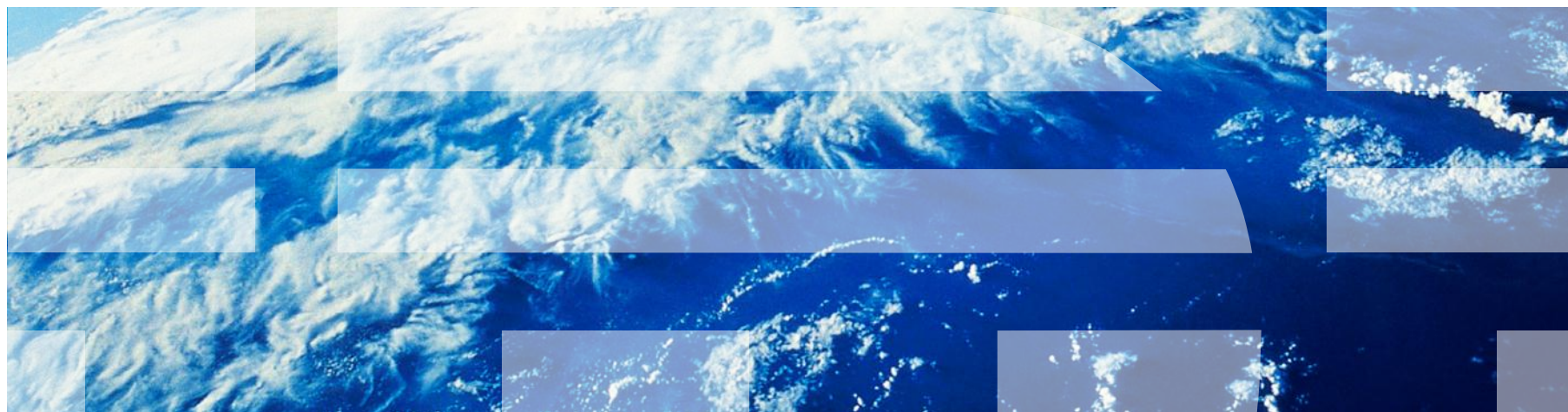


IBM Worklight Foundation V6.2.0 Getting Started

Push notifications in native iOS applications



Trademarks

- IBM, the IBM logo, ibm.com, and Worklight are trademarks or registered trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. 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](http://www.ibm.com/legal/copytrade.shtml)” 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/>

Agenda

- What are push notifications?
- Creating a Worklight native API for push notifications
- Creating and configuring an iOS native application
- Initializing WLClient and WLPush
- Subscription management
- Notification API
- Tag-based and broadcast notification

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 appears next to the application icon
 - **Sound alert**



Agenda

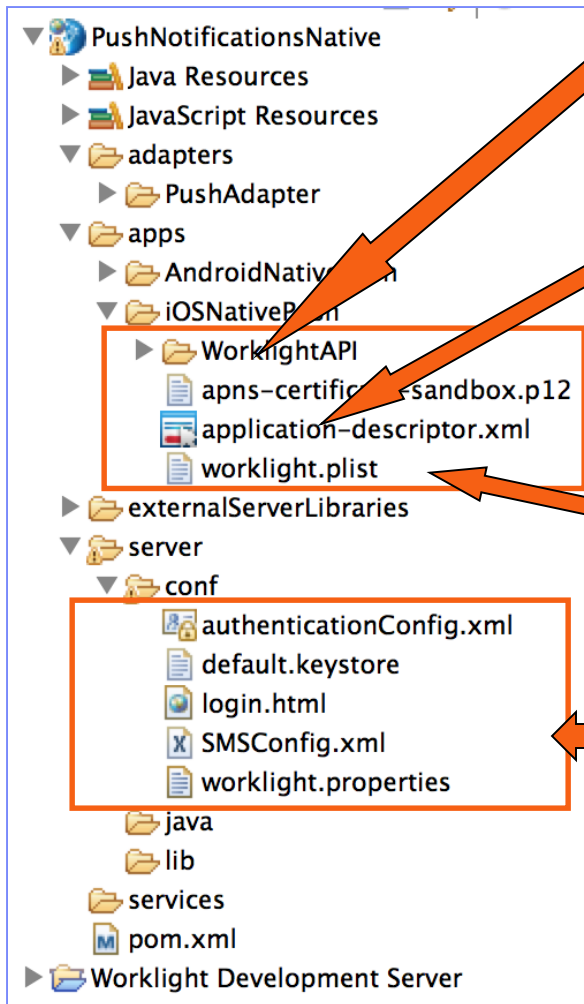
- What are push notifications?
- Creating a Worklight native API for push notifications
- Creating and configuring an iOS native application
- Initializing WLClient and WLPush
- Subscription management
- Notification API
- Tag-based and broadcast notification

Creating a Worklight native API for push notifications

- With IBM Worklight Foundation ®, native iOS applications can communicate with a Worklight Server by using the Worklight native API library.
- To serve a native iOS application, the Worklight Server must detect it.
- You can find the native API folder in the application folder of your Worklight project.
- The native API folder contains a native API library and configuration file that you must copy to your native iOS project.
- The native application contains the `application-descriptor.xml` file where you can configure the application metadata. The native application is deployed to the server.
- In this module, you learn how to generate a Worklight native API and how to use its components in your native iOS application.

Creating a Worklight native API (1 of 3)

- The Worklight native API contains several components:



The **WorklightAPI** folder is a library that you must copy 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 Worklight Server.

The **worklight.plist** file contains connectivity settings to be used by a native iOS application. Copy this file to your native iOS project.

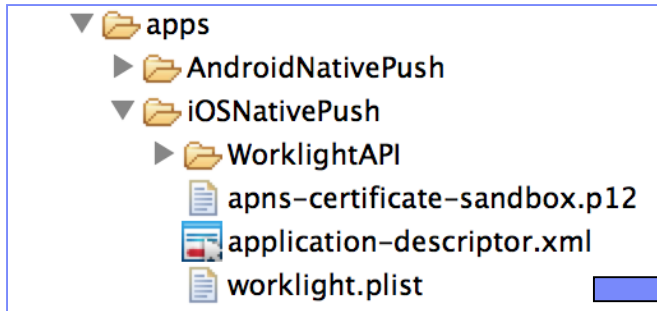
As with any Worklight project, you create the server configuration by modifying files under the **server\conf** folder.

Creating a Worklight native API (2 of 3)

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 in the **Environment** field, select **iOS**.
3. 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**).
4. Right-click the Worklight native API folder and click **Run As > Deploy Native API**.

Creating a Worklight native API (3 of 3)

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



Key	Type	Value
▼ Root	Dictionary	(9 items)
protocol	String	http
host	String	192.168.5.47
port	String	10080
wlServerContext	String	/PushNotificationsNative/
application id	String	iOSNativePush
application version	String	1.0
environment	String	iOSnative
wlUid	String	wY/mbnwKTDDYQUvuQCdSgg==
platformVersion	String	6.2.0.00.20140522-0617

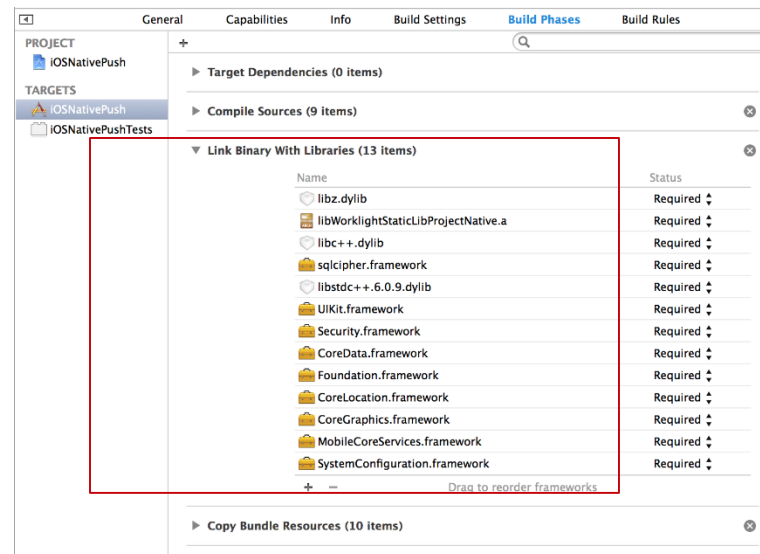
- ***protocol*** – The communication protocol to the Worklight Server can be either ***http*** or ***https***.
- ***host*** – The host name or IP address of the Worklight Server.
- ***port*** – The port of the Worklight Server.
- ***wlServerContext*** – The context root path of the application on the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

Agenda

- What are push notifications?
- Creating a Worklight native API for push notifications
- **Creating and configuring an iOS native application**
- Initializing WLClient and WLPush
- Subscription management
- Notification API
- Tag-based and broadcast notification

Creating and configuring an iOS native application (1 of 2)

- 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:
`SystemConfiguration.framework`, **`MobileCoreServices.framework`**,
`CoreData.framework`, **`Security.framework`**, **`libz.dylib`**,
`sqlcipher.framework`, **`libc++.dylib`**, **`libstdc++.6.dylib`**, and
`CoreLocation.framework`.



Creating and configuring an iOS native application (2 of 2)

- In the Build Settings:
 - Add the following entry: `$(SRCROOT)/WorklightAPI/include` for `HEADER_SEARCH_PATH`
 - In the **Other Linker Flags** field, enter the following value: `-ObjC`
 - In the Deployment section, for the **iOS Deployment Target** field, select a value that is greater than, or equal to, 5.0.

Agenda

- What are push notifications?
- Creating a Worklight native API for push notifications
- Creating and configuring an iOS native application
- **Initializing WLClient and WLPush**
- Subscription management
- Notification API
- Tag-based and broadcast notification

Initializing WLClient and WLPush (1 of 4)

- Access the `WLClient` functionality by using `[WLClient sharedInstance]` anywhere in your application.
- Initialize the connection to the server by using the `wlConnectWithDelegate` method.
- For most actions, you must 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.

Initializing WLClient and WLPush (2 of 4)

- As described on the previous slide, you must supply a connection delegate (a listener) to the methods that call Worklight Server.
- 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:
 - `onSuccess (WLResponse *) response`
 - `onFailure (WLFailResponse *) response`

Initializing WLClient and WLPush (3 of 4)

- After `wlConnectWithDelegate` finishes, either the `onSuccess` method or the `onFailure` method of the supplied `MyConnectListener` instance is called.
- In either case, the response object is sent as an argument.
- Use this object to operate data that is retrieved from the 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];
}
```


Initializing WLClient and WLPush (4 of 4)

- Access the `WLPush` functionality by using `[WLPush sharedInstance]` anywhere in your application.
- Create an instance of `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];
```

Agenda

- What are push notifications?
- Creating a Worklight native API for push notifications
- Creating and configuring an iOS native application
- Initializing WLClient and WLPush
- **Subscription management**
- Notification API
- Tag-based and broadcast notification

Subscription management – user subscription

- **User subscription**

- An entity that contains a user ID, device ID, and event source ID. The user subscription 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 owned 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 from which the user subscribed.

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 `[[W1Push sharedInstance] subscribe]` method.
- The device subscription is deleted either by an application that calls `[[W1Push sharedInstance] unsubscribe]` or when the push mediator informs Worklight Server that the device is permanently not accessible.

Agenda

- What are push notifications?
- Creating a Worklight native API for push notifications
- Creating and configuring an iOS native application
- Initializing WLClient and WLPush
- Subscription management
- Notification API
- Tag-based and broadcast notification

Implementation of the 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/unsubscribing to the event source.

Notification API: Server side (1 of 9)

- 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 Notifications are polled from the back end

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

```
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 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.

Notification API: Server side (2 of 9)

- Creating an event source (continued).
 - 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 for notification retrieval. The following parameters are mandatory:
 - **interval** – The polling interval in seconds.
 - **onPoll** – The polling implementation, which is an adapter function to be called at specified intervals.

Notification API: Server side (3 of 9)

- 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 the back end. In this sample, a **submitNotification** adapter function is called by a back end as an external API to send notifications.

Notification API: Server side (4 of 9)

- Sending a notification (continued).
 - 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 which it sends notification, and the **notificationText**. These arguments are provided by a back end, which calls this function.

Notification API: Server side (5 of 9)

- Sending a notification (continued).
 - Retrieve the active user and use it to get 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 };  
}
```

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(){}  
};
```

Notification API: Server side (6 of 9)

- Sending a notification (continued).
 - 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.

Notification API: Server side (7 of 9)

- Sending a notification (continued).
 - Retrieve the user subscription data.

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

  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** method. 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
    }
  }
]
```

Notification API: Server side (8 of 9)

- Sending a notification (continued).
 - Send notification to the user device or devices.

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

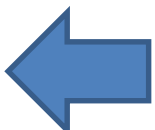
    if (userSubscription==null){
        return { result: "No subscription found for user :: " + userId };
    }

    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 };
}
```



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

Notification API: Server side (9 of 9)

- The following notification methods of the `WL.Server` class are available:
 - Use `notifyAllDevices (userSubscription, options)` to send notification to all user devices, as explained in the previous slide.
 - Use `notifyDevice (userSubscription, device, options)` to send notification to a specific device that belongs to a specific `userSubscription`.
 - Use `notifyDeviceSubscription (deviceSubscription, options)` to send the notification to a specific device.

Notification API: Client side (1 of 5)

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
{
    NSLog(@"My token from APNS : %@", deviceToken);
    _appDelegateVC.myToken = deviceToken.description;
}
```

- The token that is received by this method must be passed to the `WLPush` method. `[[WLPush sharedInstance] setTokenFromClient]:`

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


Notification API: Client side (2 of 5)

Event Source – registration

1. Register an event source within the application.
IBM Worklight Foundation provides the customizable `onReadyToSubscribe` function that is used to register an event source.
2. Set up your `onReadyToSubscribe` function in `Listener`, which implements `WLOnReadyToSubscribeListener`. This function is called 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
```

Notification API: Client side (3 of 5)

Subscribing to the event source

Prerequisite: To subscribe, a user must authenticate.

- To subscribe to the event source, use the following API.

```
- (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]` takes the following parameters:
 - An **alias**, as declared in `[[WLPush sharedInstance] registerEventSourceCallback]`
 - Optional **onSuccess** delegate
 - Optional **onFailure** delegate
- Delegates receive a response object if one is required.

Notification API: Client side (4 of 5)

Unsubscribing from an event source

- To unsubscribe from the event source, use the following API.

```
- (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]` takes the following parameters:
 - An **alias**, as declared in `WL.Client.Push.registerEventSourceCallback`
 - Optional `onSuccess` delegate
 - Optional `onFailure` delegate
- Delegates receive a response object if one is required.

Notification API Client side (5 of 5)

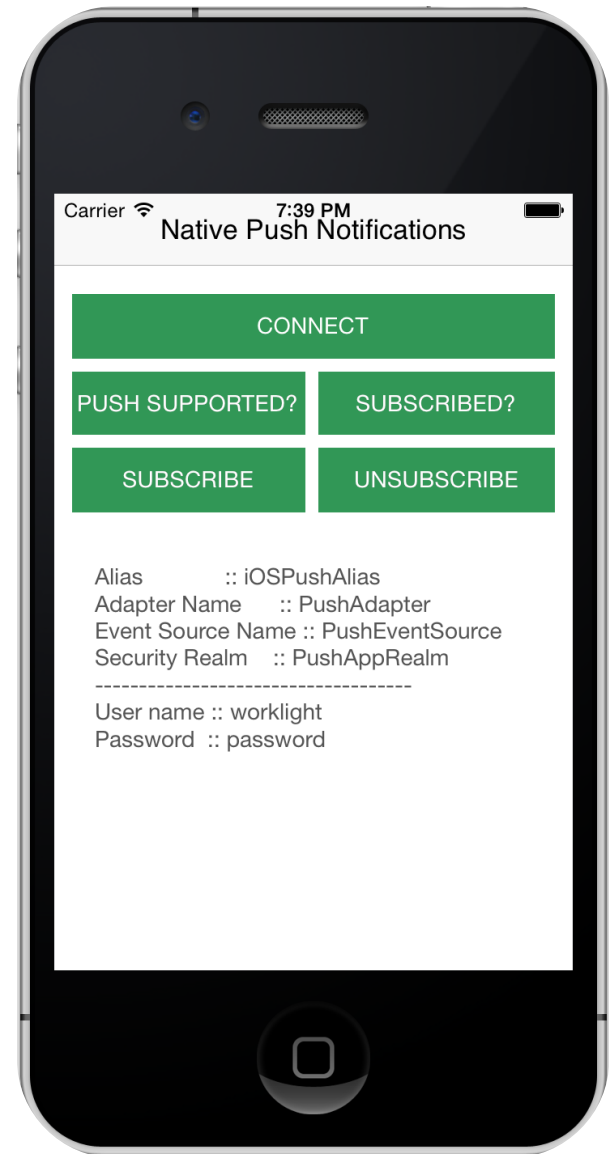
- More client side methods:
 - `[[WLPush sharedInstance] isPushSupported]` – Returns `true` if push notifications are supported by the platform, or `false` otherwise.
 - `[[WLPush sharedInstance] isSubscribed:alias]` – Returns whether the currently logged-in user is subscribed to a specified event source 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
{
    _appDelegateVC.result.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.

Receiving a procedure response

- You can find the sample for this training module in the Getting Started page of the IBM Worklight Foundation documentation website at <http://www.ibm.com/mobile-docs>.
- The sample contains two projects:
 - **PushNotificationsNative.zip** contains a Worklight native API to be deployed to your Worklight server.
 - **iOSNativePush.zip** contains a 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 the relevant server settings.



Agenda

- What are push notifications?
- Creating a Worklight native API for push notifications
- Creating and configuring an iOS native application
- Initializing WLClient and WLPush
- Subscription management
- Notification API
- Tag-based and broadcast notification

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.

Tag-based notification

- 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.

Tag-based notification

For more information about tag-based notification, see [Tag-based notification](#) in IBM Worklight Foundation user documentation.

Broadcast notification

- Broadcast notification is enabled by default for any push-enabled Worklight 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`.

Broadcast notification

For more information about broadcast notification, see [Broadcast notifications](#) in IBM Worklight Foundation user documentation.

Common API for tag-based and broadcast notifications (1 of 2)

- 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
{
    _appDelegateVC.result.text = userInfo.description;
}
```

`userInfo` - A JSON block that contains the *payload* field. This field holds other data that is sent from the Worklight 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`.

Common API for tag-based and broadcast notifications (2 of 2)

- 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 Worklight application.
 - `notificationOptions` - A JSON block that contains message properties.
- For a full list of message properties, see the IBM Worklight Foundation user documentation.

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> the sections entitled "Cookies, Web Beacons and Other Technologies" and the "IBM Software Products and Software-as-a-Service Privacy Statement" at <http://www.ibm.com/software/info/product-privacy>.

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 Developer Edition support community at:
 - <https://www.ibm.com/developerworks/mobile/worklight/connect.html>
 - If you would like a response from IBM, please provide the following information:
 - Name
 - Address
 - Company or Organization
 - Phone No.
 - Email address

Thank You

