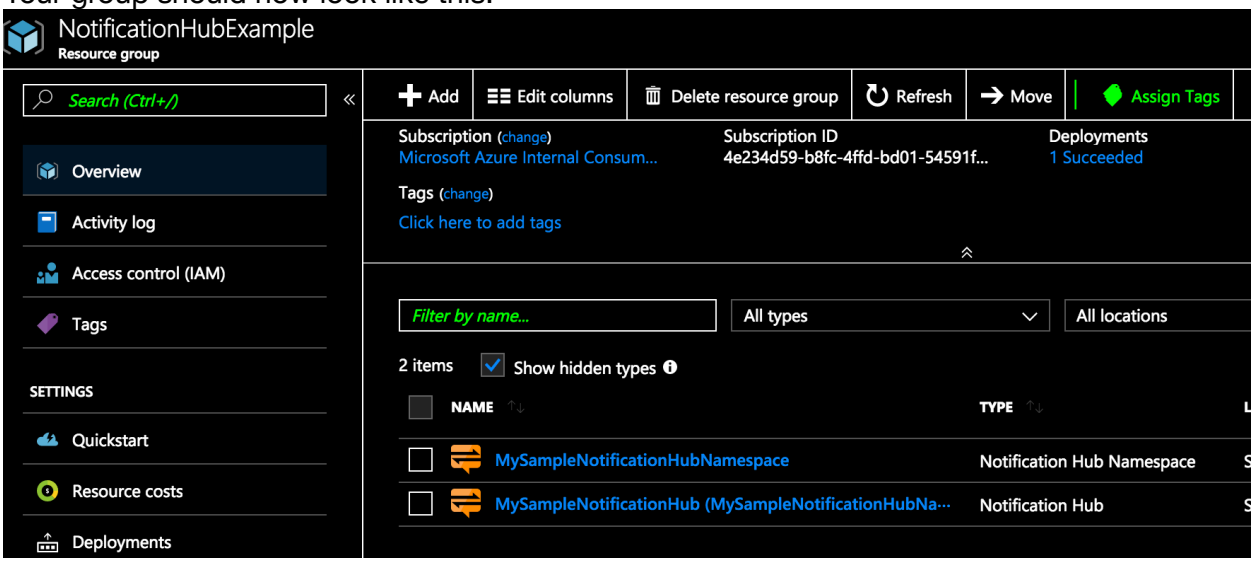
I’ve created the notification hub and namespace:




* Notification Hub
MySampleNotificationHub
* Create a new namespace
MySampleNotificationHubNamespace
Select existing
* Location
South Central US
* Resource Group
Create new
Use existing
NotificationHubExample
* Subscription
Microsoft Azure Internal Consumption (4e2
Pricing tier
Free

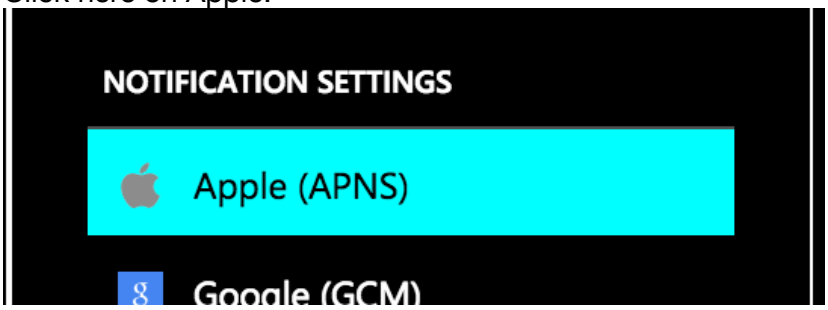


Your group should now look like this:



Click on your Notification Hub (not the Notification Hub Namespace)

Click here on Apple:

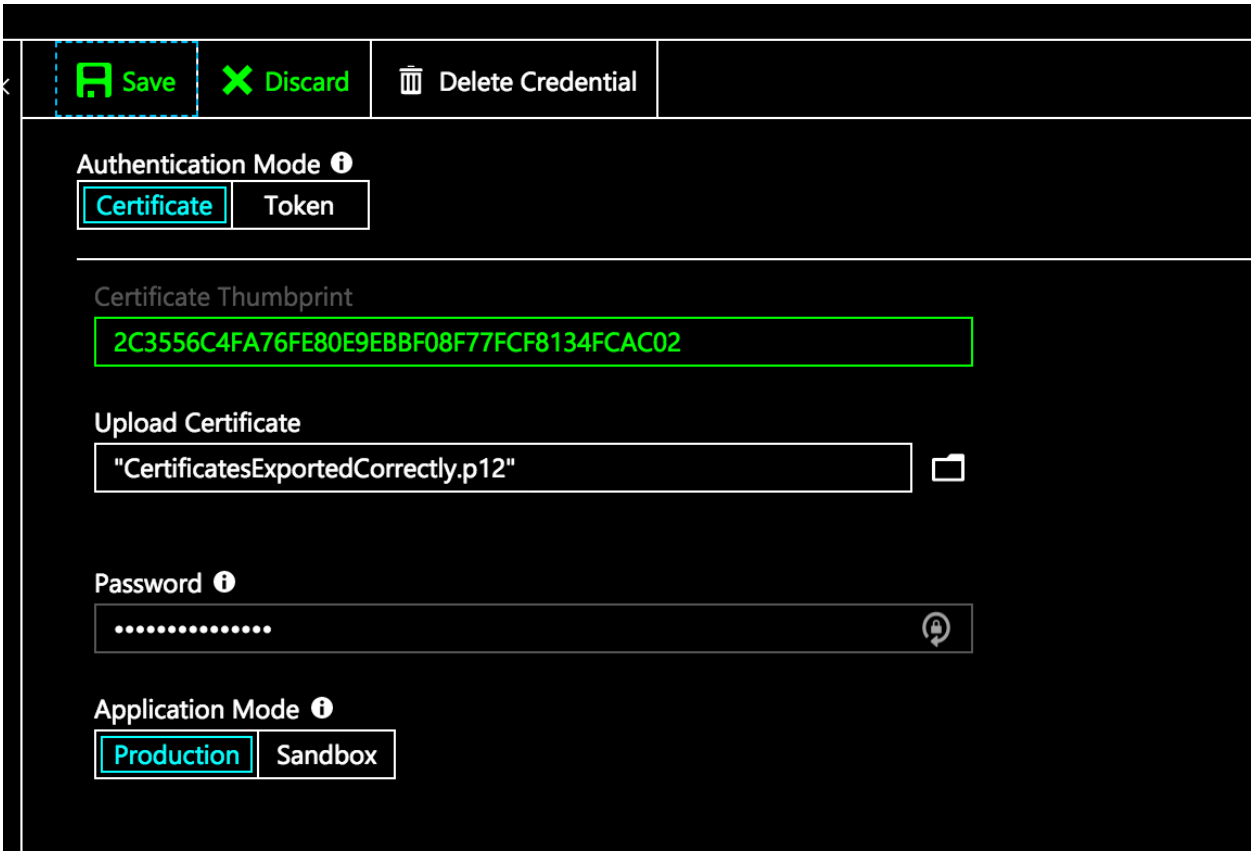


I clicked the Certificate under AUTHENTICATION MODE

*** MAKE SURE YOU DO SANDBOX FOR THIS RUNTHROUGH:
https://stackoverflow.com/questions/38847071/azure-ios-push-notification-not-receiving-on-production-mode

You must maintain two different hubs: one hub for production, and another hub for testing. This means that you must upload the certificate that you use in a sandbox environment to a separate hub than the certificate and hub that you are going to use in production. Don't try to upload different types of certificates to the same hub. This might cause notification failures.

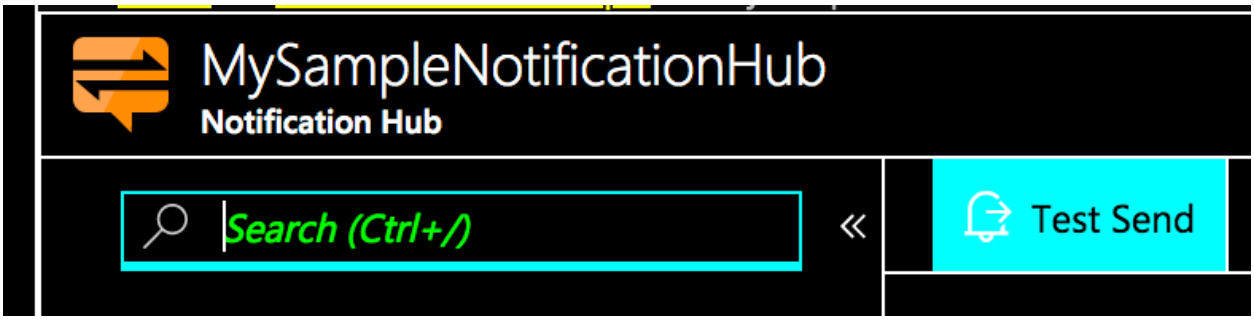
For this run-through - make sure you're selecting SANDBOX (not production):



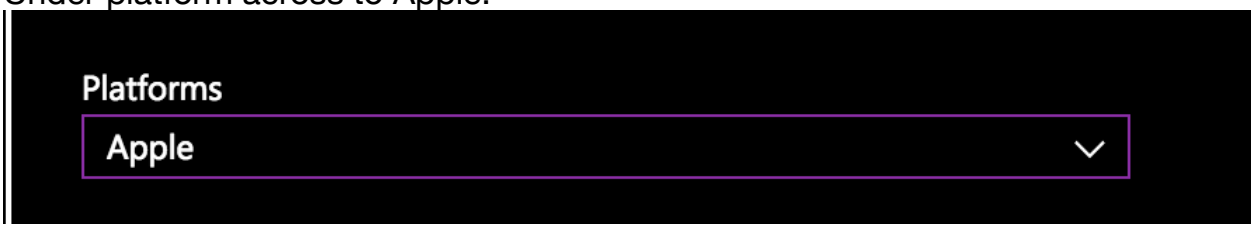
It asks me to upload certificate - which is what you did in step 8 here: https://docs.microsoft.com/en-us/azure/app-service-mobile/app-service-mobile-xamarin-forms-get-started-push#generate-the-certificate-signing-request-file

It directions should look like this...In Keychain Access, right-click the new push certificate that you created in the **Certificates**category. Click **Export**, name the file, select the **.p12** format (Remember if you're having trouble reference answer 1 and 2 in above this post: https://stackoverflow.com/questions/15662377/unable-to-export-apple-production-push-ssl-certificate-in-p12-format/21060610#21060610)

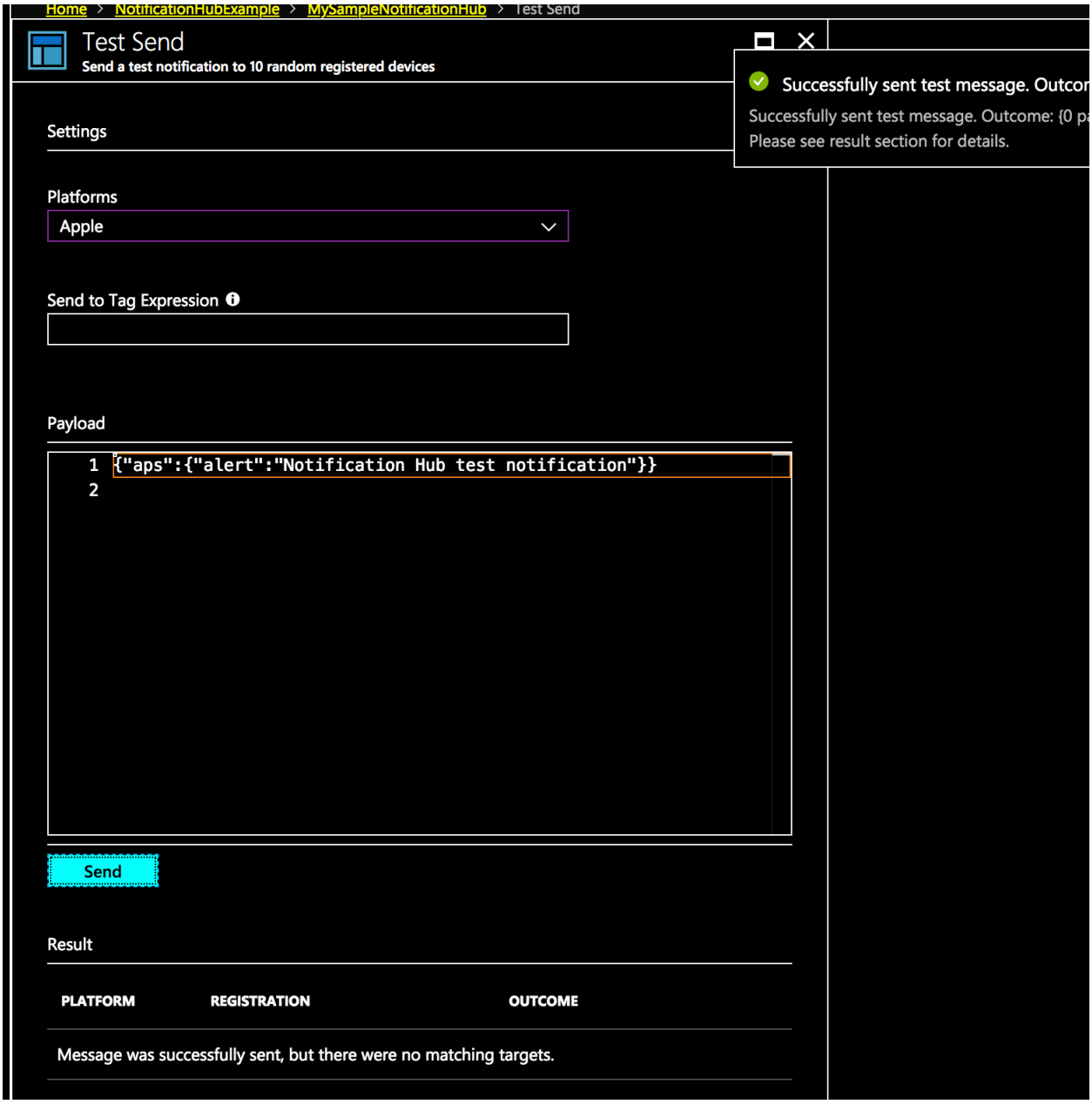
After this is set - you can send a "test" notification. It won't go anywhere but let's just verify that something gets sent on the backend.



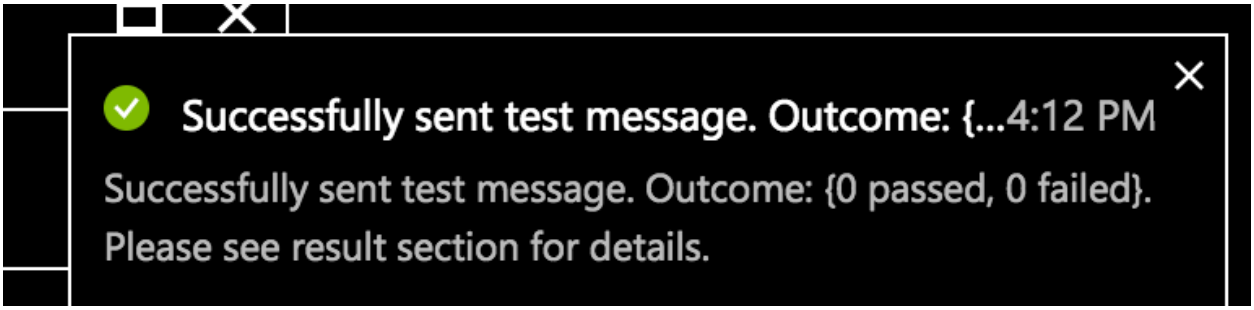
Click “Test Send”
Under platform across to Apple:



Keep everything default - and click Send.



Hopefully - you'll this success message pop-up in the upper right.



CLIENT SIDE : XAMARIN.FORMS:

In Visual Studio for Mac, I create a new blank Xamarin.Forms application.

The options don't matter too much (I prefer UI in C# vs. XAML).

What does matter is that the Bundle Identifier in Project > Project.iOS > info.plist is set to what you defined when you were creating your provisioning profile.

So not that we've got a blank app - let's start adding code.

We're going to use Rene's code from here as a jumping off point:
https://github.com/Krumelur/XamUAzureNotificationHub

I'll walk line-by-line what we'll be adding and why we're doing it.

Please note - if you see errors as you are working through the code, be sure to right click / Quick Fix / and add the necessary "using" statements.

In your app delegate - you're going to need a new method - called RequestPushPermissionAsync(); This will prompt the user to accept Notifications.

(This code of course is on the Github project:
https://github.com/andrewchungxam/XamarinFormsPushNotificationSample

AppDelegate.cs

```
18
19          //#ADD Section 1 - Step 1
20          RequestPushPermissionAsync();
21
```

We'll define the method further down (note the code is for iOS 10 and above):

It basically asks for your permission — notice there is a check at the Request but also that it needs to check the settings to make sure your user didn't revoke the permission in the device's Settings since the last time the app was used.

```
async Task RequestPushPermissionAsync()
{
    // iOS10 and later (https://developer.xamarin.com/guides/ios/platform_features/user-notifications/enhanced-user
    // Register for ANY type of notification (local or remote):
    var requestResult = await UNUserNotificationCenter.Current.RequestAuthorizationAsync(
        UNAuthorizationOptions.Alert
        | UNAuthorizationOptions.Badge
        | UNAuthorizationOptions.Sound);


    // Item1 = approved boolean
    bool approved = requestResult.Item1;
    NSError error = requestResult.Item2;
    if (error == null)
    {
        // Handle approval
        if (!approved)
        {
            Console.Write("Permission to receive notifications was not granted.");
            return;
        }

        var currentSettings = await UNUserNotificationCenter.Current.GetNotificationSettingsAsync();
        if (currentSettings.AuthorizationStatus != UNAuthorizationStatus.Authorized)
        {
            Console.WriteLine("Permissions were requested in the past but have been revoked (-> Settings app).");
            return;
        }

        UIApplication.SharedApplication.RegisterForRemoteNotifications();
    }
    else
    {
        Console.Write($"Error requesting permissions: {error}.");
    }
}
```

Next - we need to add Register for Remote Notification:

AppDelegate.cs

```
public async override void RegisteredForRemoteNotifications(
        UIApplication application, NSData deviceToken)
{
        if (deviceToken == null)
        {
                // Can happen in rare conditions e.g. after restoring a device.
                return;
        }

        Console.WriteLine($"Token received: {deviceToken}");
        await SendRegistrationToServerAsync(deviceToken);
}
```

Send to the server the registration information (note - we'll change that client below to a Function to simplify deployment — the most updated code will be in the Github. In the sample project - you're looking at you'll likely see that "var client = new MobileServiceClient(…" code commented out or replaced with a function.)

AppDelegate.cs

```
async Task SendRegistrationToServerAsync(NSData deviceToken)
{
        // This is the template/payload used by iOS. It contains the "messageParam"
        // that will be replaced by our service.
        const string templateBodyAPNS = "{\"aps\":{\"alert\":\"$(messageParam)\"}}";

        var templates = new JObject();
        templates["genericMessage"] = new JObject
        {
                {"body", templateBodyAPNS}
        };

        var client = new MobileServiceClient(XamUNotif.App.MobileServiceUrl);
        await client.GetPush().RegisterAsync(deviceToken, templates);
}
```

Handle the situation where there was a failure to register:

AppDelegate.cs

```
public override void FailedToRegisterForRemoteNotifications(UIApplication application, NSError error)
{
        Console.WriteLine($"Failed to register for remote notifications: {error.Description}");
}
```

Finally - Receive and Present the notification:

AppDelegate.cs

```
public override void DidReceiveRemoteNotification(
        UIApplication application,
        NSDictionary userInfo,
        Action<UIBackgroundFetchResult> completionHandler)
{
        // This will be called if the app is in the background/not running and if in the foreground.
        // However, it will not display a notification visually if the app is in the foreground.

        PresentNotification(userInfo);

        completionHandler(UIBackgroundFetchResult.NoData);
}

void PresentNotification(NSDictionary dict)
{
        // Extract some data from the notifiation and display it using an alert view.
        NSDictionary aps = dict.ObjectForKey(new NSString("aps")) as NSDictionary;

        var msg = string.Empty;
        if (aps.ContainsKey(new NSString("alert")))
        {
                msg = (aps[new NSString("alert")] as NSString).ToString();
        }

        if (string.IsNullOrEmpty(msg))
        {
                msg = "(unable to parse)";
        }

        MessagingCenter.Send<object, string>(this, App.NotificationReceivedKey, msg);
}
```

Next - we're going to back up to the FinishedLaunching method of the AppDelegate.cs

We're going to add _launch options.
Why? This will help us for when the user received a notification but the app is not open.

This will be in options under the FinishedLaunching parameters. Upon activation, OnActivated will run and then it will call the same PresentNotifications we covered above.

Take a look at the code here:

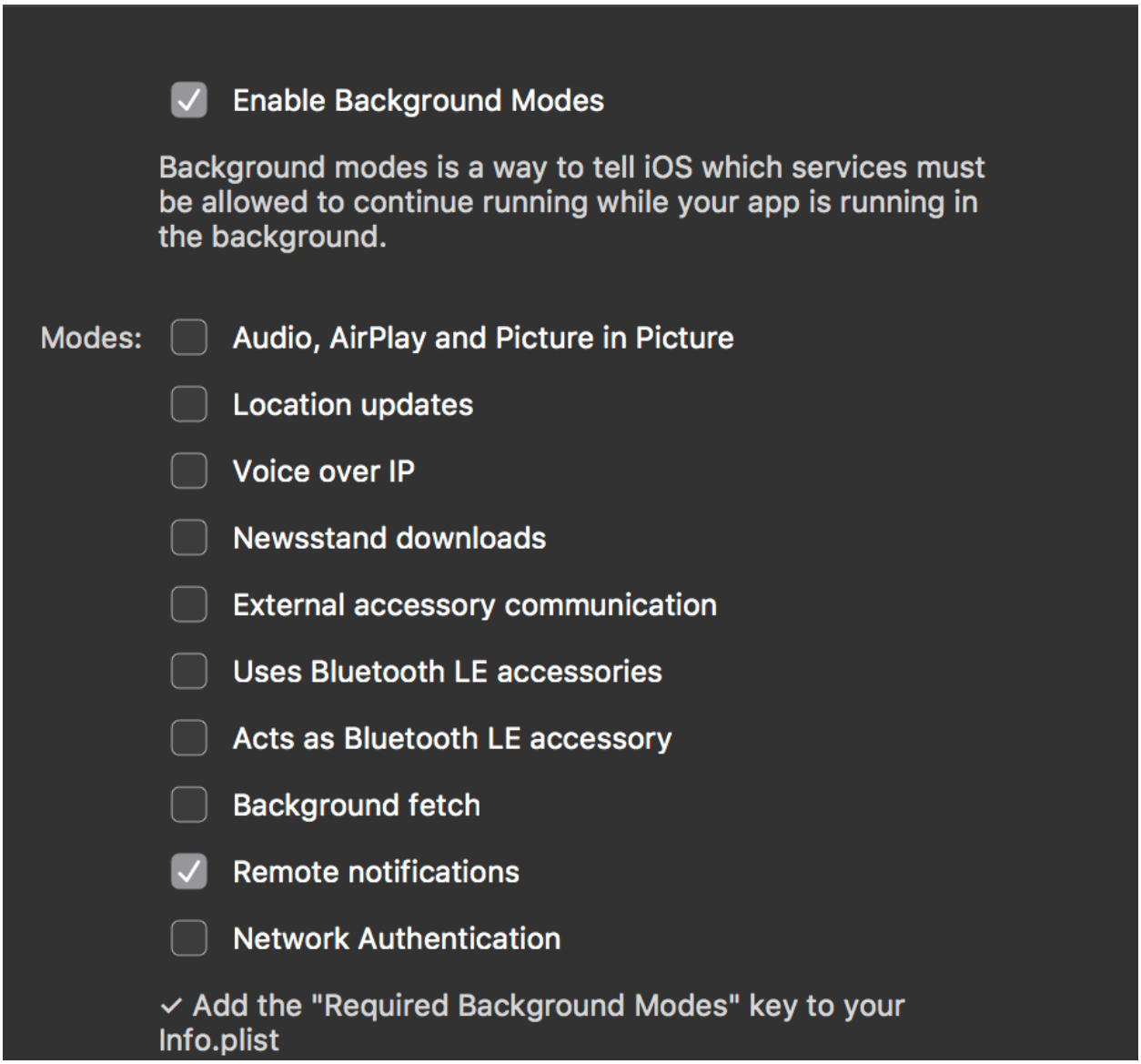
AppDelegate.cs

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

        RequestPushPermissionAsync();

        _launchOptions = options;

        return base.FinishedLaunching(app, options);
}

NSDictionary _launchOptions;

public override void OnActivated(UIApplication uiApplication)
{
        base.OnActivated(uiApplication);
        // If app was not running and we come from a notificatio badge, the notification is delivered via the options.
        if (_launchOptions != null && _launchOptions.ContainsKey(UIApplication.LaunchOptionsRemoteNotificationKey))
        {
                var notification = _launchOptions[UIApplication.LaunchOptionsRemoteNotificationKey] as NSDictionary;
                PresentNotification(notification);
        }
        _launchOptions = null;
}
```

SETTINGS

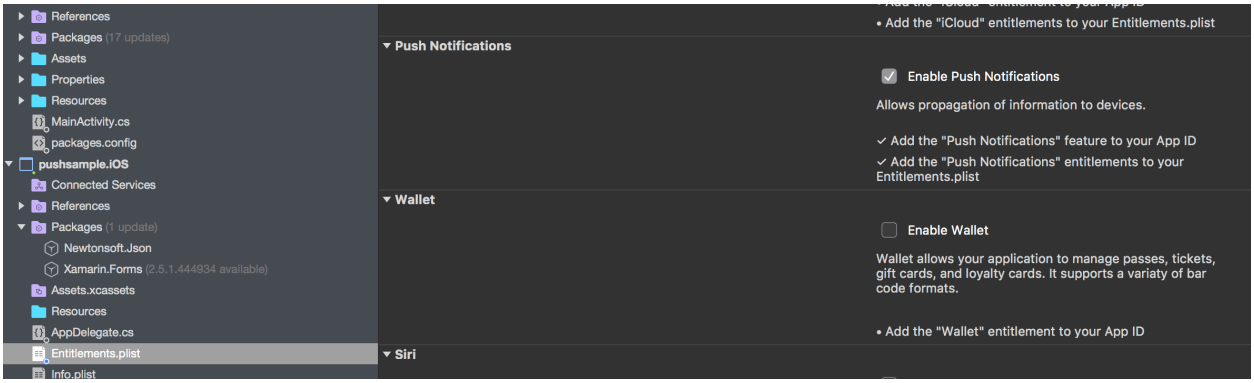
Now there are a couple things in the project settings that we've got to select so that it can receive background notifications.

Go to Info.plist and then scroll down all the way to Background Modes - click the box and more options will appear.



^ If that isn't set - it will not receive the notification.

Also set Enable Push Notifications in the Entitlements:

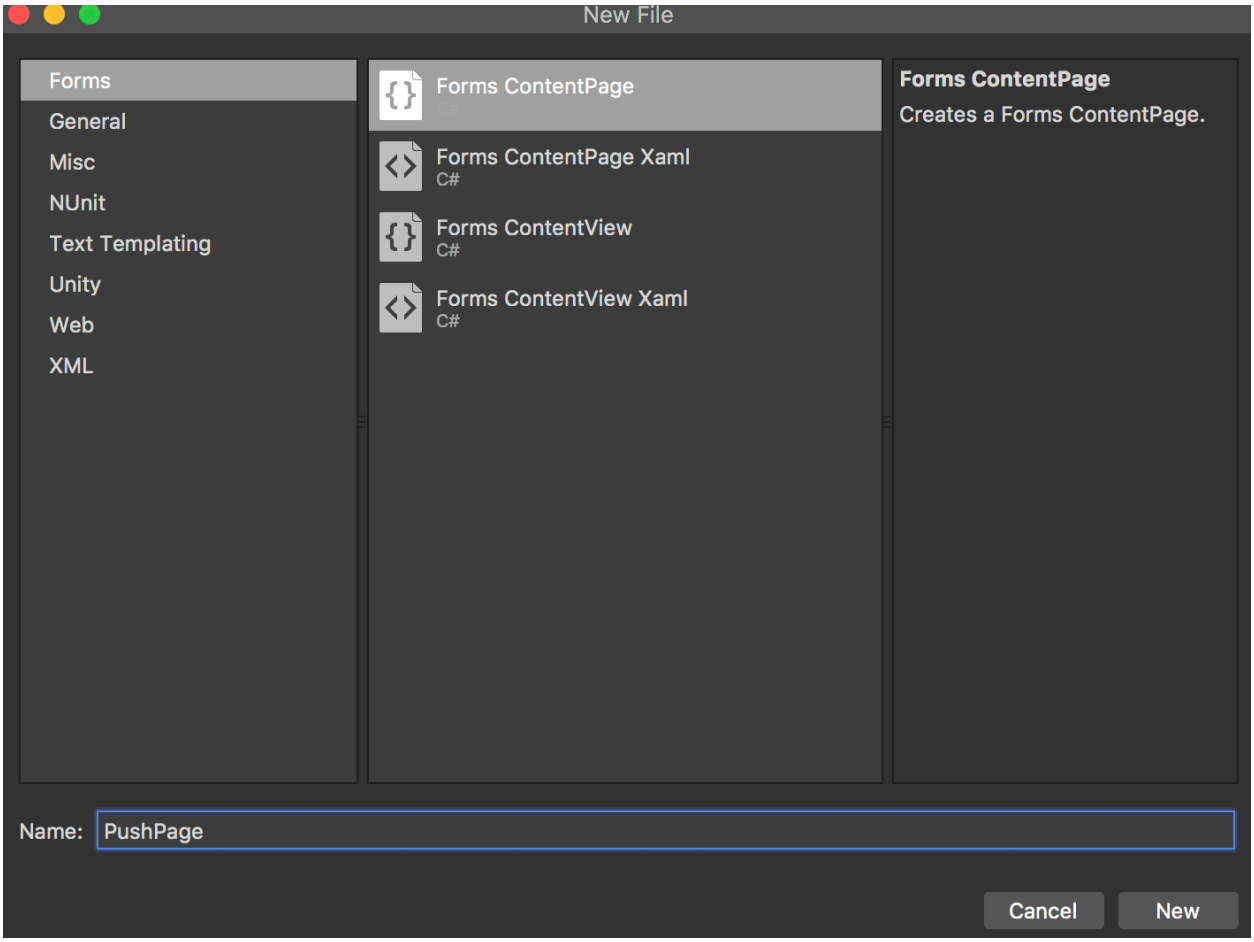


NEXT STEPS:
Now let's re-work the UI of the application.
Also - let's use a couple programs to test out how far we've actually gotten. We're going to use something akin "Postman" to send a test notification to make sure the app is correctly setup. This app is called Pusher.
After that we'll use an Azure Function to Interact with the Notification Hub.

Re-work the UI of the Application:

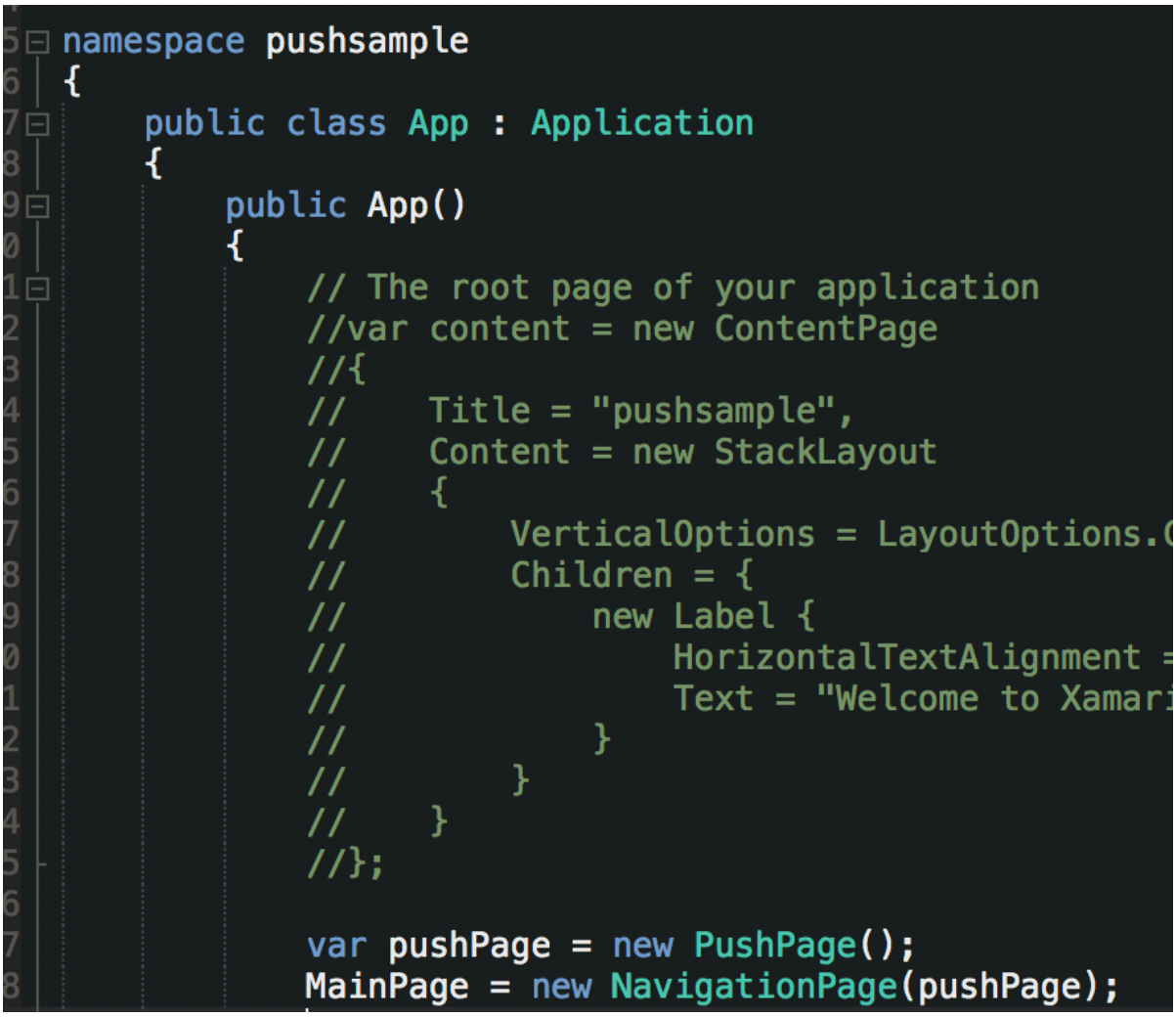
I've used a Blank app.

Create a new Forms > ContentPage



Then let's have that page be our first page.

pushsample.cs

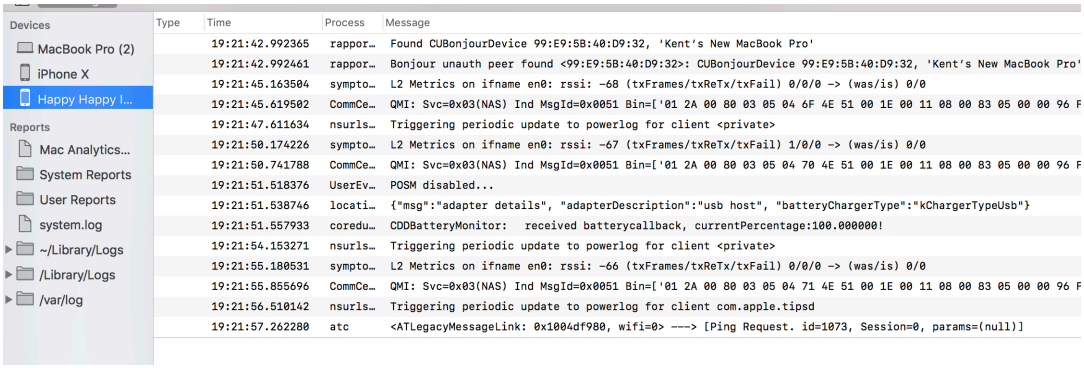


Testing the application:

Let's use this application called Pusher which is the iOS push notifications equivalent of "postman":
https://github.com/noodlewerk/NWPusher

I downloaded the binary (I did this at my own risk — but the tool was recommended by a colleague in our development education group so I figured it was worth the risk.)

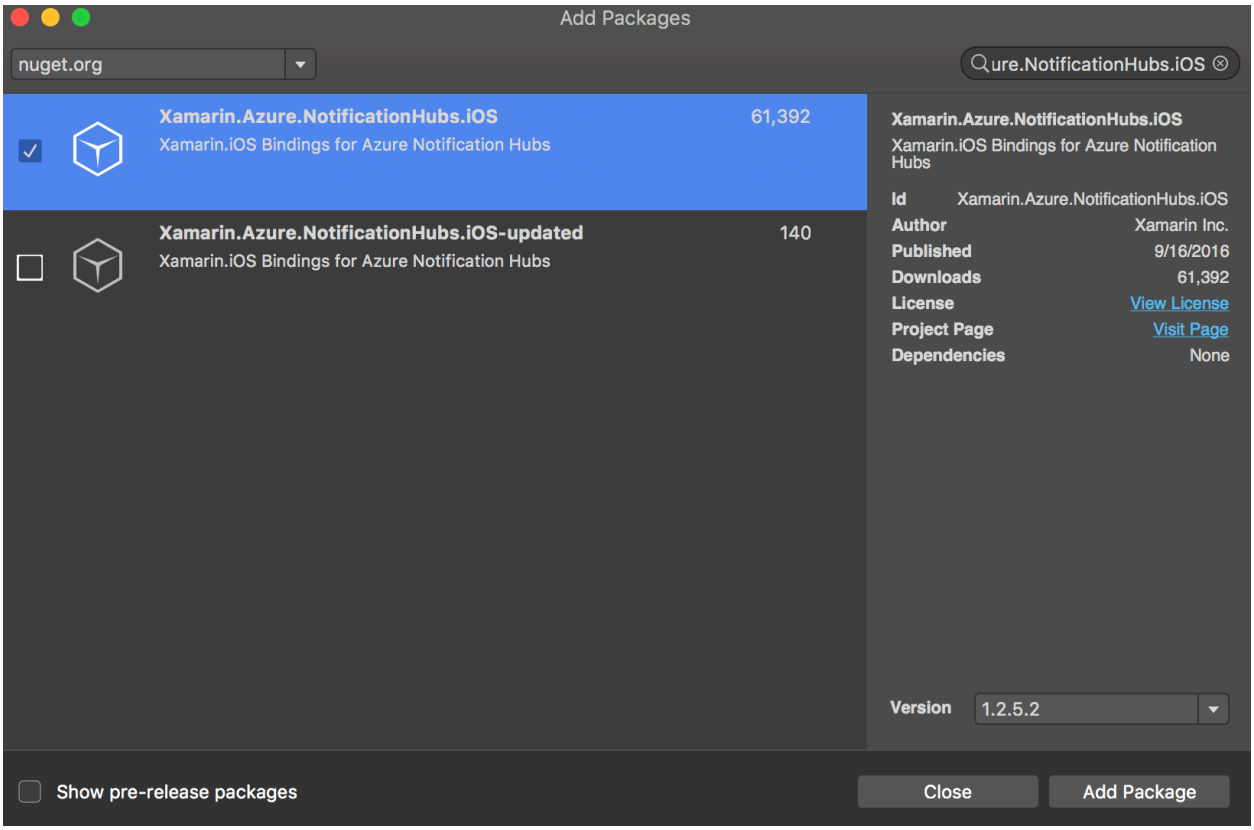
- It will ask you to upload your SSL certificate which is the same as the one you uploaded to Azure Notification Hub.
- In Finder on the Mac - open up console. Command + Space :: Console.
    - Run the app on your phone and then find it in the left bar



- Search for: token received
    - If you don't see it - rebuild your app and make sure your console is running from when your app is open
- 1) Try with the app in the background (ie. press the Home button) - you should be to press the notification and see the app open up.
- 2) Try with the app in the foreground - the text should change if successful! (The text will change to "Success"
    - NOTE - the button will not work yet.
- **Quick gotcha - if you delete the app, you have recheck for the token. Also if you recheck the token you might see old logs so make sure you're looking at the latest ones (ie check the time stamp).

Registering Notification through the iOS Client

Add the necessary package to the iOS project:
Xamarin.Azure.NotificationHubs.iOS:



pushsample.iOS > AppDelegate.cs

```
15
16  namespace pushsample.iOS
17  {
18      [Register("AppDelegate")]
19      public partial class AppDelegate : global::Xamarin.Forms.Platform.iOS.FormsApplicationDelegate
20      {
21
22          private SBNotificationHub Hub { get; set; }
23
```

(Right click to add the appropriate using statements.)

Go the shared project to your man app project.

pushsample > pushsample.cs

*Ensure that you use the correct shared access signature configuration strings on the client and on the application back end. Generally, you must use **DefaultListenSharedAccessSignature** on the client and **DefaultFullSharedAccessSignature** on the application back end (grants permissions to send notifications to Notification Hubs).*

```
// Azure app-specific connection string and hub path
public const string ConnectionString = "ConnectionString - Microsoft documentation recommends to use the Listen on client ";
public const string NotificationHubName = "<Azure hub name - ie. just the name of the hub>";
```

Get the above from your Notification Hub:

Go to AccessPolicies:

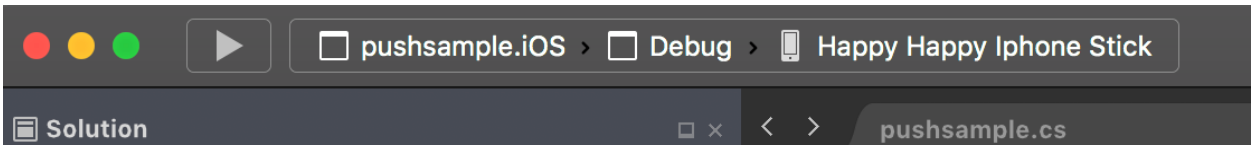
a. Go to Listen, Manage, Send and get the Connection String.
b. For the Notification Hub Name go here:

and get this:

```
async Task SendRegistrationToServerAsync(NSData deviceToken)
{
    // This is the template/payload used by iOS. It contains the "messageParam"
    const string templateBodyAPNS = "{\"aps\":{\"alert\":\"$(messageParam)\"}}";

    var templates = new JObject();
    templates["genericMessage"] = new JObject
    {
        {"body", templateBodyAPNS}
    };


    Hub = new SBNotificationHub(App.ConnectionString, App.NotificationHubName);

    Hub.UnregisterAllAsync(deviceToken, (error) => {
        if (error != null)
        {
            Console.WriteLine("Error calling Unregister: {0}", error.ToString());
            return;
        }

        NSSet tags = null; // create tags if you want
        Hub.RegisterNativeAsync(deviceToken, tags, (errorCallback) => {
            if (errorCallback != null)
                Console.WriteLine("RegisterNativeAsync error: " + errorCallback.ToString());
        });
    });

    //This will be replaced with a Function calling into a NotificationHub
    //var client = new MobileServiceClient(XamUNotif.App.MobileServiceUrl);
    //await client.GetPush().RegisterAsync(deviceToken, templates);
}
```

Delete previous versions of your app from your device. (This will cause a re-registration of your app and get a new device token for registration with your Notification Hub).

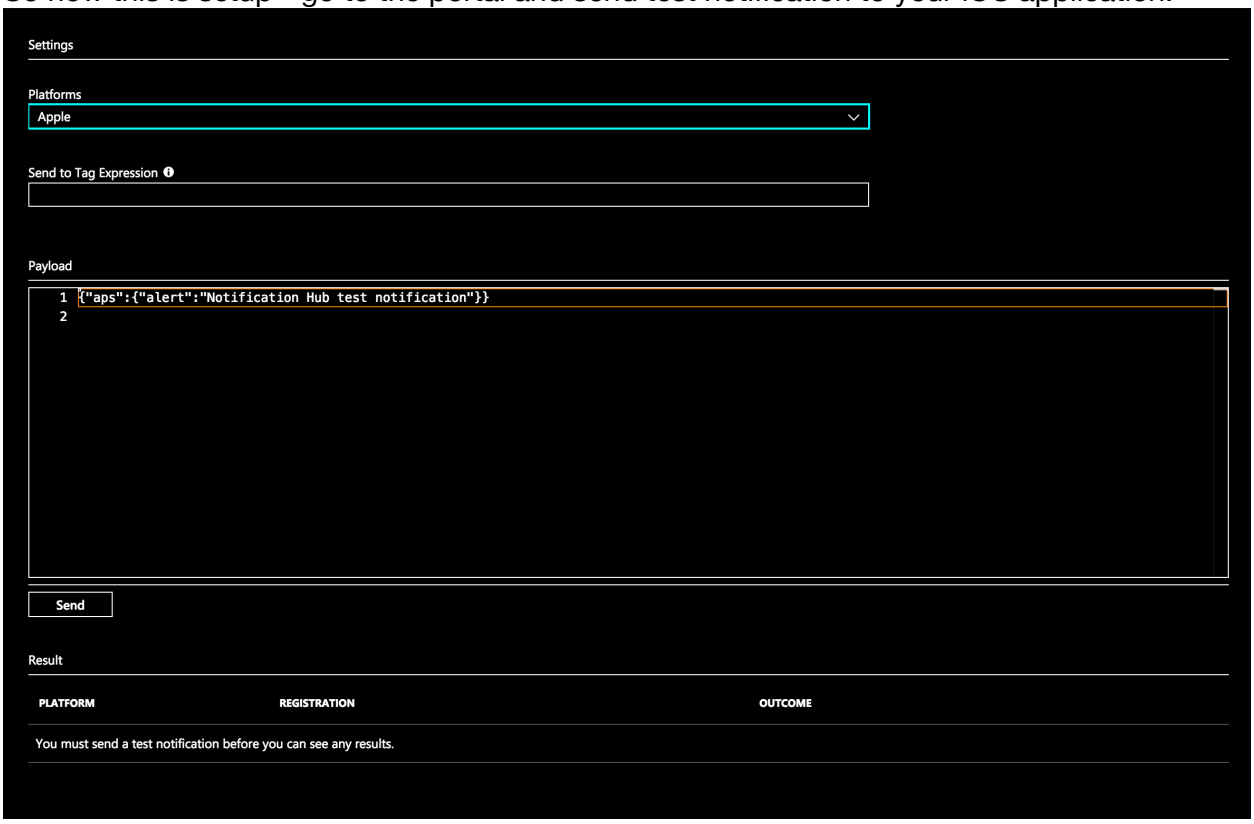
Re-run on device:



Also - added the extra line of Error Handling to make sure the notification's has the string "api" and is not null:
To see what the APS keys are: https://developer.apple.com/library/content/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/PayloadKeyReference.html

```
        void PresentNotification(NSDictionary dict)
        {
            // Check to see if the dictionary has the aps key.  This is the notification payload you would have sent
            if (null != dict && dict.ContainsKey(new NSString("aps")))
            {
                // Extract some data from the notifiation and display it using an alert view.
                NSDictionary aps = dict.ObjectForKey(new NSString("aps")) as NSDictionary;

                var msg = string.Empty;
                if (aps.ContainsKey(new NSString("alert")))
                {
                    msg = (aps[new NSString("alert")] as NSString).ToString();
                }

                if (string.IsNullOrEmpty(msg))
                {
                    msg = "(unable to parse)";
                }

                MessagingCenter.Send<object, string>(this, App.NotificationReceivedKey, msg);
            }
        }
```

So now this is setup - go to the portal and send test notification to your iOS application:



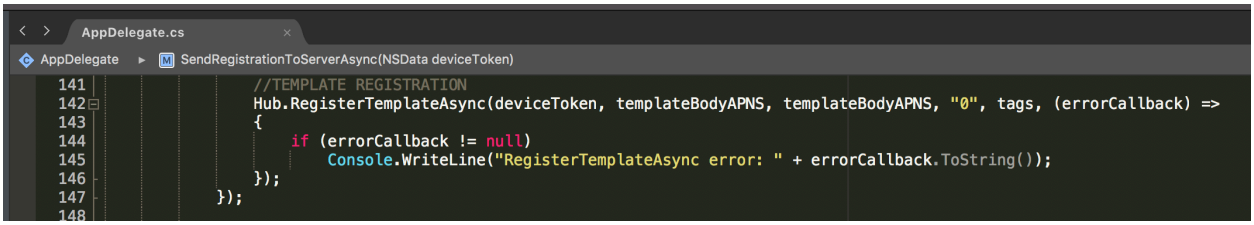
Press Send.

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

BRANCH 3

We've added a TEMPLATE REGISTRATION in the AppDelegate.s of the of the iOS project
This can be distinguished from a NATIVE registration.

NEW:

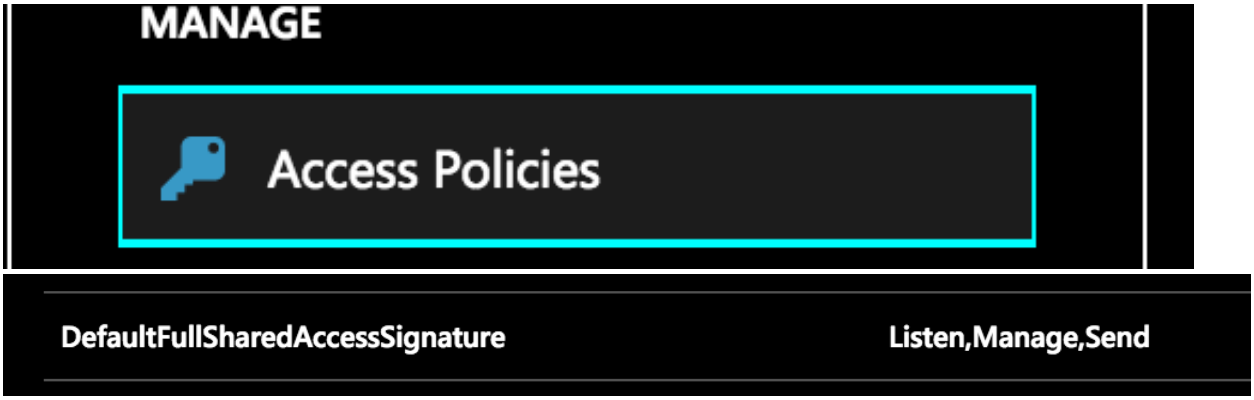


OLD:

Not sure which one you've done?

(Helpful links:
https://stackoverflow.com/questions/42980577/azure-notification-hub-what-are-registration-types-native-and-template
https://stackoverflow.com/questions/24543173/diagnosing-dropped-notifications-from-azure-notification-hub-to-apns
https://docs.microsoft.com/en-us/azure/notification-hubs/notification-hubs-push-notification-fixer#verify-registrations
** Visually look at the registrations
https://msdn.microsoft.com/en-us/magazine/dn948105.aspx
https://docs.microsoft.com/en-us/azure/notification-hubs/notification-hubs-push-notification-fixer

//Running the Console application
// Make sure to add in Azure Constants the necessary constants:
1) Full Access Connection String:



2) ConsoleApplicationNotificationHubName

- This is just the name of the notification hub

//In case you have issues running the Console application -- please set it as the Startup Project — I run this console application from Visual Studio 2017 - targeting .NET CORE 2.0
//NOTE: Notice the “Thread.Sleep(30000)” — you’ll need something like this to make sure the program has enough time to send the notification.

You’ll notice there are three types of Notifications that can be sent:
1) Send Template Notification Async Native Apple — this you’ll need to send with the Native registration set up in the iOS App
2) Send Template Notification Async - this you’ll need to send with the Template registration set up in the iOS App
3) Send Template Notification Multiple Async - this you’ll need to send with the Template registration set up in the iOS App