Handling Push Notifications in Windows 8.1 Universal and Windows 10 UWP

Overview

MobileFirst-provided Notifications API can be used in order to register & unregister devices, and subscribe & unsubscribe to tags. In this tutorial, you will learn how to handle push notification in Native Windows 8 and Windows 10 UWP applications using Csharp.

Prerequisites:

- Make sure you have read the following tutorials:
  - Push Notifications Overview (../push-notifications-overview)
  - Setting up your MobileFirst development environment (../../setting-up-your-development-environment/)
  - Adding the MobileFirst Platform Foundation SDK to windows applications (../../adding-the-mfpf-sdk/windows-8-10)
- MobileFirst Server to run locally, or a remotely running MobileFirst Server.
- MobileFirst CLI installed on the developer workstation

Jump to:

Notifications Configuration

Create a new Visual Studio project or use and existing one.
If the MobileFirst Native Windows SDK is not already present in the project, follow the instructions in the Adding the MobileFirst Foundation SDK to iOS applications (../../adding-the-mfpf-sdk/windows-8-10) tutorial.

Adding the Push SDK

1. Select Tools > NuGet Package Manager > Package Manager Console.
2. Choose the project where you want to install the MobileFirst push component.
3. Add theMobileFirst push SDK by running the **Install-Package IBM.MobileFirstPlatformFoundationPush** command.

Notifications API

MFPPush Instance

All API calls must be called on an instance of `MFPPush`. This can be by created as a `var` in a view controller such as `var push = MFPPush.sharedInstance();`, and then calling `push.methodName()` throughout the view controller.

Alternatively you can call `MFPPush.sharedInstance().methodName()` for each instance in which you need to access the push API methods.

Challenge Handlers

If the `push.mobileclient` scope is mapped to a **security check**, you need to make sure matching **challenge handlers** exist and are registered before using any of the Push APIs.

> Learn more about challenge handlers in the credential validation (../../authentication-and-security/credentials-validation/ios) tutorial.

Client-side

C Sharp Methods	Description
`Initialize()`	Initializes MFPPush for supplied context.
`IsPushSupported()`	Does the device support push notifications.
`Task <MFPPushMessageResponse> RegisterDevice(JObject options)`	Registers the device with the Push Notifications Service.
`Task <MFPPushMessageResponse> GetTags()`	Retrieves the tag(s) available in a push notification service instance.
`Task <MFPPushMessageResponse> Subscribe(String[] Tags)`	Subscribes the device to the specified tag(s).
`Task <MFPPushMessageResponse> GetSubscriptions()`	Retrieves all tags the device is currently subscribed to.
`Task <MFPPushMessageResponse> Unsubscribe(String[] Tags)`	Unsubscribes from a particular tag(s).
`Task <MFPPushMessageResponse> UnregisterDevice()`	Unregisters the device from the Push Notifications Service

Initialization

Initialization is required for the client application to connect to MFPPush service.

- The `Initialize` method should be called first before using any other MFPPush APIs.
- It registers the callback function to handle received push notifications.

```
PushClient.Initialize();
```

Is push supported

Checks if the device supports push notifications.

```
Boolean isSupported = MFPPush.GetInstance().IsPushSupported();

if (isSupported ) {
   // Push is supported
} else {
   // Push is not supported
}
```

Register device & send device token

Register the device to the push notifications service.

```
JObject Options = new JObject();
MFPPushMessageResponse Response = await MFPPush.GetInstance().RegisterDevice(Options);
if (Response.Success == true)
{
   // Successfully registered
} else {
   // Registration failed with error
}
```

Get tags

Retrieve all the available tags from the push notification service.

```
MFPPushMessageResponse Response = await MFPPush.GetInstance().GetTags();
if (Response.Success == true)
{
   Message = new MessageDialog("Avalibale Tags: " + Response.ResponseJSON["tagNames"]);
} else{
   Message = new MessageDialog("Failed to get Tags list");
}
```

Subscribe

Subscribe to desired tags.

```
string[] Tags = ["Tag1" , "Tag2"];

// Get subscription tag
MFPPushMessageResponse Response = await MFPPush.GetInstance().Subscribe(Tags);
if (Response.Success == true)
{
   //successfully subscribed to push tag
}
else
{
   //failed to subscribe to push tags
}
```

Get subscriptions

Retrieve tags the device is currently subscribed to.

```
MFPPushMessageResponse Response = await MFPPush.GetInstance().GetSubscriptions();
if (Response.Success == true)
{
   Message = new MessageDialog("Avalibale Tags: " + Response.ResponseJSON["tagNames"]);
}
else
{
   Message = new MessageDialog("Failed to get subcription list...");
}
```

Unsubscribe

Unsubscribe from tags.

```
string[] Tags = ["Tag1" , "Tag2"];

// unsubscribe tag
MFPPushMessageResponse Response = await MFPPush.GetInstance().Unsubscribe(Tags);
if (Response.Success == true)
{
   //succes
}
else
{
   //failed to subscribe to tags
}
```

Unregister

Unregister the device from push notification service instance.

```
MFPPushMessageResponse Response = await MFPPush.GetInstance().UnregisterDevice();
if (Response.Success == true)
{
   // Successfully registered
} else {
   // Registration failed with error
}
```

Handling a push notification

In order to handle a push notification you will need to set up a `MFPPushNotificationListener`. This can be achieved by implementing the following method.

1. Create a class by using interface of type MFPPushNotificationListener

```
internal class NotificationListner : MFPPushNotificationListener
{
   public async void onReceive(String properties, String payload)
{
  // Handle push notification here
}
}
```

1. Set the class to be the listener by calling `MFPPush.getInstance().listen(new NotificationListner())`
2. In the onReceive method you will receive the push notification and can handle the notification for the desired behavior.

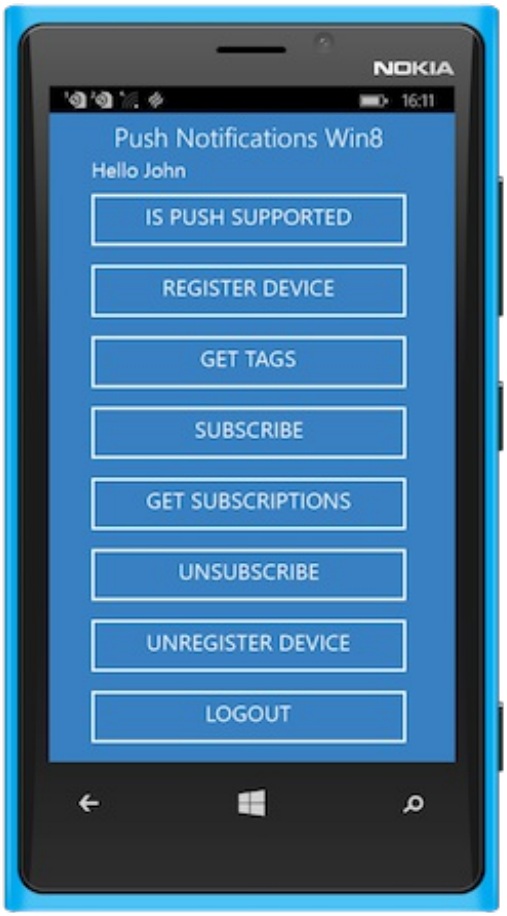
Windows Universal Push Notifications Service

No specific port needs to be open in your server configuration.

WNS uses regular http or https requests.

Sample application

Click to download



(https://github.com/MobileFirst-Platform-Developer-Center/PushNotificationsWin8/tree/release80) the Windows 8.1 Universal project.
Click to download (https://github.com/MobileFirst-Platform-Developer-Center/PushNotificationsWin8/tree/release80) Windows 10 UWP project.

Sample usage

1. Import the project to Visual Studio.
2. Configure the project with your bundleId (based on bundleId that you have created for your push notifications certificate .p12 file).
3. From a **Command-line** window, navigate to the project's root folder and run the command: `mfpdev app register`.
4. Perform the required scope mapping for **push.mobileclient**.
5. Run the app by clicking the **Run** button.

Sending a notification (../sending-push-notifications)

- Tag notification
  - Use the **MobileFirst Operations Console → [your application] → Push → Send Push tab** .
- Authenticated notification:
  - Deploy the **UserLogin** sample Security Check (../../authentication-and-security/user-authentication/security-check).
  - In **MobileFirst Operations Console → [your application] → Security tab** , map the

push.mobileclient scope to the **UserLogin** Security Check.

- Follow the instructions for REST APIs (../sending-push-notifications#rest-apis) to send the notification.