Push notifications in native iOS applications

Overview

This tutorial explains the concept, API, and usage of push notifications in the context of native iOS applications.

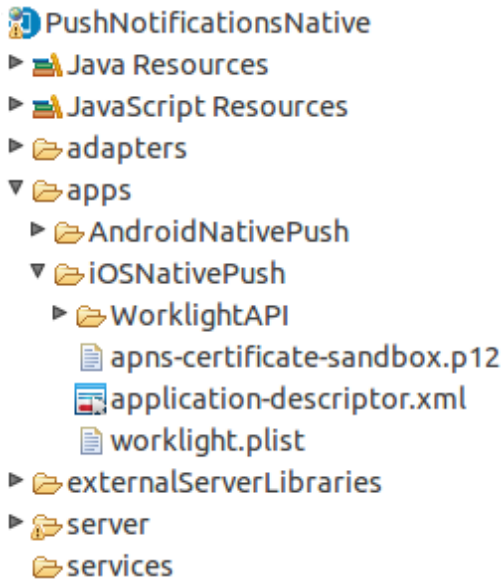
The following topics are covered:

- Setting up for push notification
- Server-side notification APIs
- Client-side notification APIs
- Tag-based notification
- Broadcast-based notification
- Silent notification
- Interactive notification
- Running the sample

To create and configure an iOS native project, first follow the "Creating your first Native iOS MobileFirst application (../../hello-world/creating-first-native-ios-mobilefirst-application/)" and "Invoking adapter procedures from native iOS applications (../../server-side-development/invoking-adapter-procedures-native-ios-applications/)" tutorials.

To learn more about the architecture and terminology of MobileFirst push notifications, refer to the "Push notifications in hybrid applications (../../notifications/push-notifications-hybrid-applications/)" tutorial.

Setting up your native iOS application for push notification



1. In MobileFirst Studio, 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="apns-certificate-p12 password"/>
   </nativeIOSApp>
   ```

4. To deploy the MobileFirst native API, right-click the native API and select **"Run As > Deploy Native API"**.

Notification

While the user subscription exists, MobileFirst Server can produce push notifications for the subscribed user. These notifications can be delivered by the adapter code to all or some of the devices from which the user subscribed.

Implementation of the push notification API consists of the following main steps:

On the server side:

Creating an event source
Sending notification

On the client side:

Sending the token and initializing the `WLPush` *class*
Registering the event source
Subscribing to/unsubscribing from the event source

Notification API: Server side

Creating an event source

This can be achieved by creating a notification event source in the adapter JavaScript™ code at a global level (outside any JavaScript function).

```
WL.Server.createEventSource({
   name: 'PushEventSource',
   onDeviceSubscribe: 'deviceSubscribeFunc',
   onDeviceUnsubscribe: 'deviceUnsubscribeFunc',
   securityTest:'PushApplication-strong-mobile-securityTest
'
});
```

- `name` – A name by which the event source is referenced.
- `onDeviceSubscribe` – An adapter function that is called when the request for user subscription is received.
- `onDeviceUnsubscribe` – An adapter function that is called when the request for user unsubscription is

received.

- `securityTest` – A security test from the **authenticationConfig.xml** file that is used to protect the event source.

Sending a notification

Notifications can be either polled from, or pushed by, the back-end system. In this example, a `submitNotifications()` adapter function is invoked by a back-end system as an external API to send notifications.

```
function submitNotification(userId, notificationText) {
    var userSubscription = WL.Server.getUserNotificationSubscription('PushAdapter.PushEventSource', userId);
    if (userSubscription === null) {
        return { result: "No subscription found for user :: " + userId };
    }
    var badgeDigit = 1;
    var notification = WL.Server.createDefaultNotification(notificationText, badgeDigit, {custom:"data"});
        WL.Server.notifyAllDevices(userSubscription, notification);
    return {
        result: "Notification sent to user :: " + userId
    };
}
```

Notification API: Client side

A device subscription belongs to a user subscription and exists in the scope of a specific user and event source. A user subscription can have several device subscriptions.

The device subscription is created when the application on a device calls the `[[WLPush sharedInstance]subscribe]` method.

The device subscription is deleted either by an application that calls `[[WLPush sharedInstance] unsubscribe]` or when the push mediator informs MobileFirst Platform Server that the device is permanently not accessible.

1. Access the `WLPush` functionality by using `[WLPush sharedInstance]` anywhere in your application.
2. Create an instance of `onReadyToSubscribeListener`.

   ```
   ReadyToSubscribeListener *readyToSubscribeListener = [[ReadyToSubscribeListener alloc] initWithController:self];
    readyToSubscribeListener.alias = self.alias;
    readyToSubscribeListener.adapterName = self.adapterName;
    readyToSubscribeListener.eventSourceName = self.eventSourceName;
   ```

3. Set the `onReadyToSubscribeListener` on `WLPush`.

   ```
   [[WLPush sharedInstance] setOnReadyToSubscribeListener:readyToSubscribeListener];
   ```

4. Pass the token to `WLPush`.

```
[[WLPush sharedInstance] setTokenFromClient:deviceToken.description];
```

Sending token to client and initializing `WLPush`

The user must initialize the `WLPush` `sharedInstance` in the app `ViewController` load method.

```
AppDelegate *appDelegate = [[UIApplication sharedApplication]delegate];
 appDelegate.appDelegateVC = self;
[[WLPush sharedInstance]init];
```

The user must add this method to the app delegate to get the token.

```
-(void)application:(UIApplication *)application didRegisterForRemoteNotificationsWithDeviceToken:(NSData
*)deviceToken{
}<br />
```

The token that is received by this method must be passed to the WLPush method.

```
[[WLPush sharedInstance] setTokenFromClient:deviceToken.description];
```

Registering the event source

IBM MobileFirst Platform Foundation provides the customizable `onReadyToSubscribe` function, which is used to register an event source.
Set up your `onReadyToSubscribe` function in Listener, which implements `WLOnReadyToSubscribeListener`.

This function is called when authentication finishes.

```
#import "ReadyToSubscribeListener.h"
#import "MyEventSourceListener.h"
<p>@implementation ReadyToSubscribeListener
<p>- (id)initWithController: (ViewController *) mainView{
    if ( self = [super init] )
    {
        vc = mainView;
    }
    return self;
}
-(void)OnReadyToSubscribe{
 [vc updateMessage:@"\nPreparing to subscribe"];
 MyEventSourceListener *eventSourceListener=[[MyEventSourceListener alloc]init];
 [[WLPush sharedInstance] registerEventSourceCallback:self.alias :self.adapterName:self.eventSourceNam
e :eventSourceListener];
 [vc updateMessage:@"Ready to subscribe..."];
}
@end<
```

Subscribing to the event source

Prerequisite: To subscribe, a user must authenticate.

To set up subscription to the event source, use the following API:

```
- (IBAction)subscribe:(id)sender {
    self.console.text=@"Trying to subscribe ...";
 MySubscribeListener *mySubscribeListener = [[MySubscribeListener alloc] initWithController:self];
 [[WLPush sharedInstance]subscribe:self.alias :nil :mySubscribeListener];
<p>}
```

`[[WLPush sharedInstance] subscribe]` takes the following parameters:

- An alias, as declared in `[[WLPush sharedInstance] registerEventSourceCallback]`
- `onSuccess` delegate (optional)
- `onFailure` delegate (optional)

Delegates receive a response object if one is required.

Unsubscribing from an event source

To set up unsubscription from an event source, use the following API:

```
- (IBAction)unsubscribe:(id)sender {
    self.console.text = @"Trying to unsubscribe ... ";
 MyUnsubscribeListener *myUnsubscribelistener = [[MyUnsubscribeListener alloc]initWithController:self];>
 [[WLPush sharedInstance]unsubscribe:self.alias :myUnsubscribelistener];
}
```

`[[WLPush sharedInstance] unsubscribe]` takes the following parameters:

- An alias, as declared in `[[WLPush sharedInstance] registerEventSourceCallback]`
- `onSuccess` delegate (optional)
- `onFailure` delegate (optional)

Delegates receive a response object if one is required.

Additional client-side APIs

`[[WLPush sharedInstance]isPushSupported]` – Returns `true` if push notifications are supported by the platform, or `false` otherwise.
`[[WLPush sharedInstance]isSubscribed:alias]` – Returns whether the currently logged-in user is subscribed to a specified event source alias.
When a push notification is received by a device, the `didReceiveRemoteNotification` method is called in the app delegate.

```
-(void)application:(UIApplication *)application didReceiveRemoteNotification:(NSDictionary *)userInfo{
    NSLog(@"Received Notification %@",userInfo.description);
    _appDelegateVC.console.text=userInfo.description;
}
```

If the application was in background mode (or inactive) when the push notification arrived, this callback is called when the application returns to the foreground.

Tag-based notification

Tags represent topics of interest to the user and provide users the ability to receive notifications according to the chosen interest.
This notification type enables devices to send and receive messages that are filtered by tags.
To start receiving tag-based notifications, the device must first subscribe to a push notification tag in an application.
Tags are defined in the **application-descriptor.xml** file:

```
<tags>
   <tag>
      <name>PushTag1</name>
      <description>About pushTag1</description
>
   </tag>
   <tag>
      <name>PushTag2</name>
      <description>About pushTag2</description
>
   </tag>
</tags>
```

Such notification is targeted to all devices that are subscribed to a tag in an application.

Client-side methods:

`[[WLPush sharedInstance]subscribeTag:tagName :options)]`

Subscribes the device to the specified tag name.

`[[WLPush sharedInstance]unsubscribeTag:tagName :options)]`

Unsubscribes the device from the specified tag name.

`[WLPush sharedInstance]isTagSubscribed:tagName]`

Returns whether the device is subscribed to a specified tag name.

Broadcast notification

Broadcast notification is enabled by default for any push-enabled MobileFirst application. A subscription to a reserved tag, `Push.ALL`, is created for every device.
Broadcast notification can be disabled by unsubscribing to the reserved tag `Push.ALL`.

Common API for tag-based and broadcast notifications

Client-side API:

When a notification is received by a device, the `didReceiveRemoteNotification` method in the app delegate is called.

```
-(void)application:(UIApplication *)application didReceiveRemoteNotification:(NSDictionary *)userInfo{<br />
    NSLog(@"Received Notification %@",userInfo.description);
    _appDelegateVC.console.text=userInfo.description;
}
```

- userInfo - A JSON block that contains the payload field. This field holds other data that is sent from the MobileFirst Platform server. It also contains the tag name for tag and broadcast notification. The tag name appears in the tag element. For broadcast notification, the default tag name is `Push.ALL`.

Server-side API

`WL.Server.sendMessage(applicationId,notificationOptions)`

This method submits a notification based on the specified target parameters and takes two mandatory parameters:

`applicationId` - The name of the MobileFirst application.

`notificationOptions` - A JSON block that contains message properties.

For a full list of message properties, see the user documentation.

Silent notification

Silent notification is the feature that sends notifications without disturbing the user. Notifications are not shown in the notification center or notification area.
Callback methods are executed even when the application is running in the background. Support for silent notifications has been added from iOS7 onwards.
For more information, refer to the topics about silent notifications in the MobileFirst user documentation and Apple 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.

Native client steps for silent notification

To handling silent notification on the client side:

1. Enable the application capability to perform background tasks on receiving the remote notifications.
2. Implement a new callback method on 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 notification

Interactive notification enables the users to take actions when a notification is received without the application being open.
When an interactive notification is received, the device shows the action buttons along with the notification message
Supported from iOS8 onwards.
For more information about interactive notification, see the MobileFirst user documentation and Apple documentation.

Server API for interactive notification

To send interactive notification, set a string to indicate the category.

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.

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

```
if([application respondsToSelector:@selector(registerUserNotificationSettings:)]){
  UIUserNotificationType userNotificationTypes = UIUserNotificationTypeNone | UIUserNotificationTyp
eSound | UIUserNotificationTypeAlert | UIUserNotificationTypeBadge;</p>
  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] init];
  cateogory.identifier = @"poll";
  [cateogory setActions:@[acceptAction,rejetAction] forContext:UIUserNotificationActionContextDefaul
t];
  [cateogory setActions:@[acceptAction,rejetAction] forContext:UIUserNotificationActionContextMinim
al];
  NSSet *catgories = [NSSet setWithObject:cateogory];
  [application registerUserNotificationSettings:[UIUserNotificationSettings settingsForTypes:userNotifi
cationTypes categories:catgories]];
}
```

- Implement the new callback method:

```
(void)application:(UIApplication *)application handleActionWithIdentifier:(NSString *)identifier forRemo
teNotification:(NSDictionary *)userInfo completionHandler:(void (^)())completionHandler
```

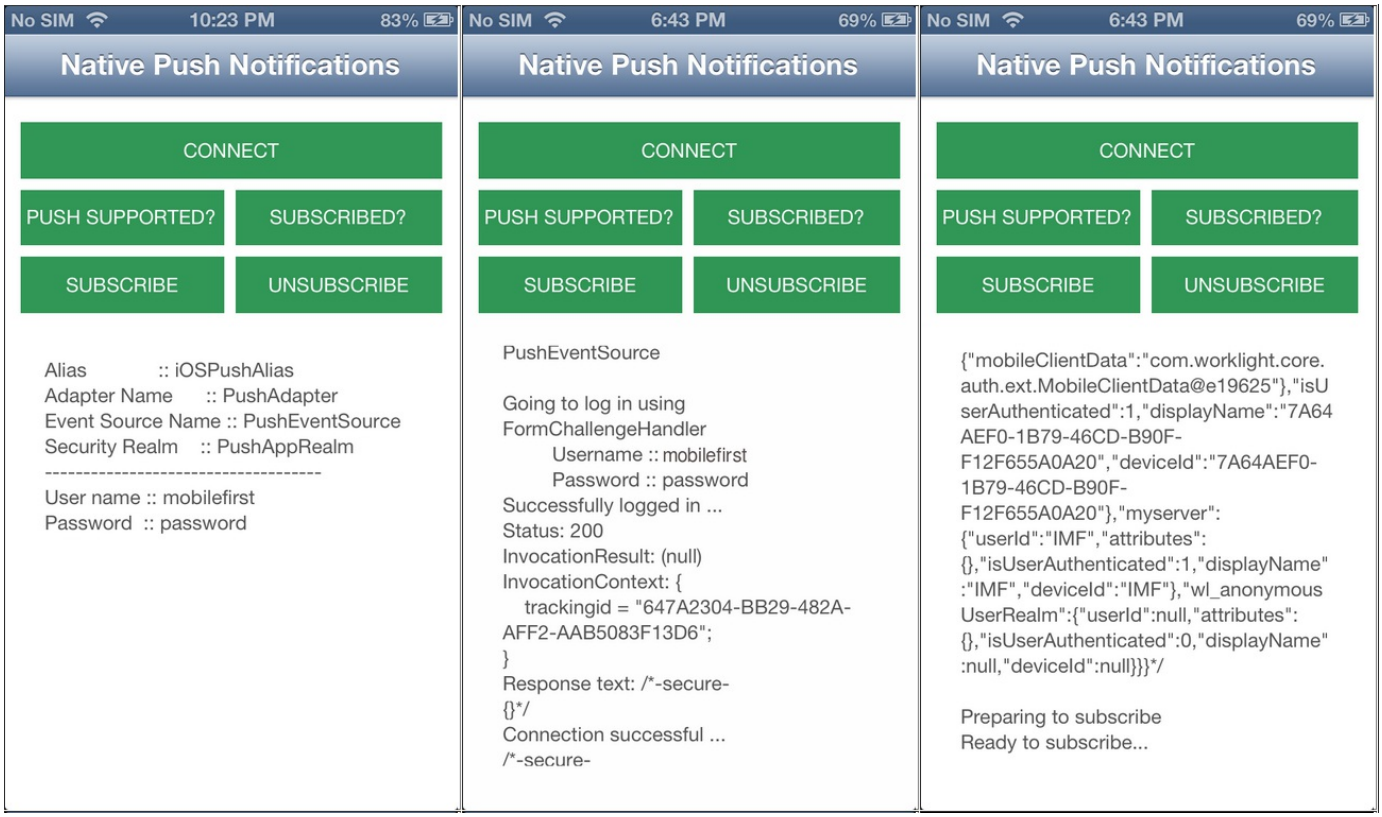
This new callback method is invoked when the user clicks the action button.

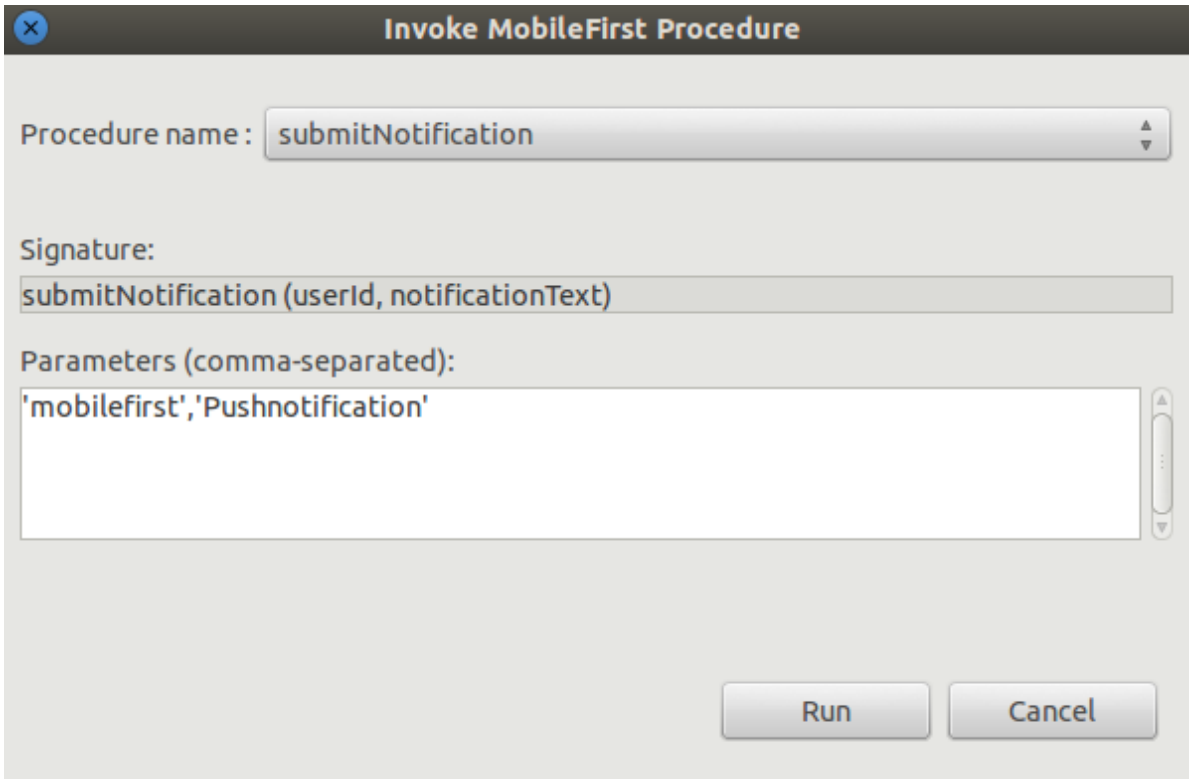
The implementation of this method must perform the action that is associated with the specified identifier and execute the block in the `completionHandler` parameter.

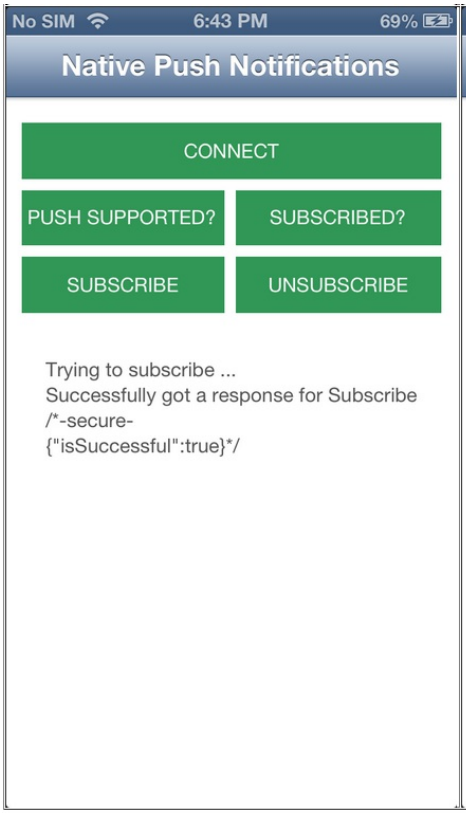
Sample application

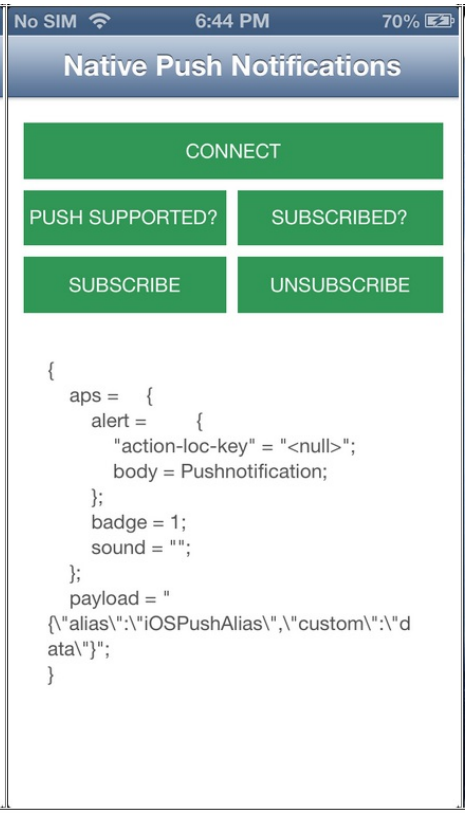
Click to download (http://public.dhe.ibm.com/software/products/en/MobileFirstPlatform/docs/v630/PushNotificationsNativeProject.zip) the Studio project.
Click to download (http://public.dhe.ibm.com/software/products/en/MobileFirstPlatform/docs/v630/iOSNativePushProject.zip) the Native project.

The sample contains two projects:
- The **PushNotificationsNativeProject.zip** file contains a MobileFirst native API that you can deploy to your MobileFirst server.
-The **iOSNativePushProject.zip** file contains a native iOS application that uses a MobileFirst native API library to subscribe for push notification and receive notifications from APNS.
Make sure to update the **worklight.plist** file in iOSNativePush with the relevant server settings.


No SIM 10:23 PM 83%
Native Push Notifications
CONNECT
PUSH SUPPORTED?
SUBSCRIBED?
SUBSCRIBE
UNSUBSCRIBE
Alias :: iOSPushAlias
Adapter Name :: PushAdapter
Event Source Name :: PushEventSource
Security Realm :: PushAppRealm
User name :: mobilefirst
Password :: password
No SIM 6:43 PM 69%
Native Push Notifications
CONNECT
PUSH SUPPORTED?
SUBSCRIBED?
SUBSCRIBE
UNSUBSCRIBE
PushEventSource
Going to log in using
FormChallengeHandler
Username :: mobilefirst
Password :: password
Successfully logged in ...
Status: 200
InvocationResult: (null)
InvocationContext: {
trackingid = "647A2304-BB29-482A-AFF2-AAB5083F13D6";
}
Response text: /*-secure-
{}*/
Connection successful ...
/*-secure-
No SIM 6:43 PM 69%
Native Push Notifications
CONNECT
PUSH SUPPORTED?
SUBSCRIBED?
SUBSCRIBE
UNSUBSCRIBE
{"mobileClientData":"com.worklight.core.auth.ext.MobileClientData@e19625"},"isUserAuthenticated":1,"displayName":"7A64AEF0-1B79-46CD-B90F-F12F655A0A20","deviceId":"7A64AEF0-1B79-46CD-B90F-F12F655A0A20"},"myserver":{"userId":"IMF","attributes":{},"isUserAuthenticated":1,"displayName":"IMF","deviceId":"IMF"},"wl_anonymousUserRealm":{"userId":null,"attributes":{},"isUserAuthenticated":0,"displayName":null,"deviceId":null}}}*/
Preparing to subscribe
Ready to subscribe...


Invoke MobileFirst Procedure
Procedure name :
submitNotification
Signature:
submitNotification (userId, notificationText)
Parameters (comma-separated):
'mobilefirst','Pushnotification'
Run
Cancel


No SIM
6:43 PM
69%
Native Push Notifications
CONNECT
PUSH SUPPORTED?
SUBSCRIBED?
SUBSCRIBE
UNSUBSCRIBE
Trying to subscribe ...
Successfully got a response for Subscribe
/*-secure-
{"isSuccessful":true}*/


No SIM
6:44 PM
70%
Native Push Notifications
CONNECT
PUSH SUPPORTED?
SUBSCRIBED?
SUBSCRIBE
UNSUBSCRIBE
{
aps = {
alert = {
"action-loc-key" = "<null>";
body = Pushnotification;
};
badge = 1;
sound = "";
};
payload = "
{\"alias\":\"iOSPushAlias\",\"custom\":\"d
ata\"}";
}


iOSNativePush
Pushnotification
Instagram
Safari
Settings
Chrome