Handling Push Notifications in Cordova applications

Overview

Before Cordova applications are able to handle any received push notifications, they must be configured for the appropriate platform. Once an application has been configured, MobileFirst-provided Notifications API can be used in order to register & unregister devices, and subscribe & unsubscribe to tags.

In this tutorial you learn how to configure a Cordova application and how to use the MobileFirst-provided Notifications API.

Prequisites:

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

Jump to

Notifications Configuration

Create a new Cordova project or use an existing one.
If the MobileFirst Cordova SDK is not already present in the project, follow the instructions in the Adding the MobileFirst Platform Foundation SDK to Cordova applications (../../../adding-the-mfpf-sdk/cordova) tutorial.

Adding the Push plug-in

1. Whether using iOS or Android, open a **command-line** window and navigate to the root of the Cordova project.
   Add the MobileFirst Push plug-in by running the command:

   ```
   cordova plugin add cordova-plugin-mfp-push
   ```

2. Run the command: `$ cordova build`

Android platform

1. Open the project in Android Studio by importing the **[project root]/platforms/android** folder.
2. In **Android → Gradle scripts**, select the **build.gradle (Module: Android)** file and the following line to `dependencies`:

   ```
   classpath 'com.google.gms:google-services:2.0.0-alpha3'
   ```

3. Navigate to **[project root]platforms/android/cordova-plugin-mfp-push/** and select the **-build-extras.gradle** file.
   - Change compile version to `8.0.0-Beta1-SNAPSHOT` in `dependencies`:

     ```
     compile group: 'com.ibm.mobile.foundation',
       name:'ibmmobilefirstplatformfoundationpush',
       version:'8.0.0-Beta1-SNAPSHOT',
       ext: 'aar',
       transitive: true
     ```

   - Add the following to `dependencies`:

     ```
     compile 'com.google.android.gms:play-services-gcm:8.4.0'
     ```

   - Add the following line at the bottom:

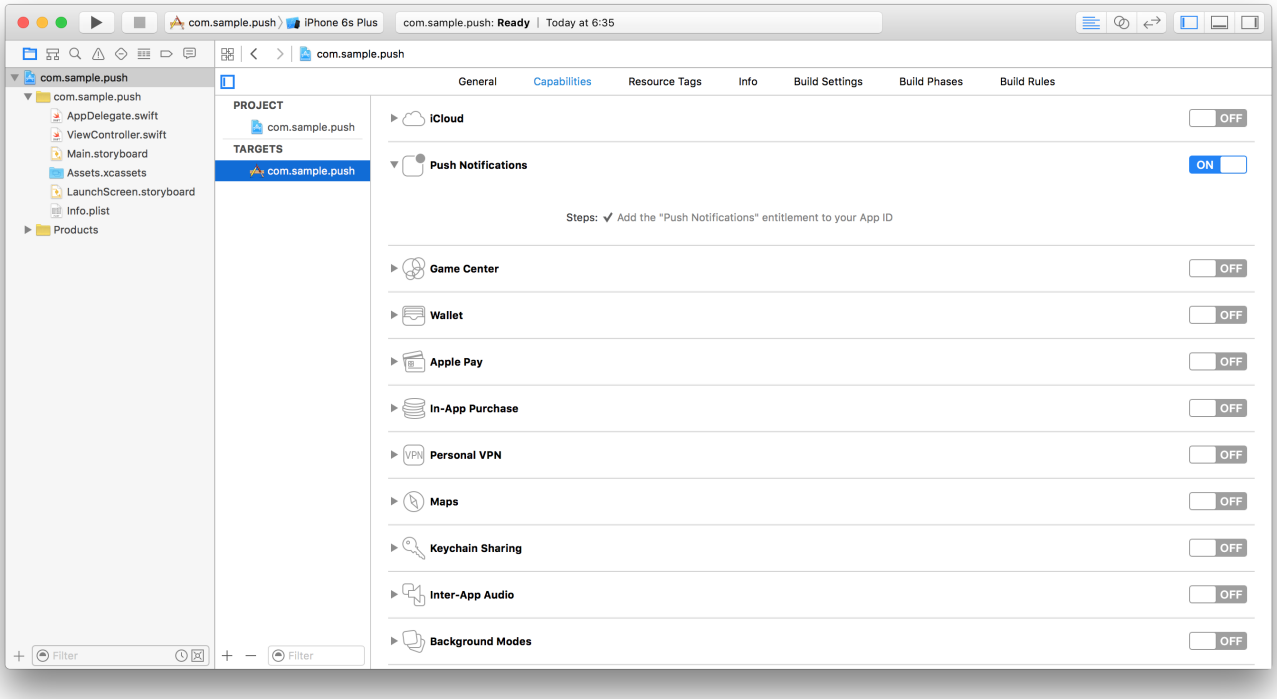
     ```
     apply plugin: 'com.google.gms.google-services'
     ```

4. Add google-services.json configuration file to app/platforms/android folder
5. Change version to `8.0.0-Beta1-SNAPSHOT` in app/platforms/android folder

iOS platform

1. Open the project in Xcode by importing the **[project root]/platforms/ios** folder.
2. Enable push notifications for your application in the **Capabilities** screen.

> **Important:** the bundleId selected for the application must match the AppId that you have previously created in the Apple Developer site. See the [Push Notifications Overview] tutorial.



Notifications API

Client-side API for tag notifications

`MFPPush.isPushSupported()` - Returns `true` if push notifications are supported by the platform, or `false` otherwise.\

```
MFPPush.isPushSupported(function(successResponse) {
    alert("Push Supported: " + successResponse);
}, function(failureResponse) {
    alert("Failed to get push support status");
    }
);
```

`MFPPush.registerDevice(options)` - Registers devices for push notifications (?!? confirm definition ?!?)

```
MFPPush.registerDevice(settings, function(successResponse) {
    alert("Successfully registered");
    enableButtons();
}, function(failureResponse) {
    alert("Failed to register");
    }
);
```

`MFPPush.getTags()` - Gets all tags available (?!? confirm definition?!?)

```
MFPPush.getTags(function(tags) {
    alert(JSON.stringify(tags));
}, function(){
    alert("Failed to get tags");
    }
);
```

`MFPPush.subscribe(tagName,options)` - Subscribes the device to the specified tag name.

```
var tags = ['sample-tag1','sample-tag2']
MFPPush.subscribe(tags,function(tags) {
    alert("Subscribed successfully");
}, function(){
    alert("Failed to subscribe");
);
```

`MFPPush.getSubsciptions(options)` - Get tags the device is subscribed to.

```
MFPPush.getSubscriptions(function(subscriptions) {
    alert(JSON.stringify(subscriptions));
}, function(){
    alert("Failed to get subscriptions");
    }
);
```

`MFPPush.unsubscribe(tagName,options)` - Unsubscribes the device to the specified tag name.

```
var tags = ['sample-tag1','sample-tag2']
MFPPush.unsubscribe(tags, function(tags) {
    alert("Unsubscribed successfully");
}, function(){
    alert("Failed to unsubscribe");
    }
);
```

`MFPPush.unregisterDevice(options)` - Unregisters device for push notifications.

```
MFPPush.unregisterDevice(function(successResponse) {
    alert("Unregistered successfully");
}, function(){
    alert("Failed to unregister");
    }
);
```

Handling a push notification

You can handle a push notification by taking message received and creating an alert.

```
var notificationReceived = function(message) {
    alert(JSON.stringify(message));
};
```

Handling a secure push notification