

Handling Push Notifications in iOS

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 iOS applications using Swift.

For information about Silent or Interactive notifications, see:

- Silent notifications ([../silent](#))
- Interactive notifications ([../interactive](#))

Prerequisites:

- Make sure you have read the following tutorials:
 - Push Notifications Overview ([../..](#))
 - Setting up your MobileFirst development environment ([../../installation-configuration/#installing-a-development-environment](#))
 - Adding the MobileFirst Foundation SDK to iOS applications ([../../application-development/sdk/ios](#))
- MobileFirst Server to run locally, or a remotely running MobileFirst Server.
- MobileFirst CLI installed on the developer workstation

Jump to:

- Notifications configuration
- Notifications API
- Handling a push notification

Notifications Configuration

Create a new Xcode project or use an existing one. If the MobileFirst Native iOS SDK is not already present in the project, follow the instructions in the [Adding the MobileFirst Foundation SDK to iOS applications](#) ([../../application-development/sdk/ios](#)) tutorial.

Adding the Push SDK

1. Open the project's existing **podfile** and add the following lines:

```

use_frameworks!

platform :ios, 8.0
target "Xcode-project-target" do
  pod 'IBMMobileFirstPlatformFoundation'
  pod 'IBMMobileFirstPlatformFoundationPush'
end

post_install do |installer|
  workDir = Dir.pwd

  installer.pods_project.targets.each do |target|
    debugXcconfigFilename = "#{workDir}/Pods/Target Support Files/#{target}/#{target}.debug.xcc
onfig"
    xcconfig = File.read(debugXcconfigFilename)
    newXcconfig = xcconfig.gsub(/HEADER_SEARCH_PATHS = .*/, "HEADER_SEARCH_PATH
S = ")
    File.open(debugXcconfigFilename, "w") { |file| file << newXcconfig }

    releaseXcconfigFilename = "#{workDir}/Pods/Target Support Files/#{target}/#{target}.release.x
cconfig"
    xcconfig = File.read(releaseXcconfigFilename)
    newXcconfig = xcconfig.gsub(/HEADER_SEARCH_PATHS = .*/, "HEADER_SEARCH_PATH
S = ")
    File.open(releaseXcconfigFilename, "w") { |file| file << newXcconfig }
  end
end

```

- Replace **Xcode-project-target** with the name of your Xcode project's target.
2. Save and close the **podfile**.
 3. From a **Command-line** window, navigate into to the project's root folder.
 4. Run the command `pod install`
 5. Open project using the **.xcworkspace** file.

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](https://firebase.google.com/docs/authentication-and-security/credentials-validation/ios)) tutorial.

Client-side

Swift Methods	Description
<code>initialize()</code>	Initializes MFPPush for supplied context.
<code>isPushSupported()</code>	Does the device support push notifications.
<code>registerDevice(completionHandler: ((WLResponse!, NSError!) -> Void)!!)</code>	Registers the device with the Push Notifications Service.
<code>sendDeviceToken(deviceToken: NSData!)</code>	Sends the device token to the server
<code>getTags(completionHandler: ((WLResponse!, NSError!) -> Void)!!)</code>	Retrieves the tag(s) available in a push notification service instance.
<code>subscribe(tagsArray: [AnyObject], completionHandler: ((WLResponse!, NSError!) -> Void)!!)</code>	Subscribes the device to the specified tag(s).
<code>getSubscriptions(completionHandler: ((WLResponse!, NSError!) -> Void)!!)</code>	Retrieves all tags the device is currently subscribed to.
<code>unsubscribe(tagsArray: [AnyObject], completionHandler: ((WLResponse!, NSError!) -> Void)!!)</code>	Unsubscribes from a particular tag(s).
<code>unregisterDevice(completionHandler: ((WLResponse!, NSError!) -> Void)!!)</code>	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.

```
MFPPush.sharedInstance().initialize();
```

Is push supported

Checks if the device supports push notifications.

```
let isPushSupported: Bool = MFPPush.sharedInstance().isPushSupported()

if isPushSupported {
    // Push is supported
} else {
    // Push is not supported
}
```

Register device & send device token

Register the device to the push notifications service.

```

MFPPush.sharedInstance().registerDevice({(options, response: WLResponse!, error: NSError!) -> Void in
    if error == nil {
        // Successfully registered
    } else {
        // Registration failed with error
    }
})

```

Notes: `options` = `[NSObject : AnyObject]` which is an optional parameter that is a dictionary of options to be passed with your register request.

Sends the device token to the server to register the device with its unique identifier.

```

MFPPush.sharedInstance().sendDeviceToken(deviceToken)

```

Note: This is typically called in the **AppDelegate** in the `didRegisterForRemoteNotificationsWithDeviceToken` method

Get tags

Retrieve all the available tags from the push notification service.

```

MFPPush.sharedInstance().getTags({(response: WLResponse!, error: NSError!) -> Void in
    if error == nil {
        print("The response is: \(response)")
        print("The response text is \(response.responseText)")
        if response.availableTags().isEmpty == true {
            // Successfully retrieved tags as list of strings
        } else {
            // Successfully retrieved response from server but there where no available tags
        }
    } else {
        // Failed to receive tags with error
    }
})

```

Subscribe

Subscribe to desired tags.

```

var tagsArray: [AnyObject] = ["Tag 1" as AnyObject, "Tag 2" as AnyObject]

MFPPush.sharedInstance().subscribe(self.tagsArray, completionHandler: {(response: WLResponse!, error: NSError!) -> Void in
    if error == nil {
        // Subscribed successfully
    } else {
        // Failed to subscribe with error
    }
})

```

Get subscriptions

Retrieve tags the device is currently subscribed to.

```

MFPPush.sharedInstance().getSubscriptions({(response: WLResponse!, error: NSError!) -> Void in
    if error == nil {
        // Successfully received subscriptions as list of strings
    } else {
        // Failed to retrieve subscriptions with error
    }
})

```

Unsubscribe

Unsubscribe from tags.

```

var tags: [String] = {"Tag 1", "Tag 2"};

// Unsubscribe from tags
MFPPush.sharedInstance().unsubscribe(tags, completionHandler: {(response: WLResponse!, error: NSError!) -> Void in
    if error == nil {
        // Unsubscribed successfully
    } else {
        // Failed to unsubscribe
    }
})

```

Unregister

Unregister the device from push notification service instance.

```

MFPPush.sharedInstance().unregisterDevice({(response: WLResponse!, error: NSError!) -> Void in
    if error == nil {
        // Unregistered successfully
    } else {
        self.showAlert("Error \ \(error.description)")
        // Failed to unregister with error
    }
})

```

Handling a push notification

Push notifications are handled by the native iOS framework directly. Depending on your application lifecycle, different methods will be called by the iOS framework.

For example if a simple notification is received while the application is running, **AppDelegate's** `didReceiveRemoteNotification` will be triggered:

```
func application(application: UIApplication, didReceiveRemoteNotification userInfo: [NSObject : AnyObject]) {  
    print("Received Notification in didReceiveRemoteNotification \ \(userInfo)")  
  
    // display the alert body  
    if let notification = userInfo["aps"] as? NSDictionary,  
        let alert = notification["alert"] as? NSDictionary,  
        let body = alert["body"] as? String {  
        showAlert(body)  
    }  
}
```

Learn more about handling notifications in iOS from the Apple documentation:
<http://bit.ly/1ESSGdQ>

Sample application

Click to download (<https://github.com/MobileFirst-Platform-Developer-Center/PushNotificationsSwift/tree/release80>) the Xcode project.

Sample usage

Follow the sample's README.md file for instructions.

Last modified on

