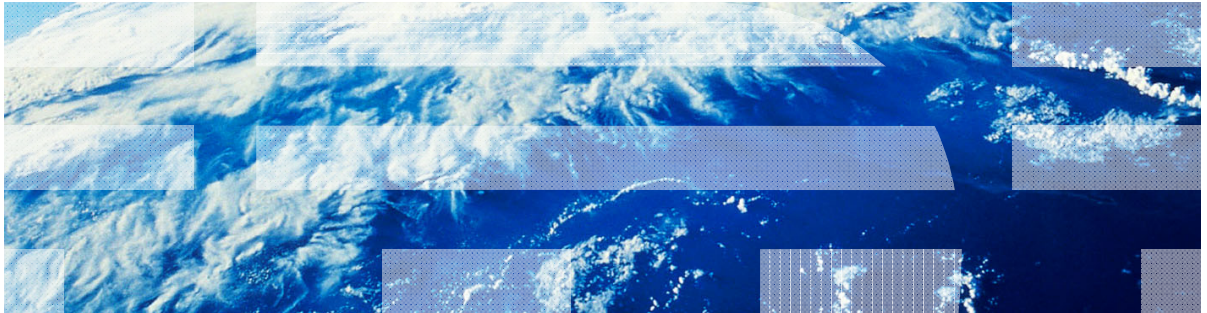


IBM Worklight V5.0.5 Getting Started

Module 41 – Push Notifications



Trademarks

- IBM, the IBM logo, and [ibm.com](http://www.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](http://www.ibm.com/legal/copytrade.shtml)” at www.ibm.com/legal/copytrade.shtml.
- Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
- 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?
- Device support
- Notification architecture
- Subscription management
- Notification API
- Setup

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 currently running
- Notifications can take several forms:
 - **Alert:** a pop-up text message
 - **Badge:** a small badge mark appearing next to the application icon
 - **Sound alert**



Agenda

- What are push notifications?
- Device support
- Notification architecture
- Subscription management
- Notification API
- Setup

Device Support

- IBM® Worklight® supports push notifications for the iOS and Android (2.2 or later) mobile platforms
- Android devices must be logged in to a Google account
- For information about push notifications for BlackBerry and Windows Phone, contact customer support

Agenda

- What are push notifications?
- Device support
- Notification architecture
- Subscription management
- Notification API
- Setup

Notification architecture

Terminology

- **Event source**
 - A push notification channel to which mobile applications can register. An event source is defined within a Worklight adapter.
- **Device token**
 - A unique identifier, obtained from the push mediator (Google or Apple), which identifies the request of a specific mobile device to receive notifications from the Worklight server.
- **User ID**
 - A unique identifier for a Worklight user. Obtained through authentication or other unique identifier such as a persistent cookie.
- **Application ID**
 - Worklight application ID. Identifies a specific Worklight application on the mobile market.

Notification architecture

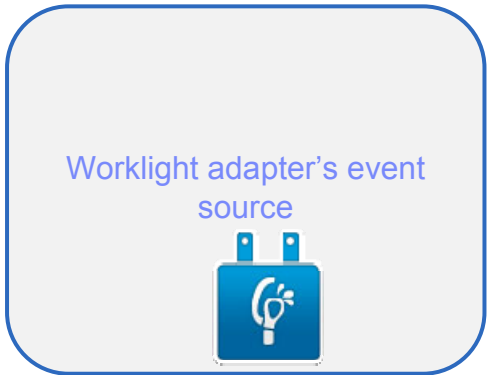
Subscription

- In order to start receiving push notifications, an application must first subscribe to a push notification *event source*
- An event source is declared in the Worklight adapter that is used by the application for push notification services
- The user must approve the push notification subscription
- When the user approves, the device registers with an Apple or Google push server to obtain a token, which is used to identify the device (“Allow notifications for application X on device Y”) and sends a subscription request to the Worklight Server. All this is performed automatically by the Worklight framework.

Notification architecture

Subscription

When the subscribe API is called, a device registers with a push services mediator and obtains a device token (done automatically by Worklight)



Notification architecture Subscription

When the token is obtained, the application subscribes to an event source

Both actions are done automatically using a single API call as described later



Notification architecture

Sending notifications

- Worklight provides a unified push notifications API

- The Adapter API allows:
 - Managing subscriptions
 - Pushing and polling notifications from back end
 - Sending push notifications to devices

- The Application API allows:
 - Subscribing to and unsubscribing from push notifications event sources
 - Handling arriving notifications

Notification architecture

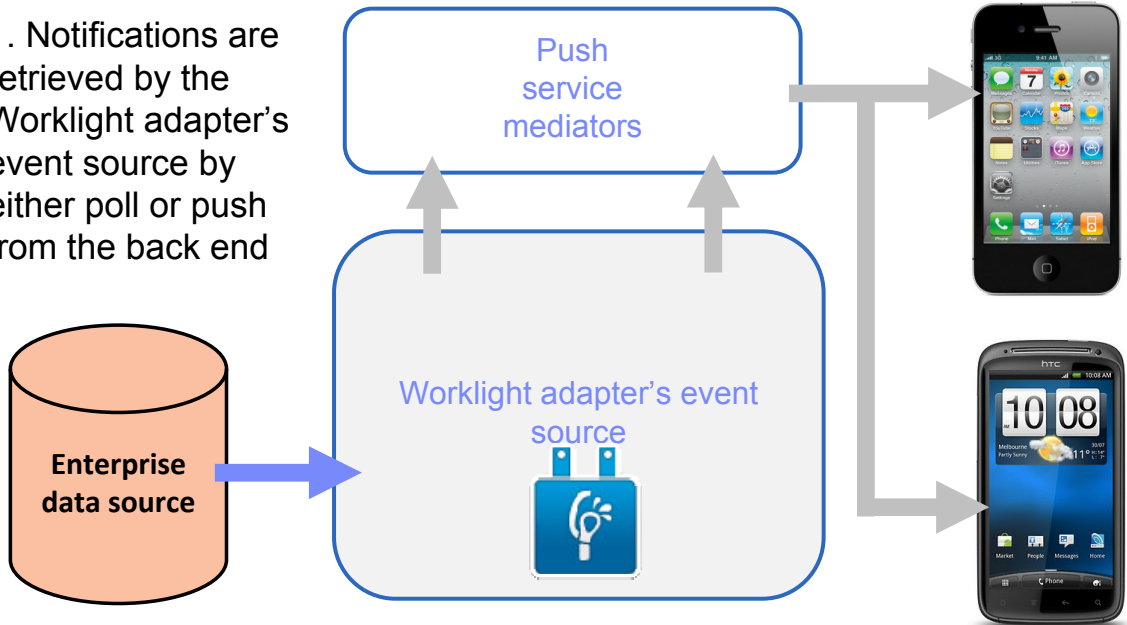
Sending notifications

- In order to send a notification, it must first be retrieved from the back end
- An event source can either poll notifications from the back end or wait for the back end to explicitly push a new notification
- When a notification is retrieved by the adapter, it is processed and sent via the corresponding push services mediator (Apple or Google)
- Additional custom logic can be added in the adapter to pre-process notifications
- The push services mediator receives the notification and sends it to a device

Notification architecture

Sending notifications

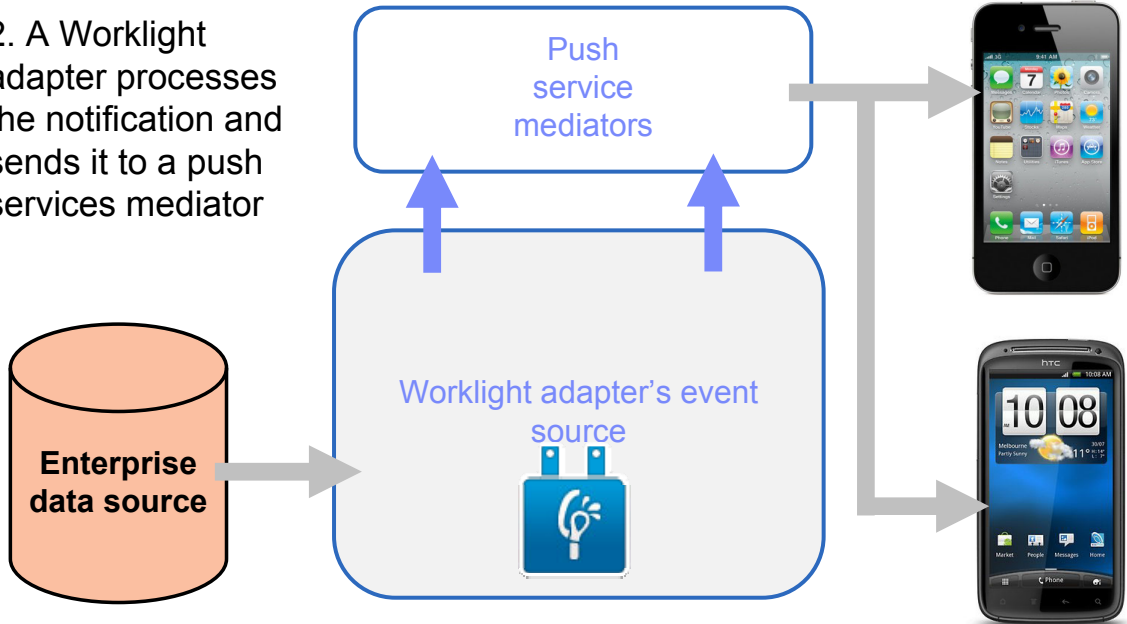
1. Notifications are retrieved by the Worklight adapter's event source by either poll or push from the back end



Notification architecture

Sending notifications

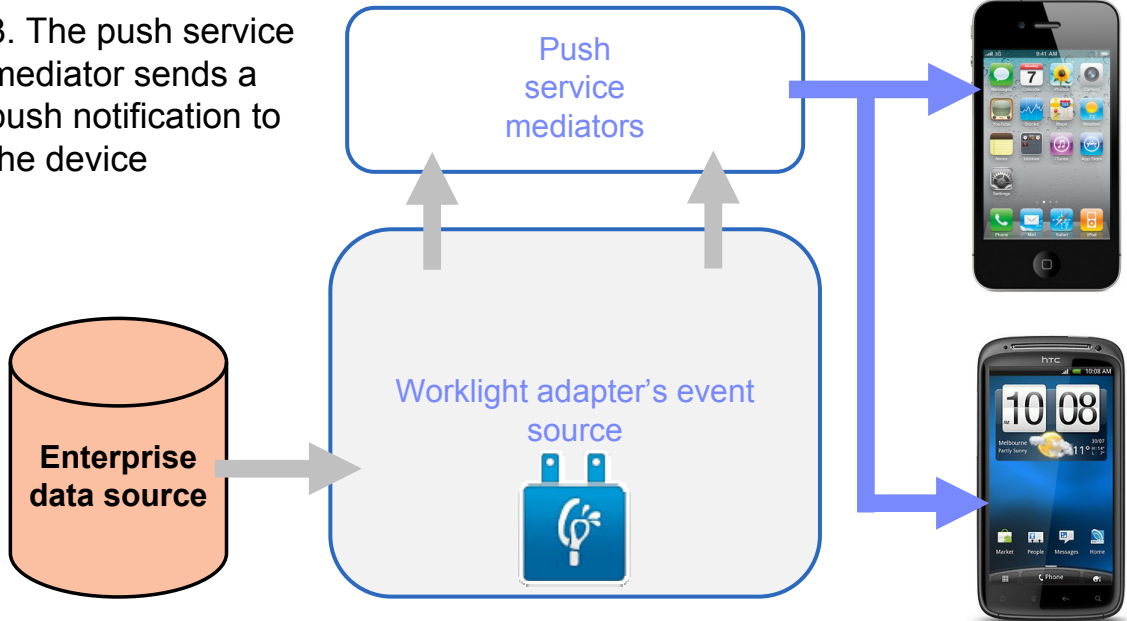
2. A Worklight adapter processes the notification and sends it to a push services mediator



Notification architecture

Sending notifications

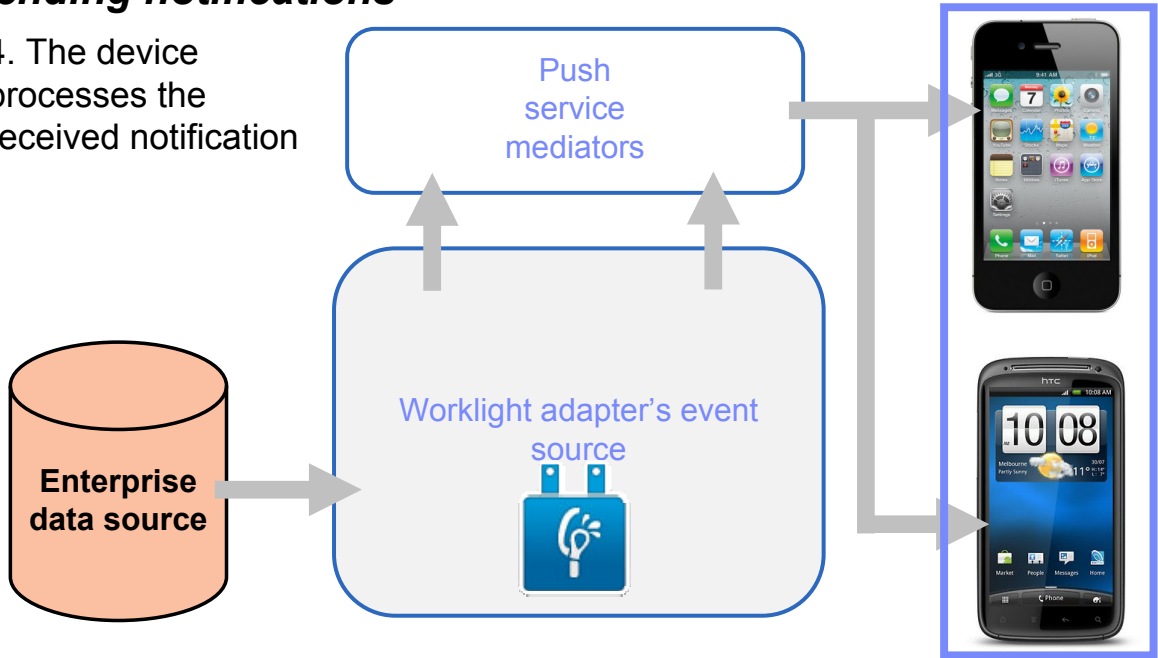
3. The push service mediator sends a push notification to the device



Notification architecture

Sending notifications

4. The device processes the received notification



Agenda

- What are push notifications?
- Device support
- Notification architecture
- **Subscription management**
- Notification API
- Setup

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

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 `WL.Client.Push.subscribe()` function
- The device subscription is deleted either by an application calling `WL.Client.Push.unsubscribe()` or when the push mediator informs the Worklight server that the device is permanently not accessible

Agenda

- What are push notifications?
- Device support
- Notification architecture
- Subscription management
- Notification API
- Setup

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

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

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.

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

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: " + 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

- Sending a notification
 - Retrieve the user subscription data

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

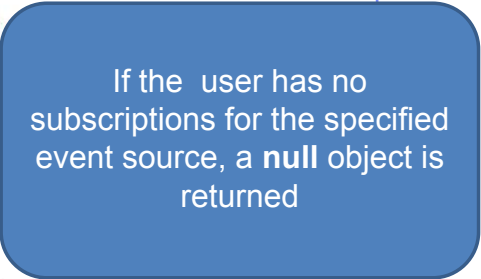
    if (userSubscription==null){
        return { result: "No subscription found for user :: " + userId };
    }

    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

Notification API: Server side

- Sending a notification
 - 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** 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
    }
  }
]
```

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);

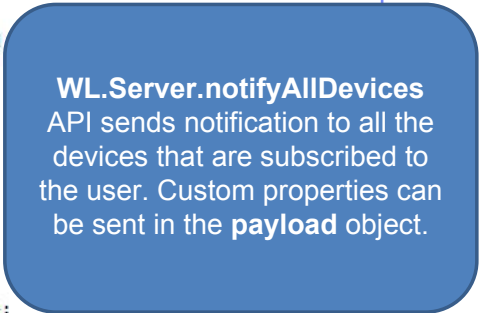
    if (userSubscription==null){
        return { result: "No subscription found for user :: " + userId };
    }

    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.

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.

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
 - Always set up your `onReadyToSubscribe` function on a global JavaScript level
 - `onReadyToSubscribe` is invoked when the authentication finishes
 - API is `WL.Client.Push.registerEventSourceCallback(alias, adapterName, eventName, callbackFunction)`

```
WL.Client.Push.onReadyToSubscribe = function(){  
    WL.Client.Push.registerEventSourceCallback(  
        "myPush",  
        "PushAdapter",  
        "PushEventSource",  
        pushNotificationReceived);  
};
```

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

```
function subscribeButtonClicked(){
    WL.Client.Push.subscribe("myPush", {
        onSuccess: pushSubscribe_Callback,
        onFailure: pushSubscribe_Callback
    });
}

function pushSubscribe_Callback(response){
    alert("PushSubscribe_Callback invoked");
}
```

- **WL.Client.Push.subscribe()** receives the following parameters:
 - An alias declared in WL.Client.Push.registerEventSourceCallback
 - Optional onSuccess callback
 - Optional onFailure callback
- Callbacks receive a response object if one is required

Notification API: Client side

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

```
function unsubscribeButtonClicked(){
    WL.Client.Push.unsubscribe("myPush", {
        onSuccess: pushUnsubscribe_Callback,
        onFailure: pushUnsubscribe_Callback
    });
}

function pushUnsubscribe_Callback(response){
    alert("pushUnsubscribe_Callback invoked");
}
```

- **WL.Client.Push.unsubscribe()** receives the following parameters:
 - An **alias** declared in WL.Client.Push.registerEventSourceCallback
 - Optional onSuccess callback
 - Optional onFailure callback
- Callbacks receive a response object if one is required

Notification API

Client side

- Additional client side APIs:
 - **WL.Client.Push.isPushSupported()** – returns true if push notifications are supported by the platform, and false otherwise
 - **WL.Client.Push.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 callback function defined in `WL.Client.Push.registerEventSourceCallback` is invoked. It receives the following arguments:

```
function pushNotificationReceived(props, payload){  
    alert("pushNotificationReceived invoked");  
    alert("props :: " + Object.toJSON(props));  
    alert("payload :: " + Object.toJSON(payload));  
}
```

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

Agenda

- What are push notifications?
- Device support
- Notification architecture
- Subscription management
- Notification API
- Setup

Setup

- To send push notifications to Android devices, you must use Google Cloud Messaging (GCM)
- You need a valid Gmail account to register to Google's GCM. In order to get a GCM key and senderId refer to:
<http://developer.android.com/guide/google/gcm/gcm.html>
- To send push notifications to iOS devices, you must use Apple Push Notifications Service (APNS)
- You must be a registered Apple iOS Developer in order to obtain an Apple APNS certificate for your application. **An APNS certificate must have a nonblank password.**
- Rename your certificate file to apns-certificate-sandbox.p12 and put it in the application root folder.

Setup

- The following servers must be accessible from a Worklight Server in order to send push notifications:
- iOS:
 - Sandbox servers:
 - gateway.sandbox.push.apple.com:2195
 - feedback.sandbox.push.apple.com:2196
 - Production servers:
 - gateway.push.apple.com:2195
 - Feedback.push.apple.com:2196
- Android:
 - <https://www.google.com>
 - <https://android.apis.google.com>

Setup

- To set up push notifications in an app, add the following lines to the `application-descriptor.xml` file
- Android

```
<android version="1.0" securityTest="PushApplication-strong-mobile-securityTest">  
  <pushSender key="Dummy_API_key" senderId="Dummy_GCM_sender_ID"/>  
</android>
```

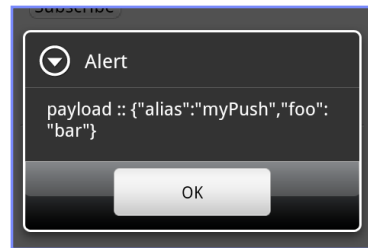
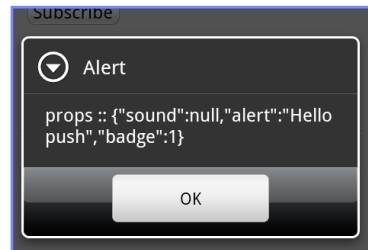
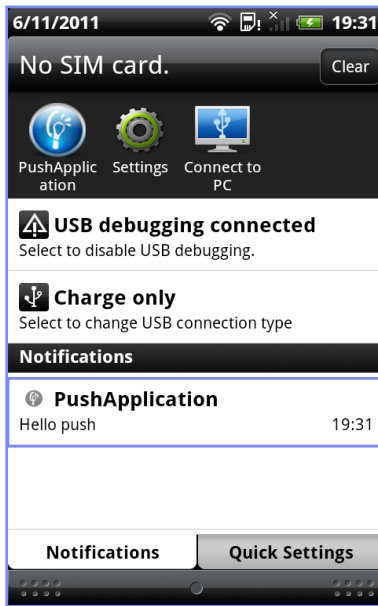
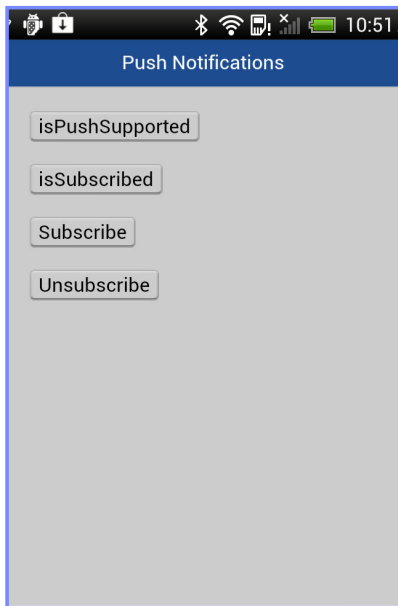
- iOS

```
<iphone version="1.0">  
  <pushSender password="certificate_password"/>  
</iphone>
```

- The Apple APNS certificate file must be placed in the root of the application's folder
- Replace `pushSender` settings values with your credentials

The Result

- 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>



Back end emulator

- The sample for this training module comes with a back end emulator which can be used to simulate push notification submission by a back end
- To run the back end emulator open the PushBackendEmulator folder of the sample project in a command prompt
- Run the supplied JAR file using the following syntax:

```
java -jar PushBackendEmulator.jar <userId> <notificationText> <serverPort>
```

- Where userId is the user name you used to login to the sample application
- For example

```
java -jar PushBackendEmulator.jar user1 "hello from push" 8080
```


Back end emulator

- The back end emulator will try to connect to a Worklight Server and call a submitNotification() adapter procedure
- It will output the invocation result to a command prompt console
- Success

```
d:\dev\git\WorklightTraining\module_21_0_PushNotifications\PushBackendEmulator>
java -jar PushBackendEmulator.jar aaa "hello push"
PushBackendEmulator
User Id: aaa
Notification text: hello push
Server URL: http://localhost:8080
sending notification
Server response :: {   "isSuccessful": true,   "result": "Notification sent to
user :: aaa"}
```

- Failure

```
d:\dev\git\WorklightTraining\module_21_0_PushNotifications\PushBackendEmulator>
java -jar PushBackendEmulator.jar asdf "hello push"
PushBackendEmulator
User Id: asdf
Notification text: hello push
Server URL: http://localhost:8080
sending notification
Server response :: {   "isSuccessful": true,   "result": "No subscription found
for user :: asdf"}
```

Check yourself questions

- Which of the below connections are mandatory for push notifications to work?
 - Client application should be able to connect to an APNS/GCM server
 - Client application should be able to connect to a Worklight server
 - Worklight server should be able to connect to an APNS/GCM server
 - All of above
- A user has several devices. He is using the application from all of them. Is it possible to send a push notification to a specific device?
 - No, when Worklight server submits a push notification it will be delivered to all of the user's devices
 - Yes, userSubscription contains deviceSubscription objects for each device the user ever registered for push notifications from. It can be used to send notification to a specific device.
 - Yes, when Worklight server submits push notification it will be delivered to the last device that a user logged in on
 - No, this is considered a security breach. If the user has more than one device, notification will not be sent at all
- Can application-related custom data be sent in a push notification?
 - No, a push notification may only contain notification text that will be shown to the user
 - Yes, application-related custom data can be sent in the push notification. The application must be running in the foreground in order to receive it.
 - Yes, application-related custom data can be sent in the push notification. The application must be running in the background in order to receive it.
 - Yes, application-related custom data can be sent in the push notification. The Application can receive it even if it was not running at the time of push notification arrival.

Check yourself questions

- Which of the below connections are mandatory for push notifications to work?
 - Client application should be able to connect to an APNS/GCM server
 - Client application should be able to connect to a Worklight server
 - Worklight server should be able to connect to an APNS/GCM server
 - All of above
- A user has several devices. He is using the application from all of them. Is it possible to send a push notification to a specific device?
 - No, when Worklight server submits push notification it will be delivered to all of the user's devices
 - Yes, userSubscription contains deviceSubscription objects for each device the user ever registered for push notifications from. It can be used to send notification to a specific device.
 - Yes, when Worklight Server submits push notification it will be delivered to a last device that a user logged in on
 - No, this is considered a security breach. In case user have more than one device, notification will not be sent at all
- Can application-related custom data be sent in a push notification?
 - No, a push notification may only contain notification text that will be shown to user
 - Yes, application-related custom data can be sent in the push notification. The application must be running in the foreground in order to receive it.
 - Yes, application-related custom data can be sent in the push notification. The application must be running in the background in order to receive it.
 - Yes, application-related custom data can be sent in the push notification. The application can receive it even if it was not running at the time of push notification arrival.

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.

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

Thank You

