

# 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.

### Prerequisites:

- Make sure you have read the following tutorials:
  - Setting up your MobileFirst development environment ([../setting-up-your-development-environment/index](https://github.ibm.com/MFPSamples/DevCenter/tree/master/tutorials/en/foundation/8.0/notifications/handling-push-notifications-in-cordova/index.md))
  - Push Notifications Overview ([../push-notifications-overview](https://github.ibm.com/MFPSamples/DevCenter/tree/master/tutorials/en/foundation/8.0/notifications/handling-push-notifications-in-cordova/index.md))
- 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
- Handling a secure 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](https://github.ibm.com/MFPSamples/DevCenter/tree/master/tutorials/en/foundation/8.0/notifications/handling-push-notifications-in-cordova/index.md)) 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.



## 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.getSubscriptions(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