Handling Push Notifications in Cordova applications

fork and edit tutorial (https://github.ibm.com/MFPSamples/DevCenter/tree/master/tutorials/en/foundation/8.0/notifications/handling-push-notifications-in-cordova/index.md) | report issue (https://github.ibm.com/MFPSamples/DevCenter/issues/new)

Overview

Before iOS and Android Cordova applications are able to receive and display push notifications, the Cordova project needs to be configured with the **cordova-plugin-mfp-push** plug-in. Once an application has been configured, MobileFirst-provided Notifications API can be used in order to register & unregister devices, subscribe & unsubscribe tags and handle notifications.

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)
  - 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
  - Android platform
  - iOS platform
- Notifications API
- Handling a push notification

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.

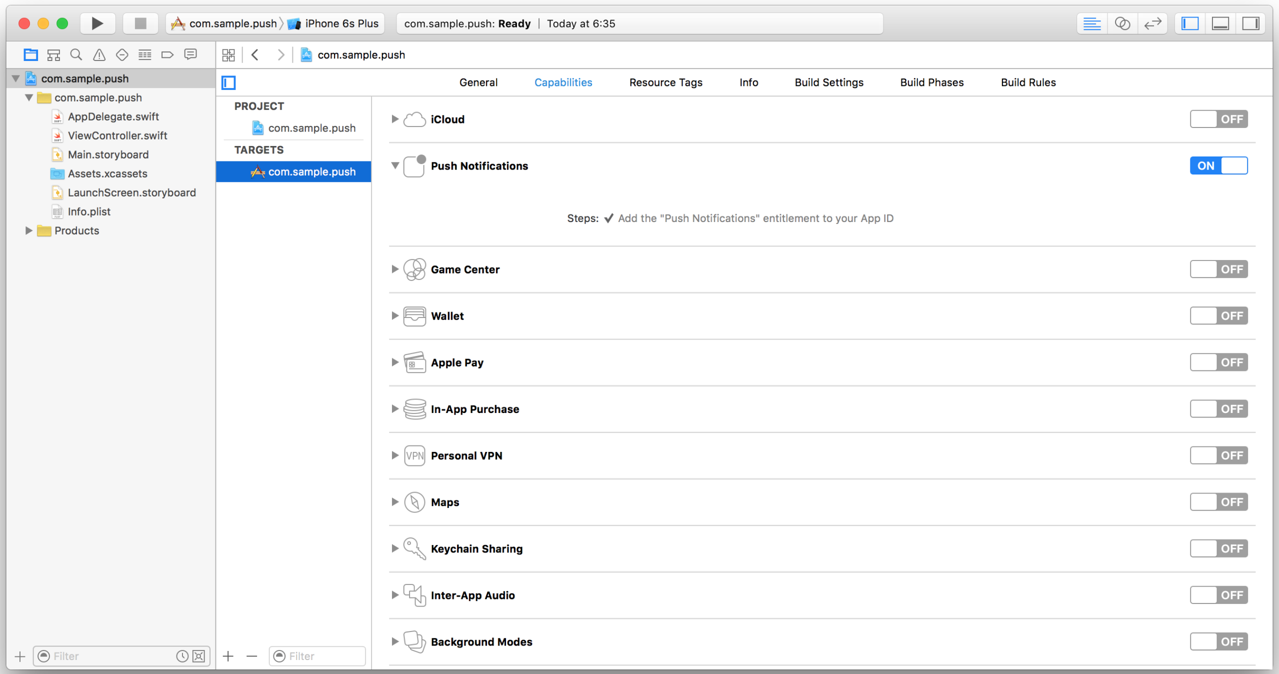
Adding the Push plug-in

1. From a **command-line** window, navigate to the root of the Cordova project.
2. Add the push plug-in to the iOS and/or Android platform by running the command:
   ```
   cordova plugin add cordova-plugin-mfp-push
   ```
3. Build the Cordova project by running the command:
   ```
   cordova build
   ```

iOS platform

The iOS platform requires an additional step.
In Xcode, 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.



iCloud	OFF
Push Notifications	ON
Game Center	OFF
Wallet	OFF
Apple Pay	OFF
In-App Purchase	OFF
Personal VPN	OFF
Maps	OFF
Keychain Sharing	OFF
Inter-App Audio	OFF
Background Modes	OFF

Steps: ✓ Add the "Push Notifications" entitlement to your App ID

Notifications API

Client-side

Javascript Function	Description
`MFPPush.isPushSupported(success,failure)`	Does the device support push notifications.
`MFPPush.registerDevice(success, failure)`	Registers the device with the Push Notifications Service.
`MFPPush.getTags(success, failure)`	Retrieves all the tags available in a push notification service instance.
`MFPPush.subscribe(tag, success, failure)`	Subscribes to a particular tag.
`MFPPush.getSubsciptions(success, failure)`	Retrieves the tags device is currently subscribed to
`MFPPush.unsubscribe(tag, success, failure)`	Unsubscribes from a particular tag.

Javascript Function	Description
`MFPPush.unregisterDevice(success, failure)`	Unregisters the device from the Push Notifications Service

API implementation

- Initialize the **MFPPush** instance.
    - Required for the client application to connect to MFPPush service with the right application context.
    - The API method should be called first before using any other MFPPush APIs.
    - Registers the callback function to handle received push notifications.

```
MFPPush.initialize (
   function(successResponse) {
      alert("Successfully intialized");
      MFPPush.registerNotificationsCallback(notificationReceived);
   },
   function(failureResponse) {
      alert("Failed to initialize");
   }
);
```

- Check if the device supports push notifications.

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

- Register the device to the push notifications service.

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

- Retrieve all the available tags from the push notification service.

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

- Subscribe to desired tags.

```
var tags = ['sample-tag1','sample-tag2'];

MFPPush.subscribe(
   tags,
   function(tags) {
      alert("Subscribed successfully");
   },
   function() {
      alert("Failed to subscribe");
   }
);
```

- Retrieve tags the device is currently subscribed to.

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

- Unsubscribe from tags.

```
var tags = ['sample-tag1','sample-tag2'];

MFPPush.unsubscribe(
   tags,
   function(tags) {
      alert("Unsubscribed successfully");
   },
   function() {
      alert("Failed to unsubscribe");
   }
);
```

- Unregister the device from push notification service instance.

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

Handling a push notification

You can handle a received push notification by operating on its response object.

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