Push notifications in native iOS applications

Overview

IBM MobileFirst Platform Foundation provides a unified set of API methods to send, or push, notifications to devices where the MobileFirst application is installed in. It is possible to send a notification in 3 distinct types: event source notifications, broadcast notifications and tag notifications.

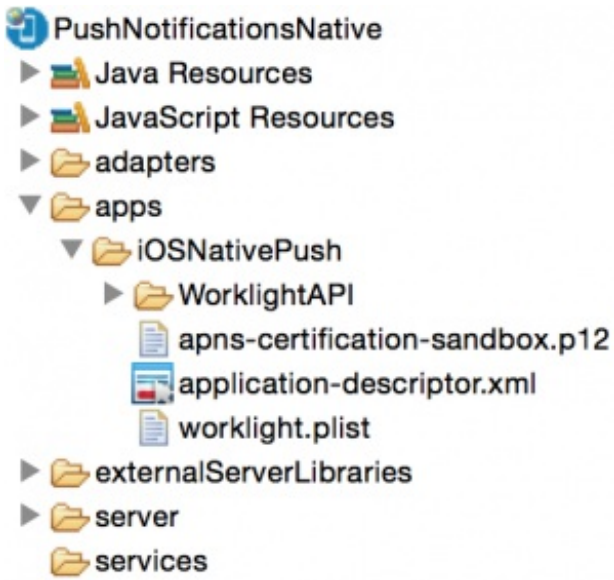
In this tutorial, the concept, API and usage of push notifications will be explained in the context of Native iOS applications.
To create and configure an iOS native project, first follow these tutorials:

- Creating your first Native iOS MobileFirst application (../../hello-world/configuring-a-native-ios-with-the-mfp-sdk/)
- Invoking adapter procedures from native iOS applications (../../server-side-development/invoking-adapter-procedures-native-ios-applications/)

The following topics are covered:

- Setting-up push notification
- Notification types
- Select a notification type

Setting up push notifications



1. Create a MobileFirst project and add a MobileFirst iOS Native API.
2. Add the Apple Push Notification Service (APNS) p12 keys to the root folder of the application (either apns-certificate-sandbox.p12 or apns-certificate-production.p12).
   apns-certificate-sandbox.p12 is used in development mode. When you move to production, use apns-certificate-production.p12.

3. In **application-descriptor.xml**, add the `pushSender` tag with the `password` attribute. Use the

.p12 keystore as the password value.

For example :

```
<nativeIOSApp xmlns="http://www.worklight.com/native-ios-descriptor" bundleId="com.iOSNativePush" id="iOSNativePush" platformVersion="6.3.0.00.20141105-0943" securityTest="MySecurityTest" version="1.0">
    <displayName>iOSNativePush</displayName>
    <description>iOSNativePush</description>
    <pushSender password="replace-with-certificate-password"/>
</nativeIOSApp>
```

4. Deploy the MobileFirst native API. In MobileFirst Studio, right-click the native API and select **"Run As > Deploy Native API"**. With the CLI, use `mfp deploy` from within the application folder.

Notification types

Event source notifications

Event source notifications are notification messages that are targeted to devices with a user subscription.

Broadcast notifications

Broadcast notifications are notification messages that are targeted to all subscribed devices.

Tag notifications

Tag notifications are notification messages that are targeted to all subscribed devices to a particular tag.

For more information, select a notification type.

Additional iOS-specific notification types

Silent notifications

iOS 7 and above.

Silent notifications is a feature allowing to send notifications without disturbing the user. Notifications are not shown in the notification center or notification bar.
Callback methods are executed even when the application is running in the background.

> For more information, refer to the "silent notifications" topics in the MobileFirst Platform user documentation, and in Apple's user documentation.

Server API for silent notification

To implement silent notification in the case of event source/broadcast/tag-based notifications, create a notification object by using the `WL.Server.createDefaultNotification` API and set the type as below:

```
notification.APNS.type = "DEFAULT" | "SILENT" | "MIXED";
```

- `DEFAULT` means normal notification, which shows the alert and is kept in the notification center if the application is running in the background.

- `SILENT` means silent notification, which does not show any alert or the message is not placed in the notification center. In the silent type, the aps tag of push notification contains only content-available.
- `MIXED` means a combination of the above: This option invokes the callback method silently and shows the alert.

Client-side API for silent notification

To handle silent notification on the client-side:

1. Enable the application capability to perform background tasks on receiving the remote notifications
   To enable background processing, select the project in XCode and in the capabilities tab, select the appropriate background modes, like Remote notifications and Background fetch.
2. Implement a new callback method in the `AppDelegate` `(application: didReceiveRemoteNotification:fetchCompletionHandler:)` to receive silent notifications when the application is running in the background.
3. In the callback, check whether the notification is silent by checking that the key content-available is set to 1.
4. Call the `fetchCompletionHandler` block method at the end of the notification handler.

Interactive notifications

iOS 8 and above.

Interactive notification enables users to take actions when a notification is received without the application being open.
When an interactive notification is received, the device shows action buttons along with the notification message.

> For more information, refer to the "interactive notifications" topics in the MobileFirst Platform user documentation, and in Apple's user documentation.

Server API for interactive notification

To send interactive notification, set a string to indicate the category.
Categories describe a custom type of notification that your application sends and contains actions that a user can perform in response.

- For event-source notifications, create a notification object and set type as below:
  `notification.APNS.category = “poll";`
- For broadcast/tag-based notifications, create a notification object and set the type as below:
  `notification.settings.apns.category = “poll";`
- The category name must be same as the one used on the client side.

Client-side steps for interactive notification

On the client side, to handle interactive notification:

- Enable the application capability to perform background tasks on receiving the remote notifications.
  This step is required if some of the actions are background-enabled.
  To enable background processing, select the project in XCode and in the capabilities tab, select the appropriate background modes like Remote notifications and Background fetch.

- Set categories before setting `deviceToken` on `WLPush` object in `(application: didRegisterForRemoteNotificationsWithDeviceTokenapplication:)` method in `AppDelegate` class.

```
if([application respondsToSelector:@selector(registerUserNotificationSettings:)]){
  UIUserNotificationType userNotificationTypes = UIUserNotificationTypeNone | UIUserNotificati
onTypeSound | UIUserNotificationTypeAlert | UIUserNotificationTypeBadge;
  UIMutableUserNotificationAction *acceptAction = [[UIMutableUserNotificationAction alloc] init];
  acceptAction.identifier = @"OK";
  acceptAction.title = @"OK";
  UIMutableUserNotificationAction *rejetAction = [[UIMutableUserNotificationAction alloc] init];
  rejetAction.identifier = @"NOK";
  rejetAction.title = @"NOK";
  UIMutableUserNotificationCategory *cateogory = [[UIMutableUserNotificationCategory alloc] ini
t];
  cateogory.identifier = @"poll";
  [cateogory setActions:@[acceptAction,rejetAction] forContext:UIUserNotificationActionContext
Default];
  [cateogory setActions:@[acceptAction,rejetAction] forContext:UIUserNotificationActionContext
Minimal];
  NSSet *catgories = [NSSet setWithObject:cateogory];
  [application registerUserNotificationSettings:[UIUserNotificationSettings settingsForTypes:user
NotificationTypes categories:catgories]];
}
```

- Implement the new callback method:

```
(void)application:(UIApplication *)application handleActionWithIdentifier:(NSString *)identifier for
RemoteNotification:(NSDictionary *)userInfo completionHandler:(void (^)())completionHandler
```

  This new callback method is invoked when the user clicks the action button.

  The implementation of this method must perform the action that is associated with the specified identifier and execute the block in the `completionHandler` parameter.