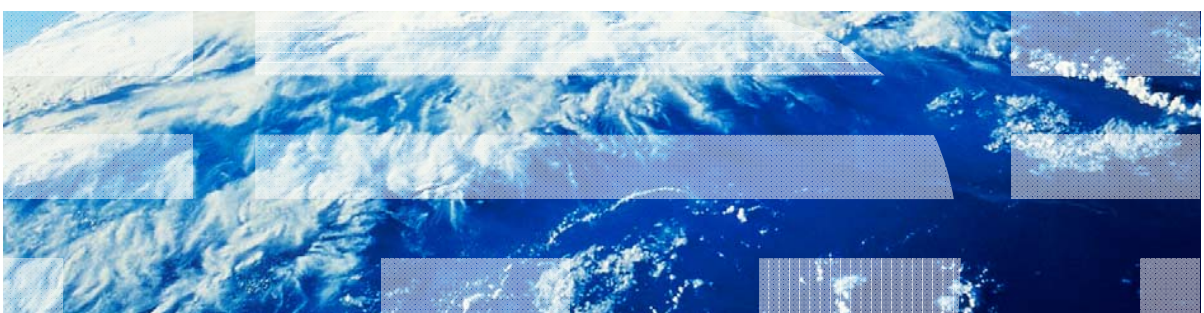
IBM Worklight V5.0.6
Getting Started

Using Worklight API for Push Notifications in native iOS applications



15 March 2013

© Copyright International Business Machines Corporation 2011, 2013. All rights reserved.
US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Trademarks

- IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Worklight is a trademark or registered trademark of Worklight, an IBM Company. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.
- Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.
- Other company products or service names may be trademarks or service marks of others.
- This document may not be reproduced in whole or in part without the prior written permission of IBM.

About IBM®

- See http://www.ibm.com/ibm/us/en/

© Copyright International Business Machines Corporation 2011, 2013. All rights reserved.

Agenda

- What are push notifications?
- Creating a Worklight native API for Push Notifications
- Create and configure an iOS native application
- Initializing the WLCLient and WLPush
- Subscription Management
- Notification API

© Copyright International Business Machines Corporation 2011, 2013. All rights reserved.

What are push notifications?

- Push notification is the ability of a mobile device to receive messages that are *pushed* from a server
- Notifications are received regardless of whether the application is running
- Notifications can take several forms:
  - **Alert**: a pop-up text message
  - **Badge**: a small badge mark that is appearing next to the application icon
  - **Sound alert**

© Copyright International Business Machines Corporation 2011, 2013. All rights reserved.

Agenda

- What are push notifications?
- Creating a Worklight native API for Push Notifications
- Create and configure an iOS native application
- Initializing the WLCLient and WLPush
- Subscription Management
- Notification API

© Copyright International Business Machines Corporation 2011, 2013. All rights reserved.

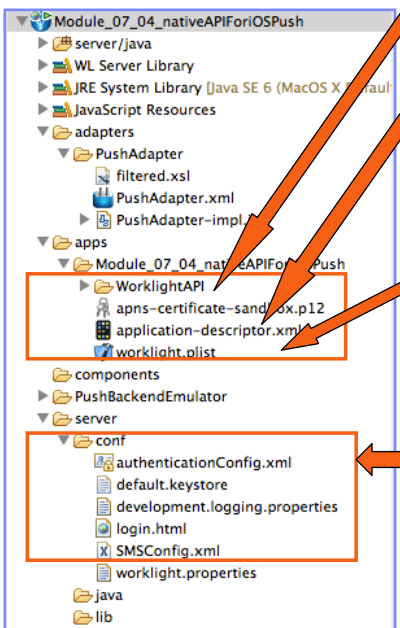
Creating a Worklight native API for Push Notifications

- IBM Worklight® provides ability for native iOS applications to communicate with a Worklight Server and use Push Notification platform by using the IBM Worklight native API library.
- To serve a native iOS application, the Worklight server must be aware of it.
- The Worklight native API is located under the `apps` folder of your Worklight project.
- The Worklight native API folder serves two purposes:
  - It contains a native API library and configuration file that must be copied to your native iOS project.
  - It contains the **application-descriptor.xml** file, which must be deployed to a Worklight server as an entry point, similar to a Worklight application.
- In this module, you learn how to create a Worklight native API and how to use its components in your native iOS application for working with Push Notifications.

© Copyright International Business Machines Corporation 2011, 2013. All rights reserved.

Creating a Worklight native API

- Worklight native API contains several components:



The **WorklightAPI** folder is a Worklight API library that must be copied to your native iOS project.

The **application-descriptor.xml** file is used to define application metadata and to configure security settings to be enforced by a Worklight server.

The **Worklight.plist** file contains connectivity settings to be used by a native iOS application. This file must be copied to your native iOS project.

Like with any Worklight project, you create the server configuration by modifying files under the **server\conf** folder.

© Copyright International Business Machines Corporation 2011, 2013. All rights reserved.

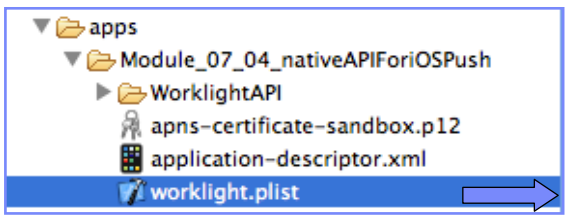
Creating a Worklight native API

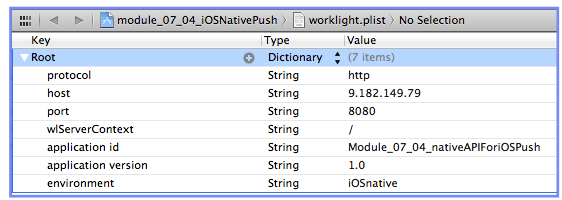
1. In Worklight Studio, create a Worklight project, and add a Worklight Native API.
2. In the New Worklight Native API dialog, enter your application name, and select **iOS** for the **Environment** field.
3. Add the APNs p12 keys (either apns-certificate-sandbox.p12 or apns-certificate-production.p12) to the root folder of the application.
4. Right-click the Worklight native API folder and select **Run As ➔ Deploy Native API**.

© Copyright International Business Machines Corporation 2011, 2013. All rights reserved.

Creating a Worklight native API

- Edit the ***worklight.plist*** file that holds the server configuration:





- ***protocol*** – The communication protocol to the Worklight server can be either ***http*** or ***https.***
- ***host*** – The hostname of the Worklight server.
- ***port*** – The port of the Worklight server.
- **wlServerContext** – The context root path of the application on the Worklight server.
- ***application id*** – The application ID as defined in the ***application-descriptor.xml*** file.
- ***application version*** – The application version.
- **environment** – The target environment of the native application (Android or iOS).

© Copyright International Business Machines Corporation 2011, 2013. All rights reserved.

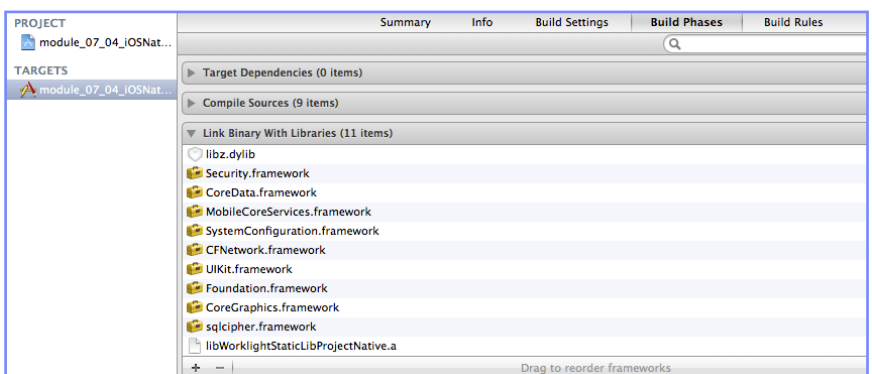
Agenda

- What are push notifications?
- Creating a Worklight native API for Push Notifications
- Create and configure an iOS native application
- Initializing the WLCLient and WLPush
- Subscription Management
- Notification API

© Copyright International Business Machines Corporation 2011, 2013. All rights reserved.

Create and configure an iOS native application

- Create an Xcode project or use an existing one.
- Copy the WorklightAPI folder and the worklight.plist file from the Eclipse Worklight native API to the root of your native project.
- Link the following libraries in your native iOS application: CFNetwork, SystemConfiguration, MobileCoreServices, CoreData, Security, libz.dylib.



© Copyright International Business Machines Corporation 2011, 2013. All rights reserved.

Agenda

- What are push notifications?
- Creating a Worklight native API for Push Notifications
- Create and configure an iOS native application
- Initializing the WLCLient and WLPush
- Subscription Management
- Notification API

© Copyright International Business Machines Corporation 2011, 2013. All rights reserved.

Initializing the WLClient and WLPush

- Access the WLClient functionality by using [WLClient sharedInstance] anywhere in your application.
- Initiate the connection to the server by using wlConnectWithDelegate method
- For most actions, you need to specify a delegate object, such as a MyConnectListener instance in the following example:

```
MyConnectListener *connectListener = [[MyConnectListener alloc] initWithController:self];
[[WLClient sharedInstance] wlConnectWithDelegate:connectListener];
[connectListener release];
```

- You learn how to create it in subsequent slides.
- Remember to import **WLClient.h** and **WLDelegate.h** in your header file.

© Copyright International Business Machines Corporation 2011, 2013. All rights reserved.

Initializing the WLClient and WLPush

- As described on a previous slide, you must to supply a connection delegate (listener) to Worklight server invocation methods.
- Create a delegate to be used in the wlConnectWithDelegate method and receive the response from the Worklight server. Name the class ***MyConnectListener***.
- The header file must specify that it implements the ***WLDelegate*** protocol.

```
#import <Foundation/Foundation.h>
#import "WLClient.h"
#import "WLDelegate.h"
#import "ViewController.h"

@interface MyConnectListener : NSObject <WLDelegate> {
    @private
    ViewController *vc;
}

- (id)initWithController: (ViewController *)mainView;
@end
```

- The ***WLDelegate*** protocol specifies that the class implements the following methods:
  - ***The onSuccess (WLResponse *)response***
  - ***The onFailure (WLFailResponse *)response***

© Copyright International Business Machines Corporation 2011, 2013. All rights reserved.

Initializing the WLCLient and WLPush

- After wlConnectWithDelegate finishes, the onSuccess method, or the onFailure method, of the supplied MyConnectListener instance is invoked.
- In both cases, the response object is sent as an argument.
- Use this object to operate data that are retrieved from server.

```
-(void)onSuccess:(WLResponse *)response{
    NSLog(@"\nConnection Success: %@", response);
    NSString *resultText = @"Connection success. ";

    if ([response responseText] != nil){
        resultText = [resultText stringByAppendingString:[response responseText]];
    }
    [vc updateView:resultText];
}

-(void)onFailure:(WLFailResponse *)response{
    NSString *resultText = @"Connection failure. ";

    if ([response responseText] != nil){
        resultText = [resultText stringByAppendingString:[response responseText]];
    }
    [vc updateView:resultText];
}
```


© Copyright International Business Machines Corporation 2011, 2013. All rights reserved.

Initializing the WLClient and WLPush

- Access the WLPush functionality by using [WLPush sharedInstance] anywhere in your application.
- Create a onReadyToSubscribeListener

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

- Set the onReadyToSubscribeListener on WLPush.

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

- Pass the token to WLPush

```
[[WLPush sharedInstance] setTokenFromClient:self.myToken];
```


© Copyright International Business Machines Corporation 2011, 2013. All rights reserved.

Agenda

- What are push notifications?
- Creating a Worklight native API for Push Notifications
- Create and configure an iOS native application
- Initializing the WLCLient
- Subscription management
- Notification API

© Copyright International Business Machines Corporation 2011, 2013. All rights reserved.

Subscription management
User subscription

- **User subscription**
  - An entity containing user ID, device ID and event source ID. It represents the intent of the user to receive notification from a specific event source.
- **Creation**
  - The user subscription for an event source is created when the user subscribes to that event source for the first time from any device
- **Deletion**
  - A user subscription is deleted when the user unsubscribes from that event source from all of his or her devices
- **Notification**
  - While the user subscription exists, the Worklight 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 the user has subscribed from

© Copyright International Business Machines Corporation 2011, 2013. All rights reserved.

Subscription management
Device subscription

- 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] API

- The device subscription is deleted either by an application that is calling [[WlPush sharedInstance] unsubscribe] or when the push mediator informs the Worklight server that the device is permanently not accessible

© Copyright International Business Machines Corporation 2011, 2013. All rights reserved.

Agenda

- What are push notifications?
- Creating a Worklight native API for Push Notifications
- Create and configure an iOS native application
- Initializing the WLCLient and WLPush
- Subscription Management
- Notification API

© Copyright International Business Machines Corporation 2011, 2013. All rights reserved.

Notification API: Server side

- Start by creating an event source
  - Declare a notification event source in the adapter JavaScript™ code at a global level (outside any JavaScript function).

Notifications are pushed by the back end

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

Notifications are polled from the back end

```
WL.Server.createEventSource({
    name: 'PushEventSource',
    onDeviceSubscribe: 'deviceSubscribeFunc',
    onDeviceUnsubscribe: 'deviceUnsubscribeFunc',
    securityTest:'PushApplication-strong-mobile-securityTest',
    poll: {
        interval: 3,
        onPoll: getNotificationsFromBackend
    }
});
```

  - name – a name by which the Event Source is referenced
  - onDeviceSubscribe – an adapter function that is invoked when user subscription request is received
  - onDeviceUnsubscribe – an adapter function that is invoked when user unsubscribe request is received
  - securityTest – a security test from **authenticationConfig.xml** file used to protect the event source

© Copyright International Business Machines Corporation 2011, 2013. All rights reserved.

Notification API: Server side

- Start by creating an event source
  - Declare a notification event source in the adapter JavaScript code at a global level (outside any JavaScript function).

Notifications are pushed by the back end

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

Notifications are polled from the back end

```
WL.Server.createEventSource({
    name: 'PushEventSource',
    onDeviceSubscribe: 'deviceSubscribeFunc',
    onDeviceUnsubscribe: 'deviceUnsubscribeFunc',
    securityTest:'PushApplication-strong-mobile-securityTest',
    poll: {
        interval: 3,
        onPoll: getNotificationsFromBackend
    }
});
```

  - poll – a method that will be used for notification retrieval. The following parameters are required:
    - **interval** – polling interval in seconds
    - **onPoll** – polling implementation – an adapter function to be invoked at specified intervals

© Copyright International Business Machines Corporation 2011, 2013. All rights reserved.

Notification API: Server side

- Sending a notification

```
function submitNotification(userId, notificationText){
    var userSubscription =
        WL.Server.getUserNotificationSubscription('PushAdapter.PushEventSource', userId);

    if (userSubscription==null){
        return { result: "No subscription found for user :: " + userId
    }

    var deviceSubscriptions =
        userSubscription.getDeviceSubscriptions();

    WL.Logger.debug("submitNotification >> userId :: " + userId + ", t

    WL.Server.notifyAllDevices(userSubscription, {
        badge: 1,
        sound: "sound.mp3",
        activateButtonLabel: "ClickMe",
        alert: notificationText,
        payload: {
            foo : 'bar'
        }
    });

    return { result: "Notification sent to user :: " + userId };
}
```

As described previously, notifications can be either polled from the back end or pushed by one. In this sample, a **submitNotifications()** adapter function is invoked by a back end as an external API to send notifications.

© Copyright International Business Machines Corporation 2011, 2013. All rights reserved.

Notification API: Server side

- Sending a notification
  - Obtain notification data

```
function submitNotification(userId, notificationText){
    var userSubscription =
        WL.Server.getUserNotificationSubscription('PushAdapter.PushEventSource', userId);

    if (userSubscription==null){
        return { result: "No subscription found for user :: " + userId };
    }

    var deviceSubscriptions =
        userSubscription.getDeviceSubscriptions();

    WL.Logger.debug("submitNotification >> userId :: " + userId + ", t

    WL.Server.notifyAllDevices(userSubscription, {
        badge: 1,
        sound: "sound.mp3",
        activateButtonLabel: "ClickMe",
        alert: notificationText,
        payload: {
            foo : 'bar'
        }
    });

    return { result: "Notification sent to user :: " + userId };
}
```

The submitNotification function receives the **userId** to send notification to and the **notificationText**. These arguments are provided by a back end which invokes this function

© Copyright International Business Machines Corporation 2011, 2013. All rights reserved.

Notification API: Server side

- Sending a notification
  - Retrieve the active user and use it to get user's subscription data

```
function submitNotification(userId, notificationText){
    var userSubscription =
        WL.Server.getUserNotificationSubscription('PushAdapter.PushEventSource', userId);

    if (userSubscription==null){
        return { result: "No subscription found for user :: " + userId };
    }

    var deviceSubscriptions =
        userSubscription.getDeviceSubscriptions();

    WL.Logger.debug("submitNotification >> userId

    WL.Server.notifyAllDevices(userSubscription,
        badge: 1,
        sound: "sound.mp3",
        activateButtonLabel: "ClickMe",
        alert: notificationText,
        payload: {
            foo : 'bar'
        }
    });

    return { result: "Notification sent to user
}
```

A user subscription object contains the information about all of the user's subscriptions. Each user subscription can have several device subscriptions. The object structure is as follows:

```
{
    userId: 'bjones',
    state: {
        customField: 3
    },
    getDeviceSubscriptions: function(){}
};
```


© Copyright International Business Machines Corporation 2011, 2013. All rights reserved.

Notification API: Server side

- Sending a notification
  - Retrieve the user subscription data

```
function submitNotification(userId, notificationText){
    var userSubscription =
        WL.Server.getUserNotificationSubscription('PushAdapter.PushEventSource', userId);

    if (userSubscription==null){
        return { result: "No subscription found for user :: " + userId };
    }

    var deviceSubscriptions =
        userSubscription.getDeviceSubscriptions();

    WL.Logger.debug("submitNotification >> userId :: " + userId

    WL.Server.notifyAllDevices(userSubscription, {
        badge: 1,
        sound: "sound.mp3",
        activateButtonLabel: "ClickMe",
        alert: notificationText,
        payload: {
            foo : 'bar'
        }
    });

    return { result: "Notification sent to user :: " + userId };
}
```

If the user has no subscriptions for the specified event source, a **null** object is returned

© Copyright International Business Machines Corporation 2011, 2013. All rights reserved.

Notification API: Server side

- Sending a notification
  - Retrieve the user subscription data

```
function submitNotification(userId, notificationText){
    var userSubscription =
        WL.Server.getUserNotificationSubscription('PushAdapter.PushEve

    if (userSubscription==null){
        return { result: "No subscription found for user :: " + userId
    }

    var deviceSubscriptions =
        userSubscription.getDeviceSubscriptions();

    WL.Logger.debug("submitNotification >> userId :: " + userId + ", t

    WL.Server.notifyAllDevices(userSubscription, {
        badge: 1,
        sound: "sound.mp3",
        activateButtonLabel: "ClickMe",
        alert: notificationText,
        payload: {
            foo : 'bar'
        }
    });

    return { result: "Notification sent to user :: " + userId };
}
```

Separate subscription data for each of the user's devices can be obtained by using the **getDeviceSubscriptions** API. The result is an array of objects with the following structure:

```
[
  {
    alias: "myPush",
    device: "4AooAq83gUSoas.....",
    token: 'KQz0srTUXsOqh.....',
    applicationId: 'PushApp',
    platform: 'Android',
    options: {
        customOption: 'aaa',
        alert: true,
        badge: true,
        sound: true
    }
  }
]
```


© Copyright International Business Machines Corporation 2011, 2013. All rights reserved.

Notification API: Server side

- Sending a notification
  - Send notification to the users device or devices

```
function submitNotification(userId, notificationText){
    var userSubscription =
        WL.Server.getUserNotificationSubscription('PushAdapter.PushEventSource', userId);

    if (userSubscription==null){
        return { result: "No subscription found for user :: " + userId };
    }

    var deviceSubscriptions =
        userSubscription.getDeviceSubscriptions();

    WL.Logger.debug("submitNotification >> userId :: " + userId

    WL.Server.notifyAllDevices(userSubscription, {
        badge: 1,
        sound: "sound.mp3",
        activateButtonLabel: "ClickMe",
        alert: notificationText,
        payload: {
            foo : 'bar'
        }
    });

    return { result: "Notification sent to user :: " + userId };
}
```

WL.Server.notifyAllDevices API sends notification to all the devices that are subscribed to the user. Custom properties can be sent in the **payload** object.

© Copyright International Business Machines Corporation 2011, 2013. All rights reserved.

Notification API: Server side

- Several APIs exist for notification sending.
- WL.Server.notifyAllDevices(**`userSubscription`**, **`options`**) is used to send notification to all a user's devices (see previous slide).
- WL.Server.notifyDevice(**`userSubscription`**, **`device`**, **`options`**) is used to send notification to a specific device belonging to a specific userSubscription.
- WL.Server.notifyDeviceSubscription(**`deviceSubscription`**, **`options`**) is used to send the notification to a specific device.

© Copyright International Business Machines Corporation 2011, 2013. All rights reserved.

Notification API: Client side

Sending token to client and WLPush initialization

The user must initialize the WLPush sharedInstance in the App's ViewController's 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
{
    NSLog(@"My token from APNS : %@", deviceToken);
    _appDelegateVC.myToken = deviceToken.description;
}
```

– The token that is received by this method should be passed to the WLPush method. [[WLPush sharedInstance] setTokenFromClient]:

```
[[WLPush sharedInstance] setTokenFromClient:self.myToken];
```


© Copyright International Business Machines Corporation 2011, 2013. All rights reserved.

Notification API: Client side

- Event Source – registration

  The first task is to register an event source within the application

  IBM® Worklight provides the customizable onReadyToSubscribe function that is used to register an event source

  Set up your onReadyToSubscribe function in Listener which implements WLOnReadyToSubscribeListener. It is invoked when the authentication finishes

```
#import "ReadyToSubscribeListener.h"
#import "MyEventSourceListener.h"

@implementation ReadyToSubscribeListener

- (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.eventSourceName :eventSourceListener];
    [vc updateMessage:@"Ready to subscribe..."];
}

@end
```


© Copyright International Business Machines Corporation 2011, 2013. All rights reserved.

Notification API: Client side

Event Source – subscribing and unsubscribing

A user must be authenticated in order to subscribe

Use the following API to subscribe to the event source

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

- **[[WLPush sharedInstance] subscribe]**receives the following parameters:
  - An alias declared in [[WLPush sharedInstance] registerEventSourceCallback]
  - Optional onSuccess delegate
  - Optional onFailure delegate
- Delegates receive a response object if one is required

© Copyright International Business Machines Corporation 2011, 2013. All rights reserved.

Notification API: Client side

- **Event Source** – subscribing and unsubscribing
  - Use the following API to unsubscribe from the event source

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

```
-(void) onSuccess:(WLResponse *)response{
    [vc updateMessage:@"Successfully got a response for Unsubscribe"];
    [vc updateMessage:response.responseText];
}

-(void)onFailure:(WLFailResponse *)response{
    [vc updateMessage:@"Failed get a response for Unsubscribe"];
    [vc updateMessage:response.responseText];
}
```

  - **[[WLPush sharedInstance] unsubscribe]** receives the following parameters:
    - An **alias** declared in WL.Client.Push.registerEventSourceCallback
    - Optional onSuccess delegate
    - Optional onFailure delegate
  - Delegates receive a response object if one is required

© Copyright International Business Machines Corporation 2011, 2013. All rights reserved.

Notification API Client side

- Additional client side APIs:
  - **[[WLPush sharedInstance]isPushSupported]** – returns true if push notifications are supported by the platform, and false otherwise
  - **[[WlPush sharedInstance]isSubscribed :alias]** – returns whether the currently logged-in user is subscribed to a specified event source alias
- When push notification is received by a device , the didReceiveRemoteNotification method in the app delegate is invoked.

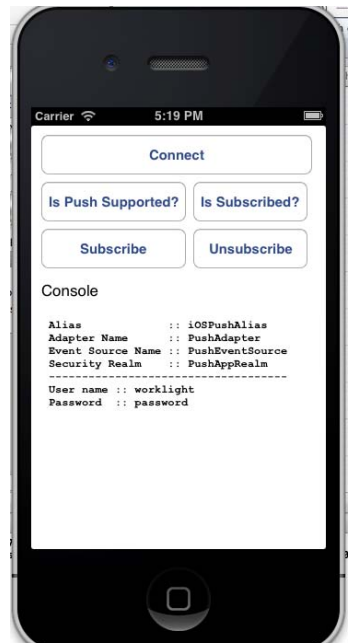
```
- (void)application:(UIApplication*)application didReceiveRemoteNotification
                  :(NSDictionary*)userInfo
{
    _appDelegateVC.result.text = userInfo.description;
}
```

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

© Copyright International Business Machines Corporation 2011, 2013. All rights reserved.

Receiving a procedure response

- The sample for this training module can be found in the Getting Started page of the IBM Worklight documentation website at http://www.ibm.com/mobile-docs.
- Sample contains two projects:
  - The **module_07_4_NativeAPIForIOSPush.zip** contains a Worklight native API to be deployed to your Worklight server.
  - The **module_07_4_iOSNativePush.zip** contains native iOS application that uses a Worklight native API library to communicate with the Worklight server.
- Make sure to update the **wlclient.plist** file in iOSNativeApp with relevant server settings.



© Copyright International Business Machines Corporation 2011, 2013. All rights reserved.

Notices

- Permission for the use of these publications is granted subject to these terms and conditions.
- This information was developed for products and services offered in the U.S.A.
- IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.
- IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:
  - IBM Director of Licensing
    IBM Corporation
    North Castle Drive
    Armonk, NY 10504-1785
    U.S.A.
- For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:
  - Intellectual Property Licensing
    Legal and Intellectual Property Law
    IBM Japan Ltd.
    1623-14, Shimotsuruma, Yamato-shi
    Kanagawa 242-8502 Japan
- **The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:** INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.
- This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.
- Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.
- IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.
- Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:
  - IBM Corporation
    Dept F6, Bldg 1
    294 Route 100
    Somers NY 10589-3216
    USA
- Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.
- The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.
- Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

COPYRIGHT LICENSE:

- This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.
- Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:
  - © (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

Privacy Policy Considerations

- IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.
- Depending upon the configurations deployed, this Software Offering may use session cookies that collect session information (generated by the application server). These cookies contain no personally identifiable information and are required for session management. Additionally, persistent cookies may be randomly generated to recognize and manage anonymous users. These cookies also contain no personally identifiable information and are required.
- If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent. For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at http://www.ibm.com/privacy and IBM's Online Privacy Statement at http://www.ibm.com/privacy/details/en/us sections entitled "Cookies, Web Beacons and Other Technologies" and "Software Products and Software-as-a-Service".

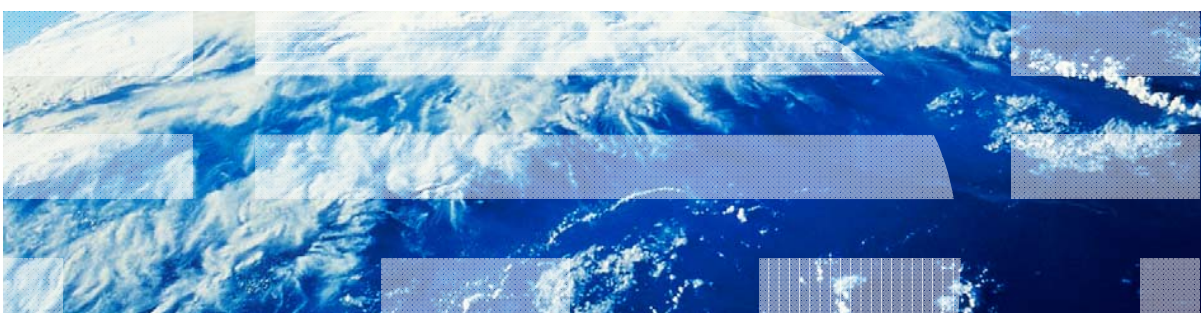
© Copyright International Business Machines Corporation 2011, 2013. All rights reserved.

Support and comments

- For the entire IBM Worklight documentation set, training material and online forums where you can post questions, see the IBM website at:
  - http://www.ibm.com/mobile-docs
- **Support**
  - Software Subscription and Support (also referred to as Software Maintenance) is included with licenses purchased through Passport Advantage and Passport Advantage Express. For additional information about the International Passport Advantage Agreement and the IBM International Passport Advantage Express Agreement, visit the Passport Advantage website at:
    - http://www.ibm.com/software/passportadvantage
  - If you have a Software Subscription and Support in effect, IBM provides you assistance for your routine, short duration installation and usage (how-to) questions, and code-related questions. For additional details, consult your IBM Software Support Handbook at:
    - http://www.ibm.com/support/handbook
- **Comments**
  - We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this document. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.
  - For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer.
  - When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state.
  - Thank you for your support.
  - Submit your comments in the IBM Worklight forums at:
    - https://www.ibm.com/developerworks/mobile/mobileforum.html
  - If you would like a response from IBM, please provide the following information:
    - Name
    - Address
    - Company or Organization
    - Phone No.
    - Email address

© Copyright International Business Machines Corporation 2011, 2013. All rights reserved.

Thank You



© Copyright International Business Machines Corporation 2011, 2013. All rights reserved.
US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.