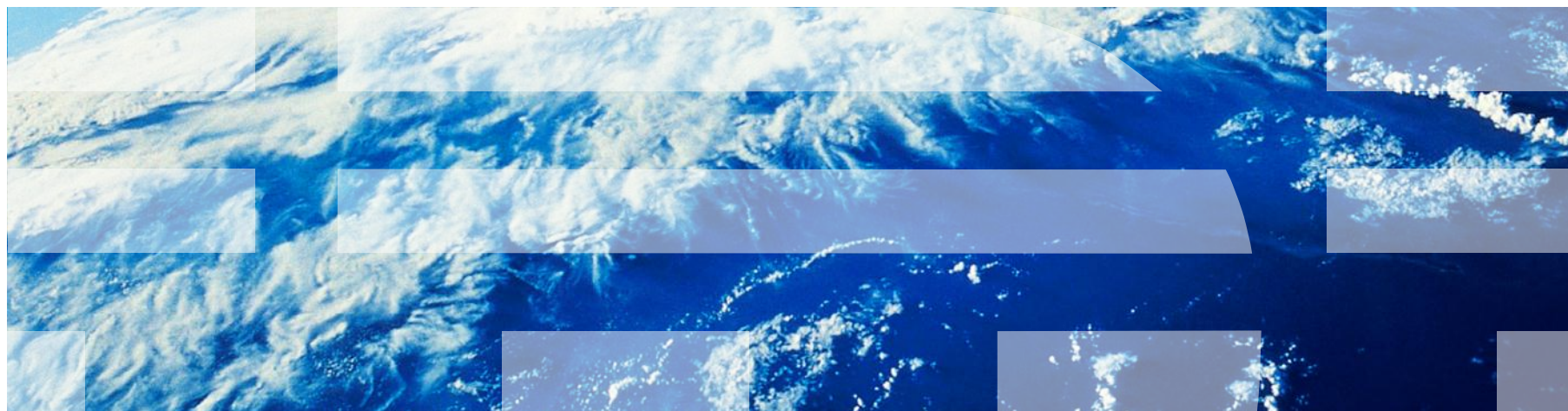# IBM Worklight V6.0.0
Getting Started

Push Notifications

26 June 2014

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.

- 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

  – Server side

  – Client side

- Setup

  – Guidelines

  – Project configuration

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 (and are platform-dependent):

  - **Alert**: a pop-up text message

  - **Badge, Tile**: a graphical representation that allows a short text or image

  - **Banner, Toast:** a disappearing pop-up text message at the top of the device display

  - **Sound alert**

Agenda

- What are push notifications?

- Device support

- Notification architecture

- Subscription management

- Notification API

  – Server side

  – Client side

- Setup

  – Guidelines

  – Project configuration

Device Support

- IBM Worklight® supports push notifications for the following mobile platforms:

  - Android (2.2 or later)

  - iOS

  - Windows Phone 8

Agenda

- What are push notifications?

- Device support

- Notification architecture

- Subscription management

- Notification API

  – Server side

  – Client side

- Setup

  – Guidelines

  – Project configuration

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 (Apple, Google, or Microsoft), 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.

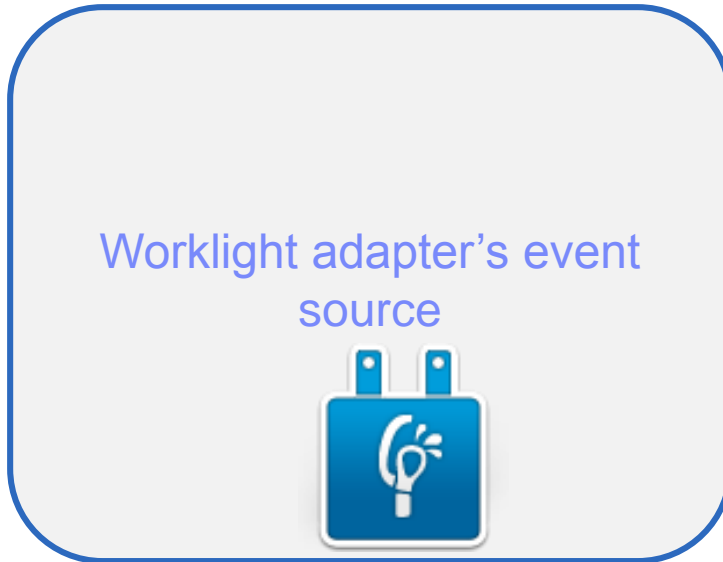# *Notification architecture*

Subscription

- 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, Google, or Microsoft push server to obtain a token that is used to identify the device ("Allow notifications for application X on device Y"), and sends a subscription request to the Worklight Server.

  - This operation is performed automatically by the IBM Worklight framework.

Notification architecture

Subscription

When the subscribe API is called, a device registers with a push service mediator and obtains a device token (done automatically by IBM Worklight).
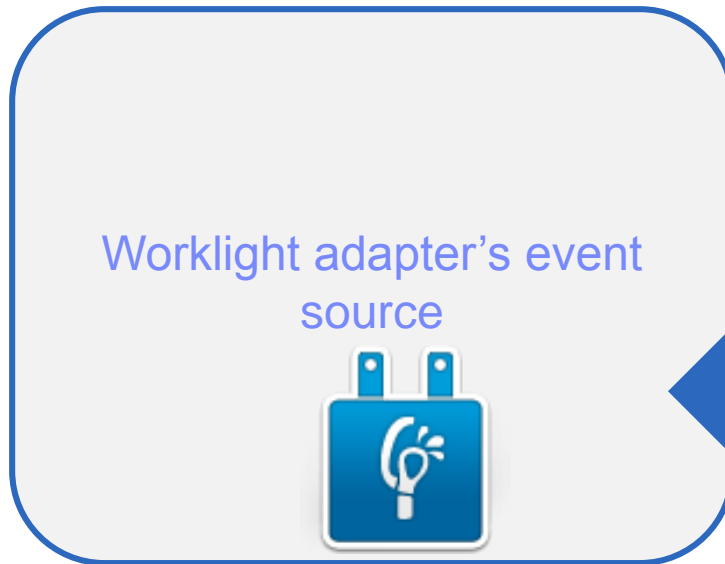
Push service mediators

Worklight adapter's event source

Notification architecture

Subscription

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

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

Push service mediators

Worklight adapter's event source

Notification architecture

Sending notifications

- IBM Worklight provides a unified push notifications API.

- The Adapter API allows:

  - Managing subscriptions

  - Pushing and polling notifications from backend

  - Sending push notifications to devices

- The Application API allows:

  - Subscribing to and unsubscribing from push notifications event sources

  - Handling arriving notifications
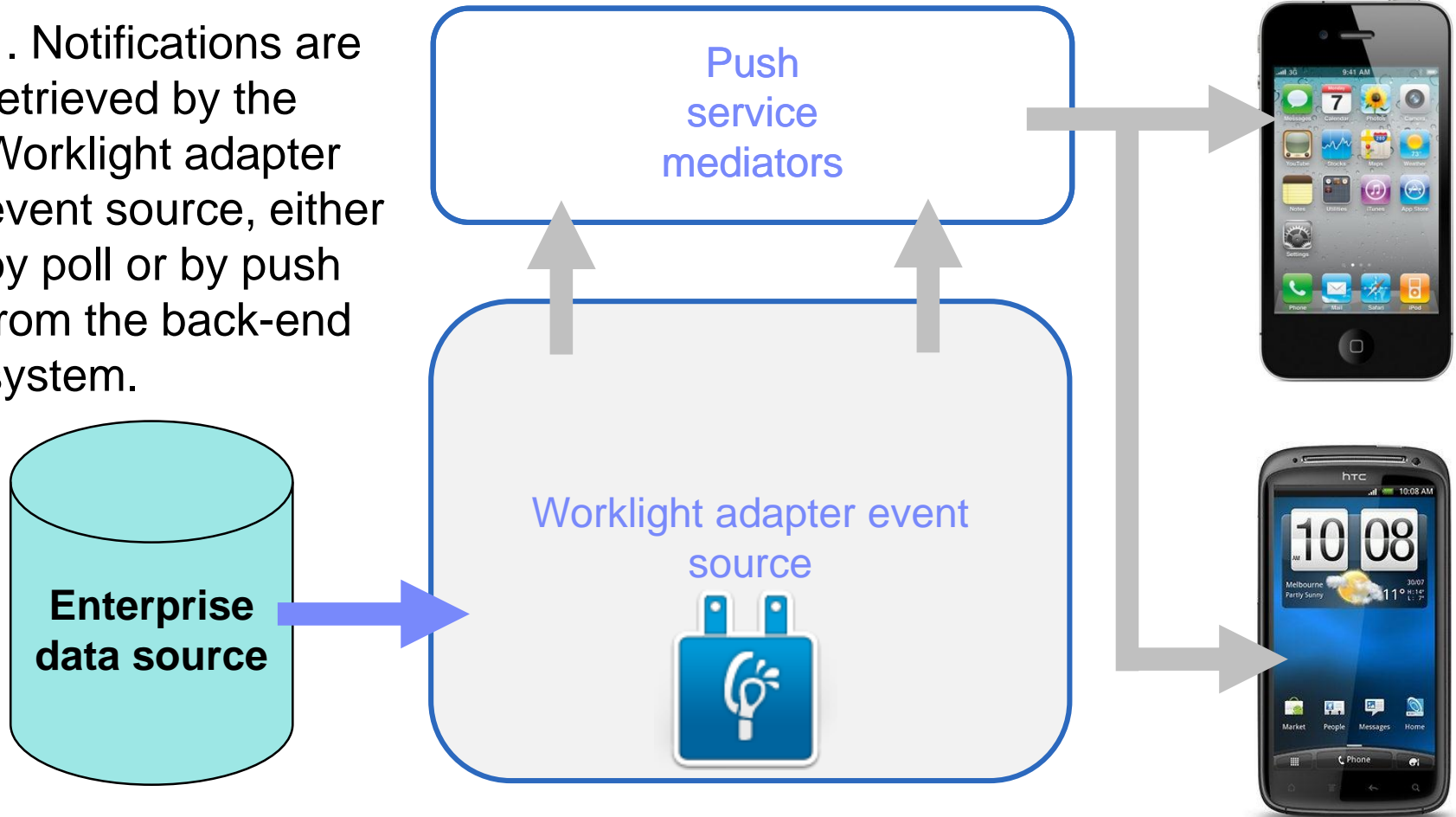
Notification architecture

Sending notifications

- To send a notification, this notification must first be retrieved from the backend.

- An event source can either poll notifications from the back-end system , or wait for the back-end system to explicitly push a new notification.

- When a notification is retrieved by the adapter, it is processed and sent through the corresponding push service mediator (Apple, Google, or Microsoft).

- Additional custom logic can be added in the adapter to pre-process notifications.

- The push service mediator receives the notification and sends it to a device.

Notification architecture
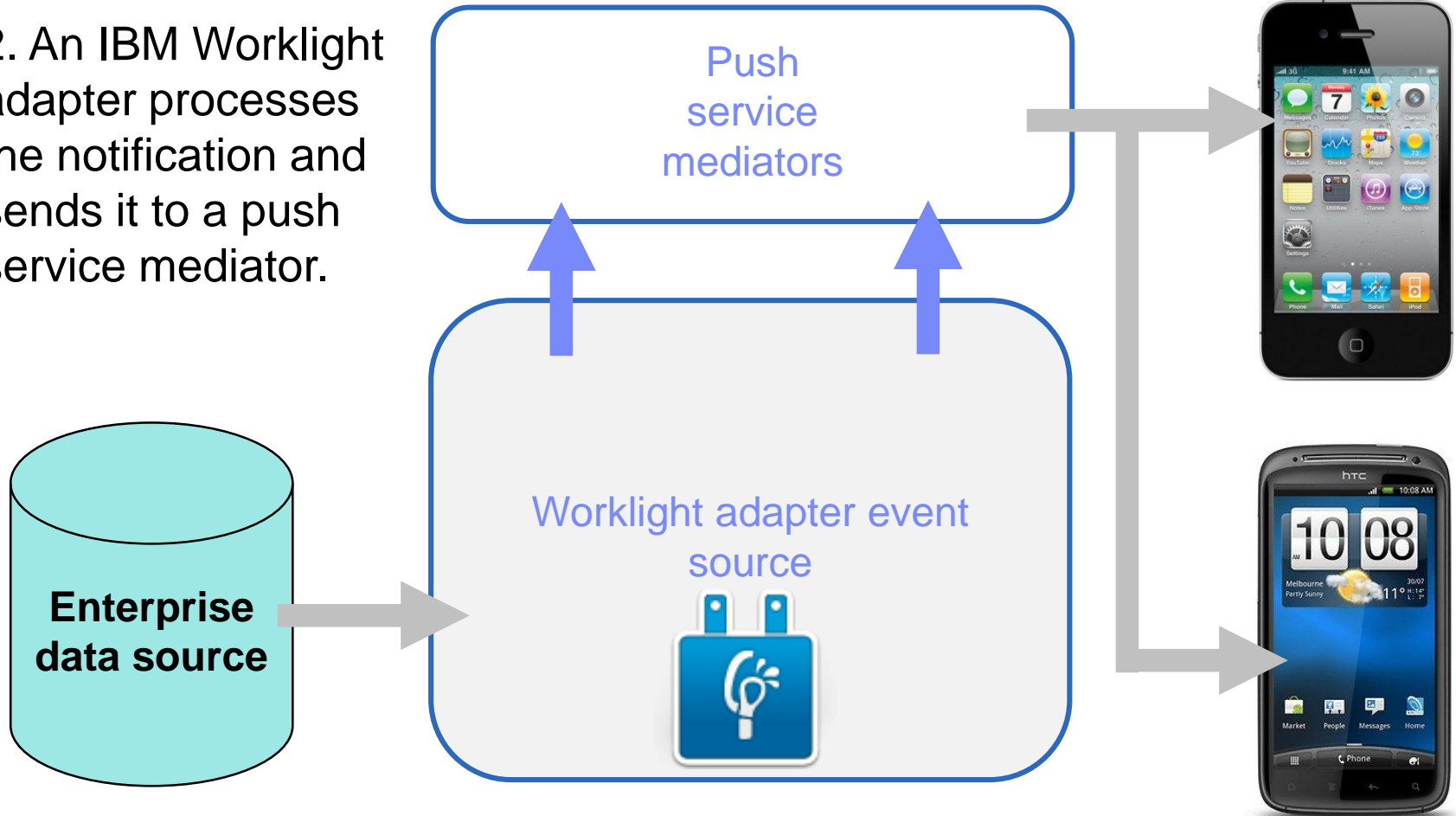
Sending notifications

1. Notifications are retrieved by the Worklight adapter event source, either by poll or by push from the back-end system.

Enterprise data source

Push service mediators

Worklight adapter event source

Notification architecture

Sending notifications

2. An IBM Worklight adapter processes the notification and sends it to a push service mediator.

Push service mediators

Worklight adapter event source

Enterprise data source

Notification architecture

Sending notifications

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

Push service mediators

Worklight adapter event source

Enterprise data source
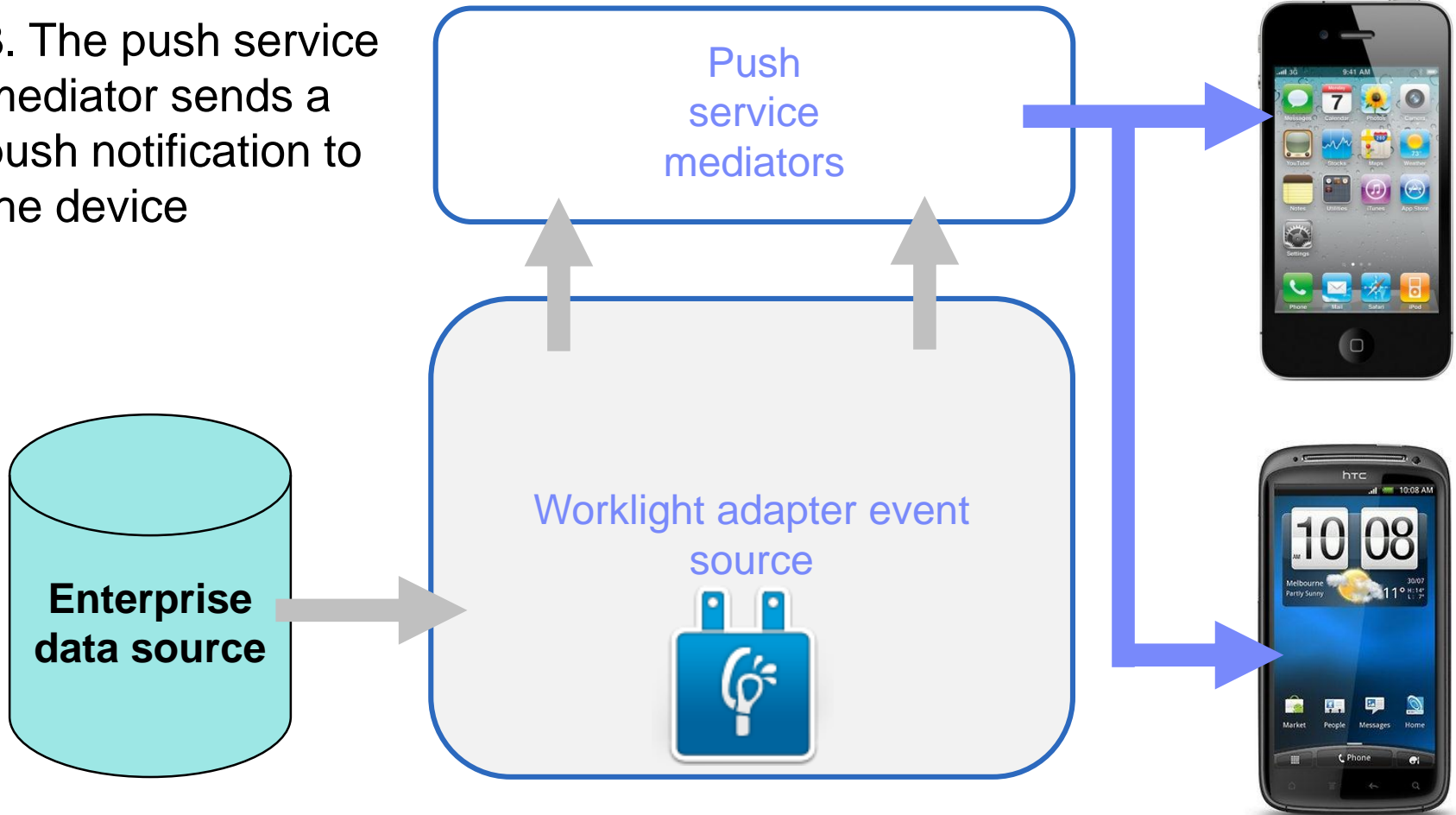
Notification architecture

Sending notifications

4. The device processes the received notification
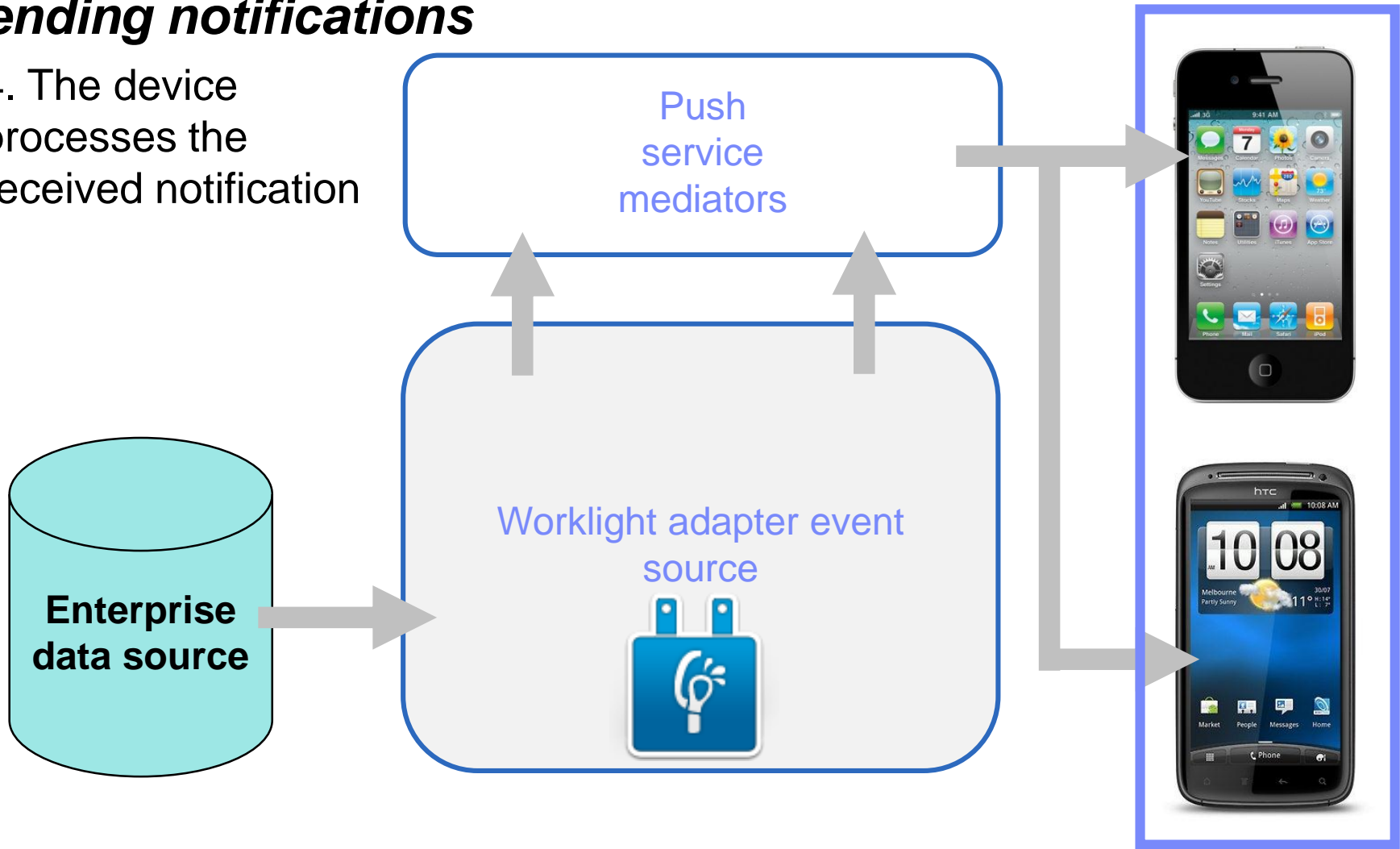
Push service mediators

Worklight adapter event source

Enterprise data source

Agenda

- What are push notifications?

- Device support

- Notification architecture

- Subscription management

- Notification API

  – Server side

  – Client side

- Setup

  – Guidelines

  – Project configuration

Subscription management

User subscription

- **User subscription**
  - An entity that contains a user ID, a device ID and an 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 first subscribes to the event source from any device.

- **Deletion**
  - A user subscription is deleted when the user unsubscribes from the event source from all 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 that 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 **WL.Client.Push.subscribe()**.

- The device subscription is deleted either by an application that calls **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
  - Server side
  - Client side

- Setup
  - Guidelines
  - Project configuration

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 the user subscription request is received
- **onDeviceUnsubscribe** – an adapter function that is invoked when the user unsubscribe request is received
- **securityTest** – a security test from **authenticationConfig.xml** that is 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 backend

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

Notifications are polled from the backend

```
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

- Send 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

    var notification = WL.Server.createDefaultNotification(notif

    WL.Server.notifyAllDevices(userSubscription, notification);

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

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

Notification API - Server side

- Send 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 :: " +
    }

    var deviceSubscriptions =
        userSubscription.getDeviceSubscriptions();

    WL.Logger.debug("submitNotification >> userId :: " + userId                        xt);

    var notification = WL.Server.createDefaultNotification(not

    WL.Server.notifyAllDevices(userSubscription, notification)

    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 system that invokes this function.

Notification API - Server side

- Send a notification:

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

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

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

    var deviceSubscriptions =
        userSubscription.getDeviceSubscri

    WL.Logger.debug("submitNotification >                                      t);

    var notification = WL.Server.createDe

    WL.Server.notifyAllDevices(userSubscr

    return { result: "Notification sent t
}
```
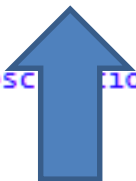
A user subscription object contains the information about all of the user's subscriptions. Each user subscription can have several device subscriptions.
The object structure is as follows:

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

Notification API - Server side

- Send 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 :: "       t);

    var notification = WL.Server.createDefaultNotificat

    WL.Server.notifyAllDevices(userSubscription, notific

    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

- Send a notification:

  - Retrieve the user subscription data

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

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

    var deviceSubscriptions =
        userSubscription.getDeviceSubscriptions();

    WL.Logger.debug("sub    otification >> userId :: " + user                    t);

    var notification = W    erver.createDefaultNotification(no

    WL.Server.notifyAllDevices(userSubscription, notification

    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

- Send 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
    }

    var deviceSubscriptions =
        userSubscription.getDeviceSubscriptions(

    WL.Logger.debug("submitNotification >> userId        text          notificationText);

    var notification = WL.Server.createDefaultNotification(notificationText, 3, {foo : 'bar'});

    WL.Server.notifyAllDevices(userSubscription   notification);

    return { result: "Notification sent to use   : " + userId };
}
```
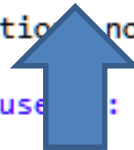
WL.Server.createDefaultNotification API creates and returns a default notification JSON block for the supplied values.

Notification API - Server side

- Send 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 + ",                    t);

    var notification = WL.Server.createDefaultNotification(notificationText, 3, {foo : 'bar'});

    WL.Server.notifyAllDevices(userSubscription, notification);

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

The **Text** to be pushed to the device

Notification API - Server side

- Send 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
                                                                              xt);

    var notification = WL.Server.createDefaultNotification(notificationText, 3, {foo : 'bar'});

    WL.Server.notifyAllDevices(userSubscription, notification);

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

Badge number that will be displayed on the application icon or tile (in environments that support it)

Notification API - Server side

- Send 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 use

    }

    var deviceSubscriptions =
        userSubscription.getDeviceSubscriptions();

    WL.Logger.debug("submitNotification >> userId :: " + userId + ", text :: " + notificationText);

    var notification = WL.Server.createDefaultNotification(notificationText, 3, {foo : 'bar'});

    WL.Server.notifyAllDevices(userSubscription, notification);

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

Payload is a JSON object that is transferred to the application and that can contain custom properties.

Notification API - Server side

- Send 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 ::
    }

    var deviceSubscriptions =
        userSubscription.getDeviceSubscriptions();

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

    var notification = WL.Server.createDefaultNotification(notificationText, 3, {foo : 'bar'});

    WL.Server.notifyAllDevices(userSubscription, notification);

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

> **WL.Server.notifyAllDevices** API sends notification to all the devices that are subscribed to the user.

Notification API - Server side

- Several APIs exist for notification sending.

  - Use **WL.Server.notifyAllDevices(`userSubscription, options`)** to send notification to all a user's devices (see previous slide).

  - Use **WL.Server.notifyDevice(`userSubscription, device, options`)** to send notification to a specific device that belongs to a specific userSubscription.

  - Use **WL.Server.notifyDeviceSubscription(`deviceSubscription, options`)** to send the notification to a specific device.

Agenda

- What are push notifications?

- Device support

- Notification architecture

- Subscription management

- Notification API

  – Server side

  – Client side

- Setup

  – Guidelines

  – Project configuration

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.

  – The API is: **WL.Client.Push.registerEventSourceCallback(`alias`, `adapterName`, `eventSourceName`, `callbackFunction`)**

```javascript
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 to subscribe.
  - Use the following API to subscribe to the event source:

```javascript
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** that is 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** that is 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 API methods:

  - **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 a 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 function is invoked when the application returns to foreground.

Agenda

- What are push notifications?

- Device support

- Notification architecture

- Subscription management

- Notification API

  – Server side

  – Client side

- Setup

  – Guidelines

  – Project configuration

Setup - Guidelines

- Setting up push notifications depends on the platforms, as indicated in this slide and subsequent ones.

Android

- To send push notifications to Android devices, you must use the Google Cloud Messaging (GCM) service.

- You need a valid Gmail account to register to Google's GCM.
  - For more information about how to get a GCM key and a **senderId** (to use in the Worklight project), see http://developer.android.com/guide/google/gcm/gs.html

- Android OS 2.2.x/2.3.x devices must be synchronized with a Gmail account.

- Android OS 4.x does not impose Gmail account synchronization.

Setup - Guidelines

iOS

- To send push notifications to iOS devices, you must use the Apple Push Notifications Service (APNS).

- You must be a registered Apple iOS Developer to obtain an Apple APNS certificate for your application. An APNS certificate must have a non-blank password.

- When you are in development, rename your certificate file to **apns-certificate-sandbox.p12** and place it in the environment root folder or in the application root folder. **The environment root folder takes the highest priority.**

- When you move to production, rename your certificate file to **apns-certificate-production.p12** and place it in the environment root folder or in the application root folder. **The environment root folder takes the highest priority.**

- When the hybrid application has both iPhone and iPad environments, it requires separate certificates for push notification. In that case, place those certificates in the corresponding environment folders.

Setup - Guidelines

Windows Phone 8

- To send push notifications to Windows Phone 8 devices, you must use the Microsoft Push Notifications Service (MPNS).

- <u>Non-authenticated</u> push notification does not require any setup from the developer.

- <u>Authenticated</u> push notification requires a Windows Phone Dev Center account.

- To use authenticated push, you must use a certificate that is issued by a Microsoft-trusted root certificate authority.

- *For production, consider using authenticated push notification to ensure that your information is not comprised. Consider using non-authenticated push only for development-time.*

Setup - Guidelines

- The following servers must be accessible from a Worklight Server 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**:

  – The ports to open are: 5228, 5229, and 5230. GCM typically only uses 5228, but it sometimes uses 5229 and 5230.

  – GCM doesn't provide specific IP addresses, so you should allow your firewall to accept outgoing connections to all IP addresses contained in the IP blocks listed in Google's ASN of 15169.

Setup - Guidelines

Windows Phone 8:

- No specific ports needs to be open in your server configuration. MPNS uses regular `http` or `https` requests.

Agenda

- What are push notifications?

- Device support

- Notification architecture

- Subscription management

- Notification API
  - Server side
  - Client side

- Setup
  - Guidelines
  - Project configuration

Setup – Project configuration

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

- These settings are also editable with the Application Descriptor Editor in Design mode.

- **Android:**

```
<android version="1.0">
    <pushSender key="GCM_key" senderId="GCM_ID"/>
```

  – Replace **GCM_Key** and **GCM_ID** with the values as received from GCM.

- **iOS:**

```
<iphone bundleId="com.PushNotificationsApp" version="1.0">
    <pushSender password="certificate_password"/>
```

  – Place the Apple APNS certificate file at the root of the application folder or environment folder (see slide 47).

  – Replace **certificate password** with your actual certificate password.

Setup – Project configuration

- **Windows Phone 8:**

- To set up non-authenticated push:

```
<windowsPhone8 version="1.0">
    <uuid>11578325-b204-42f5-acac-895430ffa09d</uuid>
    <pushSender/>
</windowsPhone8>
```

- To set up authenticated push:

```
<windowsPhone8 version="1.0">
    <uuid>11578325-b204-42f5-acac-895430ffa09d</uuid>
    <pushSender>
        <authenticatedPush keyAlias="myAliasName" keyAliasPassword="myAliasPassword" serviceName="certificateCommonName"/>
    </pushSender>
</windowsPhone8>
```

- For more information about how to use the certificate file in the Worklight project, follow the instructions that are provided in the IBM Worklight user documentation, under the "Windows Phone 8 push notifications" topic.

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
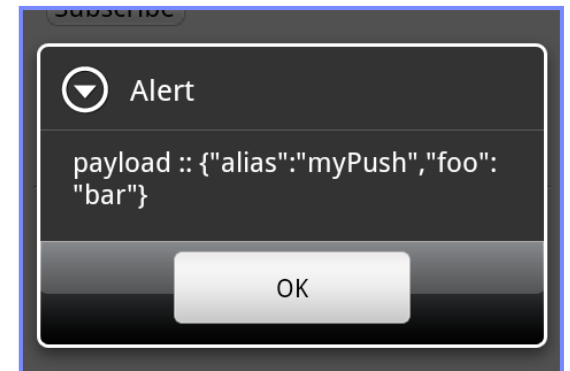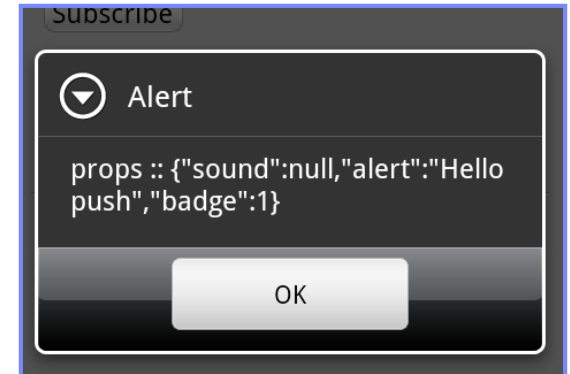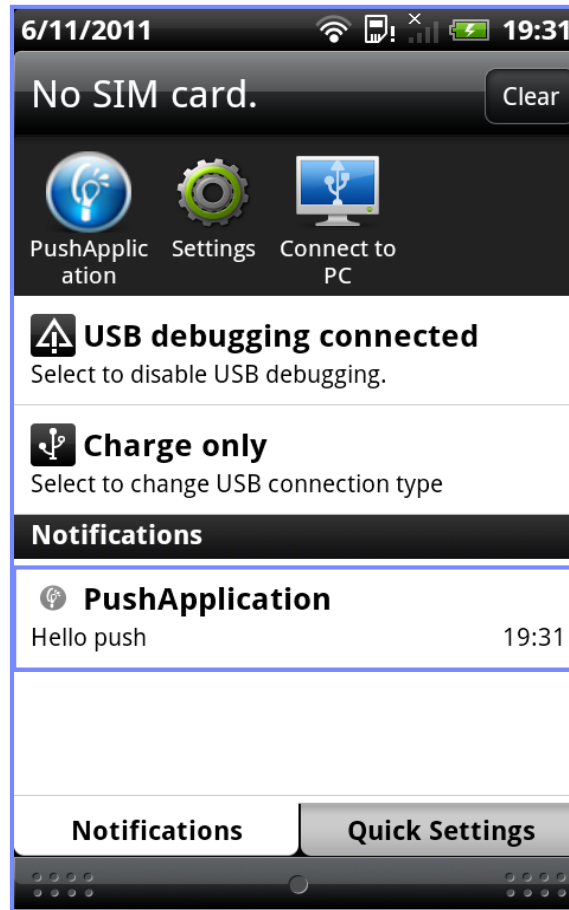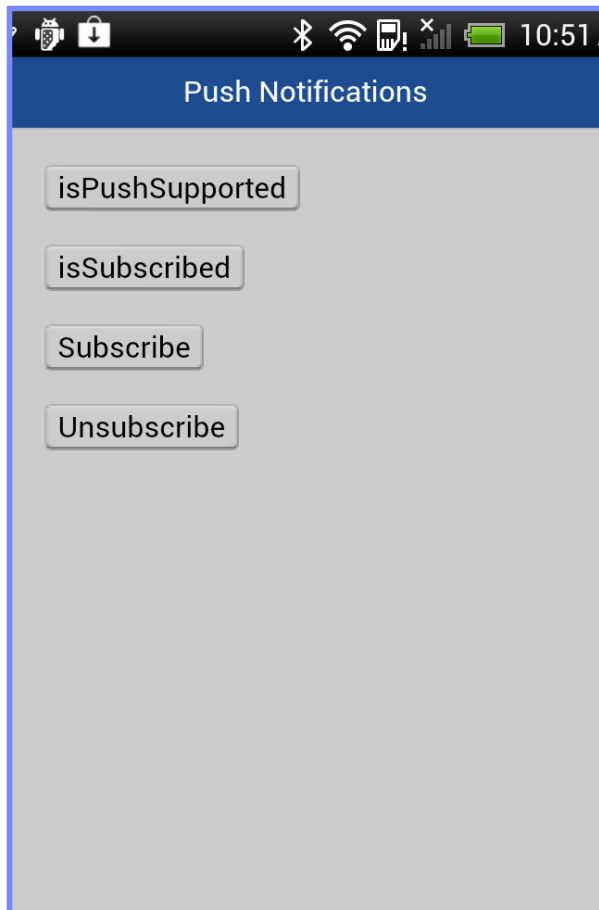
Back end emulator

- The sample for this training module comes with a backend emulator that you can use to simulate push notification submissions by a back-end system.

- To run the backend emulator, open the **PushBackendEmulator** folder of the sample project in a command prompt, and then run the supplied JAR file by using the following syntax:

```
java –jar PushBackendEmulator.jar <userId> <notificationText> <context root> <serverPort-optional>
```

- **userId** is the user name you used to login to the sample application.

- **contextRoot** is the context root of your Worklight project

- For example:

```
java –jar PushBackendEmulator.jar JohnDoe "My first push notification" myContextRoot 10080
```

Backend emulator

- The backend 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:

```
c:\>java -jar C:\PushBackendEmulator.jar JohnDoe "hello push" PushNotificationsProject 10080
PushBackendEmulator
User Id: JohnDoe
Notification text: hello push
Server URL: http://localhost:10080/PushNotificationsProject
sending notification
Server response :: {    "isSuccessful": true,    "result": "Notification sent to user :: JohnDoe"}
```

- Failure:

```
c:\>java -jar C:\PushBackendEmulator.jar JohnMissing "hello push" PushNotificationsProject 10080
PushBackendEmulator
User Id: JohnMissing
Notification text: hello push
Server URL: http://localhost:10080/PushNotificationsProject
sending notification
Server response :: {    "isSuccessful": true,    "result": "No subscription found for user :: JohnMissing"}
```

Check yourself questions

- Which of the following connections are mandatory for push notifications to work?
  - Client application should be able to connect to an APNS/GCM/MPNS server.
  - Client application should be able to connect to a Worklight server.
  - Worklight server should be able to connect to an APNS/GCM/MPNS server.
  - All of them
- A user has several devices, and 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 is delivered to all of the user's devices.
  - Yes, **userSubscription** contains **deviceSubscription** objects for each device that the user 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 to.
  - No, doing so 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 might 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 to receive it.
  - Yes, application-related custom data can be sent in the push notification. The application must be running in the background 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 that the push notification arrived.

Check yourself questions

- Which of the following connections are mandatory for push notifications to work?
  - Client application should be able to connect to an APNS/GCM/MPNS server.
  - Client application should be able to connect to a Worklight server.
  - Worklight server should be able to connect to an APNS/GCM/MPNS server.
  - All of them
- A user has several devices, and 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 is delivered to all of the user's devices.
  - Yes, **userSubscription** contains **deviceSubscription** objects for each device that the user 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 to.
  - No, doing so 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 might 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 to receive it.
  - Yes, application-related custom data can be sent in the push notification. The application must be running in the background 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 that the push notification arrived.

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