IBM Worklight Foundation

# IBM Worklight Foundation V6.2.0

## C# client-side API for native Windows Phone 8 apps

20 June 2014

## About IBM

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

# Contents

## Tables

## About this document

This document is intended for Windows Phone 8 developers who want to access IBM® Worklight® Foundation services from Windows Phone 8 applications written in C#. The document guides you through the available classes and methods.

# 1 API overview

The IBM Worklight Foundation C# client-side API for native Windows Phone 8 applications exposes two main capabilities:

- Calling back-end services to retrieve data and perform back-end transactions.
- Writing custom log lines for reporting and auditing purposes.

The IBM Worklight Foundation C# client-side API for native Windows Phone 8 applications is available as part of the Worklight Studio.

| Type | Name | Description | Implemented By |
|---|---|---|---|
| Properties file | `worklight.properties` | Properties file that contains the necessary data to use the Worklight SDK. | IBM |
| Class | `WLClient` | Singleton class that exposes methods to communicate with the Worklight Server, in particular `invokeProcedure` for calling a back-end service. | IBM |
| Class | `WLProcedure InvocationData` | Class that contains all data necessary to call a procedure. | IBM |
| Class | `WLRequestOptions` | Class that you use to add request parameters, headers, and invocation context. | IBM |
| Interface | `WLResponseListener` | Interface that defines methods that a listener for the `WLClient invokeProcedure` method implements to receive notifications about the success or failure of the method call. | Application developer |
| Class | `WLResponse` | Class that contains the result of a procedure invocation. | IBM |
| Class | `WLFailResponse` | Class that extends `WLResponse`. This class contains error codes, messages, and the status in `WLResponse`. This class also contains the original response DataObject from the server. | IBM |

| Type | Name | Description | Implemented By |
|---|---|---|---|
| Class | `WLProcedureInvocationResult` | Class that extends `WLResponse`. This class contains the result of calling a back-end service, which includes statuses and data items that the adapter function retrieves from the server. | IBM |
| Class | `WLProcedureInvocationFailResponse` | Class that extends `WLFailResponse` and that you can use to retrieve the invocation error messages. | IBM |
| Class | `WLErrorCode` | Class that contains an error code and its message that arrive from the Worklight Server. | IBM |
| Class | `BaseChallengeHandler` | This class is an abstract base class for all Challenge Handlers. | IBM |
| Class | `ChallengeHandler` | This class is an abstract class that you must extend to create custom challenge handlers. | Application Developer |
| Class | `WLChallengeHandler` | This class is an abstract base class for Worklight Challenge Handlers. You must extend it to implement your own version of a Worklight Challenge Handler. | IBM |

*Table 1-1: IBM Worklight Foundation C# client-side API for Windows Phone 8 applications – packages, classes, interfaces, and files*

# 2 API reference

## 2.1 Example Code

The following code samples show how to use the IBM Worklight Foundation C# client-side API for native Windows Phone 8 applications.

### 2.1.1 Example: connecting to the Worklight Server and calling a procedure

#### Initializing the Worklight Client

```
WLClient client = WLClient.getInstance();
client.connect(new MyConnectResponseListener());
```

#### Implementation of a Response Listener for connect

```
public class MyConnectResponseListener : WLResponseListener{
  public void onFailure(WLFailResponse response) {
    Debug.WriteLine("Response fail: " + response.getErrorMsg());
  }
  public void onSuccess(WLResponse response) {
    WLProcedureInvocationData invocationData = new
WLProcedureInvocationData("myAdapterName", "myProcedureName");
    invocationData.setParameters(new Object[]{"stringParam"});
    String myContextObject = new String("This is my context object");
    WLRequestOptions options = new WLRequestOptions();
    options.setInvocationContext(myContextObject);
    WLClient.getInstance().invokeProcedure(invocationData, new
MyInvokeListener(), options);
  }
}
```

#### Implementation of a Response Listener for Procedure Invocation

```
public class MyInvokeListener : WLResponseListener {
  public void onFailure(WLFailResponse response) {
    Debug.WriteLine("Response failed: " + response.getErrorMsg());
  }
  public void onSuccess(WLResponse response) {
    WLProcedureInvocationResult invocationResponse =
((WLProcedureInvocationResult) response);
    JObject items;
    try {
      items = invocationResponse.getResponseJSON();
      // do something with the items
    } catch (JSONException e) {
      }
  }
}
```

## 2.2 Class WLClient

This singleton class exposes methods that you use to communicate with the Worklight Server.

### 2.2.1 Method getInstance

#### Syntax

```
public static WLClient getInstance()
```

#### Description

This method gets the singleton instance of `WLClient`.

### 2.2.2 Method connect

#### Syntax

```
public void connect(WLResponseListener
responseListener)
```

#### Description

This method sends an initialization request to the Worklight Server, establishes a connection with the server, and validates the application version.

**Important:** You must call this method before any other `WLClient` methods that communicate with the Worklight Server.

#### Parameters

| Type | Name | Description |
|---|---|---|
| **WLResponseListener** | responseListener | When the server returns a successful response, the `WLResponseListener` `onSuccess` method is called. If an error occurs, the `onFailure` method is called |

*Table 2-1: Method connect parameters*

### 2.2.3 Method invokeProcedure

#### Syntax

```
public void invokeProcedure (
WLProcedureInvocationData invocationData,
WLResponseListener responseListener,
WLRequestOptions requestOptions)
```

```
public void invokeProcedure (
WLProcedureInvocationData invocationData,
WLResponseListener listener)
```

#### Description

This method sends an asynchronous call to an adapter procedure. The response is returned to the callback functions of the provided responseListener.

If the invocation succeeds, the `onSuccess` method is called. If the invocation fails, the `onFailure` method is called.

#### Parameters

| Type | Name | Description |
|---|---|---|
| **WLProcedure InvocationData** | invocationData | The invocation data for the procedure call. |
| **WLResponseListener** | responseListener | The listener object whose callback methods `onSuccess` and `onFailure` are called. |
| **WLRequestOptions** | requestOptions | Optional. Invocation options. |

*Table 2-2: Method invokeProcedure parameters*

### 2.2.4 Method logActivity

#### Syntax

```
public void logActivity (String activityType)
```

#### Description

This method reports a user activity for auditing or reporting purposes. The activity is stored in the raw table.

**Important:** Ensure that `reports.exportRawData` is set to **true** in the `worklight.properties` file, else the activity is not stored in the database. Also, ensure that the following properties are entered appropriately in the `worklight.properties` file:

```
wl.reports.db.type
wl.reports.db.url
wl.reports.db.username
wl.reports.db.password
```

#### Parameters

| Type | Name | Description |
|---|---|---|
| **String** | `activityType` | A string that identifies the activity. |

*Table 2-3: Method logActivity parameters*

### 2.2.5 Method setHeartBeatInterval

#### Syntax

```
public void setHeartBeatInterval (int value)
```

#### Description

This method sets the interval, in seconds, at which the Worklight Server sends the heartbeat signal. You use the heartbeat signal to ensure that the session with the server is kept alive when the app does not issue any call to the server, such as `invokeProcedure`. By default, the interval is set to 20 minutes.

#### Parameters

| Type | Name | Description |
|---|---|---|
| **int** | `value` | An interval value in seconds, at which the heartbeat signal is sent to Worklight Server. |

*Table 2-4: Method setHeartBeatInterval parameters*

### 2.2.6 Method getPush

#### Syntax

```
public WLPush getPush ()
```

#### Description

This method returns a WLPush object that the application can use to perform actions such as subscribing and unsubscribing to Push notifications.

## 2.3 Class WLProcedureInvocationData

This class contains all data necessary to call a procedure, including the following elements:

- The names of the adapter and procedure to call.
- The parameters that the procedure requires.

### 2.3.1 Method setParameters

#### Syntax

```
public void setParameters(Object [] parameters)
```

#### Description

This method sets the request parameters.

#### Parameters

| Type | Name | Description |
|---|---|---|
| `Object []` | `parameters` | An array of objects of primitive types (`String`, `Integer`, `Float`, `Boolean`, `Double`). The order of the objects in the array is the order in which they are sent to the adapter. |

*Table 2-5: Method setParameters parameters*

#### Example

```
invocationData.setParameters(new Object[]{"stringParam", true, 1.0,
1});
```

## 2.4 Class WLRequestOptions

This class contains the request parameters, headers, and invocation context.

### 2.4.1 Method addParameter

#### Syntax

```
public void addParameter(String name,String value)
```

#### Description

This method adds a request parameter with the given name and value.

#### Parameters

| Type | Name | Description |
|---|---|---|
| **`String`** | `name` | The name of the parameter. |
| **`String`** | `value` | The value of the parameter. |

*Table 2-6: Method addParameter parameters*

### 2.4.2 Method addParameters

#### Syntax

```
public void addParameters(Dictionary<String,
String> parameters)
```

#### Description

This method adds a table of request parameters.

#### Parameters

| Type | Name | Description |
|---|---|---|
| **`Dictionary <String,String>`** | `parameters` | Request parameters table |

*Table 2-7: Method addParameters parameters*

### 2.4.3 Method getParameter

#### Syntax

```
public String getParameter(String name)
```

#### Description

This method returns the value of the parameter that is set.

#### Parameters

| Type | Name | Description |
|---|---|---|
| **`String`** | `name` | The name of the parameter. |

*Table 2-8: Method getParameter parameters*

### 2.4.4 Method getParameters

#### Syntax

```
public Dictionary<String,String> getParameters()
```

#### Description

This method returns the parameters table.

### 2.4.5 Method getResponseListener

#### Syntax

```
public WLResponseListener getResponseListener()
```

#### Description

This method returns the response listener for this request.

### 2.4.6 Method addHeader

#### Syntax

```
public void addHeader(String name,String value)
```

#### Description

You can use this method to add a header with the given name and value.

#### Parameters

| Type | Name | Description |
|---|---|---|
| **String** | Name | The name of the header. |
| **String** | Value | The value of the header. |

*Table 2-9: Method addHeader parameters*

### 2.4.7 Method setHeaders

#### Syntax

```
public void setHeaders(WebHeaderCollection
extraHeaders)
```

#### Description

This method sets the request with the given headers.

#### Parameters

| Type | Name | Description |
|---|---|---|
| **WebHeaderCollection** | extraHeaders | The headers to be set. |

*Table 2-10: Method setHeaders parameters*

### 2.4.8 Method getHeaders

#### Syntax

```
public WebHeaderCollection getHeaders()
```

#### Description

This method returns the headers that are set for this request.

### 2.4.9 Methods getInvocationContext, setInvocationContext

#### Syntax

```
public Object getInvocationContext()


public void setInvocationContext(Object
invocationContext)
```

#### Parameters

| Type | Name | Description |
|---|---|---|
| **`Object`** | `invocationContext` | An object that is returned with `WLResponse` to the listener methods `onSuccess` and `onFailure`. You can use this object to identify and distinguish different `invokeProcedure` calls. This object is returned as is to the listener methods. |

*Table 2-11: Methods getInvocationContext, setInvocationContext parameters*

## 2.5 Interface WLResponseListener

This interface defines methods that the listener for the `WLClient.invokeProcedure` method implements to receive notifications about the success or failure of the method call.

### 2.5.1 Method onSuccess

#### Syntax

```
public void onSuccess (WLResponse response)
```

#### Description

This method is called after successful calls to the `WLCLient connect` or `invokeProcedure` methods.

#### Parameters

| Type | Name | Description |
|---|---|---|
| **`WLResponse`** | `response` | The response that the server returns, along with any invocation context object and status. |

*Table 2-12: Method onSuccess parameters*

### 2.5.2 Method onFailure

#### Syntax

```
public void onFailure (WLFailResponse response)
```

**Description**

This method is called if any failure occurred during the execution of the `WLCLient connect` or `invokeProcedure` methods.

**Parameters**

| Type | Name | Description |
|---|---|---|
| **`WLFailResponse`** | `response` | A response that contains the error code and error message. Optionally, this response contains the results from the server, and any invocation context object and status. |

*Table 2-13: Method onFailure parameters*

## 2.6 Class WLResponse

This class contains the result of a procedure invocation. Worklight passes this class as an argument to the listener methods of the `WLClient invokeProcedure` method.

### 2.6.1 Method getStatus

**Syntax**

```
public HttpStatusCode getStatus()
```

**Description**

This method retrieves the `HTTP` status from the response.

### 2.6.2 Method getInvocationContext

**Syntax**

```
public Object getInvocationContext()
```

**Description**

This method retrieves the invocation context object that is passed when the `invokeProcedure` method is called.

### 2.6.3 Method getResponseText

**Syntax**

```
public String getResponseText()
```

**Description**

This method retrieves the original response text from the server.

### 2.6.4 Method getResponseJSON

**Syntax**

```
public JObject getResponseJSON()
```

#### Description

This method retrieves the response text from the server in JSON format.

## 2.7 Class WLFailResponse

This class extends `WLResponse`. This class contains error codes, messages, the status in `WLResponse`, and the original response DataObject from the server.

### 2.7.1 Method getErrorCode

#### Syntax

```
public WLErrorCode getErrorCode ()
```

#### Description

The `WLErrorCode` section describes the possible errors.

### 2.7.2 Method getErrorMsg

#### Syntax

```
public String getErrorMsg()
```

#### Description

This method returns an error message that is for the developer and not necessarily suitable for the user.

## 2.8 Class WLProcedureInvocationResult

This class extends `WLResponse`. This class contains statuses and data that an adapter procedure retrieves.

### 2.8.1 Method getResult

#### Syntax

```
public JObject getResult()
```

#### Description

This method returns a `JObject` that represents the JSON response from the server.

### 2.8.2 Method isSuccessful

#### Syntax

```
public boolean isSuccessful()
```

#### Description

This method returns **true** if the procedure invocation was technically successful. Application errors are returned as part of the retrieved data, and not in this flag.

## 2.9 Class WLProcedureInvocationFailResponse

This class extends `WLFailResponse`. This class contains statuses and data that an adapter procedure retrieves.

### 2.9.1 Method getProcedureInvocationErrors

#### Syntax

```
public List<String> getProcedureInvocationErrors()
```

#### Description

This method returns a list of applicative error messages that are collected while the procedure is called.

### 2.9.2 Method getResult

#### Syntax

```
public JObject getResult()
```

#### Description

This method returns a `JObject` that represents the JSON response from the server.

## 2.10 Class WLErrorCode

This class contains the error code and its description that the server returns.

### 2.10.1 Method getDescription

#### Syntax

```
public String getDescription()
```

#### Description

This method returns the description of this error code instance.

### 2.10.2 Method valueOf

#### Syntax

```
public static WLErrorCode valueOf(String errorCode)
```

#### Description

This method returns the error code instance of the `errorCode` that is given.

**Error Codes**

`UNEXPECTED_ERROR` – Unexpected `errorCode` occurred. Please try again.

`REQUEST_TIMEOUT` – Request timed out.

`UNRESPONSIVE_HOST` – The service is currently unavailable.

`PROCEDURE_ERROR` – Procedure invocation `errorCode`.

`PROCEDURE_PROTECTED_ERROR` – Procedure is protected.

`APP_VERSION_ACCESS_DENIAL` – Application version denied.

`APP_VERSION_ACCESS_NOTIFY` – Notify application version changed.

## 2.11 Class BaseChallengeHandler

This class is an abstract base class for all Challenge Handlers.

### 2.11.1 Constructor

#### Syntax

```
public BaseChallengeHandler(String realm)
```

#### Description

This method creates a `BaseChallengeHandler` object for a particular `realm`.

### 2.11.2 Method handleChallenge(T challenge)

#### Syntax

```
public abstract void handleChallenge(T challenge)
```

#### Description

This method must be implemented by the subclass to handle the challenge logic. For example, show a login form in a challenge from a `FormBasedAuthenticator`.

## 2.12 Class ChallengeHandler

This class is an abstract class that you must extend to create custom challenge handlers.

### 2.12.1 Constructor

#### Syntax

```
public ChallengeHandler(String realmName)
```

#### Description

This method creates a `ChallengeHandler` object for a particular `realm`.

### 2.12.2 Method isCustomResponse

#### Syntax

```
public abstract bool isCustomResponse(WLResponse
response)
```

#### Description

You must implement this method and parse the response to determine whether the response from the server is a challenge for this `ChallengeHandler`. For example, a `ChallengeHandler` for a realm with a form-based authenticator must parse the response to search for the `j_security_test` parameter and return `true` if found.

### 2.12.3 Method submitSuccess

#### Syntax

```
protected void submitSuccess(WLResponse response)
```

#### Description

You must call this method from the subclass within the `onSuccess` of your `ChallengeHandler`.

### 2.12.4 Method submitLoginForm

#### Syntax

```
protected void submitLoginForm(String requestURL,
Dictionary<String, String> requestParameters,
Dictionary<String, String> requestHeaders, int
requestTimeoutInMs, String requestMethod )
```

#### Description

This helper method submits a login form by making an HTTP request to the specified `requestURL`.

#### Parameters

| Type | Name | Description |
|---|---|---|
| **String** | `requestURL` | The full or relative URL to which the request must be made. |
| **Dictionary <String, String>** | `requestParameters` | A Dictionary object with name-value pairs of request parameters. |
| **Dictionary <String, String>** | `requestHeaders` | A Dictionary object consisting of the additional headers that must be sent along with the `HTTP` request. |

| Type | Name | Description |
|---|---|---|
| **`int`** | `requestTimeoutInMs` | The time in milliseconds the request must wait before timing out. |
| **`String`** | `requestMethod` | The method to use. Specify `GET` or `POST`. |

*Table 2-14: Method submitLoginForm parameters*

### 2.12.5 Method submitAdapterAuthentication

#### Syntax

```
protected void submitAdapterAuthentication(String
WLProcedureInvocationData invocationData,
WLRequestOptions requestOptions )
```

#### Description

This helper method submits a response to a challenge made by an `AdapterAuthenticator` by using an `invokeProcedure` call to the adapter procedure.

#### Parameters

| Type | Name | Description |
|---|---|---|
| **`WLProcedureInvocationData`** | `invocationData` | The `WLProcedureInvocationData` object that contains the name of the adapter and the procedure. |
| **`WLRequestOptions`** | `requestOptions` | A `WLRequestOptions` object with request options. |

*Table 2-15: Method submitAdapterAuthentication parameters*

## 2.13 Class WLChallengeHandler

This class is an abstract base class for Worklight Challenge Handlers. You must extend it to implement your own version of a Worklight Challenge Handler, for example, the `RemoteDisableChallengeHandler`.

### 2.13.1 Constructor

#### Syntax

```
public WLChallengeHandler(String realm)
```

#### Description

This method creates a `WLChallengeHandler` object for a particular `realm`.

### 2.13.2 Method submitChallengeAnswer

#### Syntax

```
public void submitChallengeAnswer(Object answer)
```

#### Description

This method sends the answer back to the server.

## 2.14 Class WLPush

This class contains all the methods required to work with Push notifications. You cannot instantiate this class directly. To get a reference to this class, use the getPush() method of WLClient.

To enable Push notifications, add the pushSender element to the application descriptor of your Native API application.

```
<nativeWindowsPhone8App>
...
      <pushSender>
            <authenticatedPush
serviceName="service_name" keyAlias="key_alias"
keyAliasPassword="key_password"/>
      </pushSender>

…
</nativeWindowsPhone8App>
```

MPNS provides two ways to send push notifications to devices. One, is a non-authenticated mode where the Push requests are throttled. The second is an authenticated mode where push requests are not throttled. Sending authenticated push notification requires authenticating Worklight with MPNS by using a SSL certificate. For more information, go to http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff941099(v=vs.105).aspx

If you are not using authenticated push notifications, you can leave the pushSender tag empty.

Open the WMAppManifest.xml file of your application and under the capabilities section, select ID_CAP_PUSH_NOTIFICATION.

### 2.14.1 Method registerEventSourceCallback

#### Syntax

```
public void registerEventSourceCallback(String
alias, String adapter, String eventSource,
WLEventSourceListener eventSourceListener)
```

#### Description

This method registers a `WLEventSourceListener` that is called whenever a notification arrives from the specified event source.

**Parameters**

| Type | Name | Description |
|---|---|---|
| `String` | `alias` | Mandatory string. A short ID that you use to identify the event source when the push notification arrives. You can provide a short alias, rather than the full names of the adapter and event source. This action frees space in the notification text, which is limited in length. |
| `String` | `adapter` | Mandatory string. The name of the adapter that contains the event source. |
| `String` | `eventSource` | Mandatory string. The name of the event source. |
| `WLEventSourceListener` | `eventSourceListener` | Mandatory listener class. When a notification arrives, the `WLEventSourceListener`.onReceive method is called. |

*Table 2-16: Method registerEventSourceCallback parameters*

### 2.14.2 Method subscribe

**Syntax**

```
public void subscribe(String alias, WLPushOptions
pushOptions, WLResponseListener respListener)
```

**Description**

This method subscribes the user to the event source with the specified alias.

**Parameters**

| Type | Name | Description |
|---|---|---|
| `String` | `alias` | Mandatory string. The event source alias, as defined in registerEventSourceCallback. |
| `WLPushOptions` | `pushOptions` | This instance contains the custom subscription parameters that the event source in the adapter supports. |
| `WLResponseListener` | `respListener` | The listener object whose callback methods are called by the Worklight runtime when a subscribe call succeeds or fails. |

*Table 2-17: Method subscribe parameters*

### 2.14.3 Method unsubscribe

**Syntax**

```
public void unsubscribe(String alias,
WLResponseListener respListener)
```

**Description**

This method unsubscribes the user from the event source with the specified alias.

**Parameters**

| Type | Name | Description |
|---|---|---|
| **String** | alias | Mandatory string. The event source alias, as defined in registerEventSourceCallback. |
| **WLResponse Listener** | respListener | The listener object whose callback methods are called by the Worklight runtime when a subscribe call succeeds or fails. |

*Table 2-18: Method unsubscribe parameters*

### 2.14.4 Method isSubscribed

**Syntax**

```
public void isSubscribed(String alias)
```

**Description**

This method returns whether the currently logged-in user is subscribed to the specified event source alias.

**Parameters**

| Type | Name | Description |
|---|---|---|
| **String** | alias | Mandatory string. The event source alias. |

*Table 2-19: Method isSubscribed  parameters*

### 2.14.5 Method subscribeTag

**Syntax**

```
public void subscribeTag(String tagName,
WLPushOptions pushOptions, WLResponseListener
respListener)
```

**Description**

This method subscribes the device to the tag.

**Parameters**

| Type | Name | Description |
|---|---|---|
| **`String`** | `tagName` | Mandatory string. The name of the tag. |
| **`WLPushOptions`** | `pushOptions` | This instance contains the custom subscription parameters that the event source in the adapter supports. |
| **`WLResponseListener`** | `respListener` | The listener object whose callback methods are called by the Worklight runtime when a subscribe call succeeds or fails. |

*Table 2-20: Method `subscribeTag` parameters*

### 2.14.6 Method unsubscribeTag

**Syntax**

```
public void unsubscribeTag(String tagName,
WLResponseListener respListener)
```

**Description**

This method unsubscribes the device from the tag.

**Parameters**

| Type | Name | Description |
|---|---|---|
| **`String`** | `tagName` | Mandatory string. The name of the tag. |
| **`WLResponseListener`** | `respListener` | The listener object whose callback methods are called by the Worklight runtime when a subscribe call succeeds or fails. |

*Table 2-21: Method un`subscribeTag` parameters*

### 2.14.7 Method isTagSubscribed

**Syntax**

```
public void isTagSubscribed(String tagName)
```

**Description**

This method returns whether the device is subscribed to the specified tag.

**Parameters**

| Type | Name | Description |
|---|---|---|
| `String` | `tagName` | Mandatory string. The name of the tag. |

*Table 2-22: Method isTagSubscribed  parameters*

### 2.14.8 Property onReadyToSubscribeListener

#### Type

WLOnReadyToSubscribeLister

#### Access

`Read/Write`

#### Description

This property sets the `WLOnReadyToSubscribeListener` callback to be notified when the device is ready to subscribe to push notifications.

### 2.14.9 Property notificationListener

#### Type

WLNotificationListener

#### Access

`Read/Write`

#### Description

This property sets the `WLNotificationListener` callback to be notified when the push notification arrives.

## 2.15 Interface WLOnReadyToSubscribeListener

This interface defines the method that is notified when a device is ready to subscribe.

### 2.15.1 Method onReadyToSubscribe

#### Syntax

```
void onReadyToSubscribe()
```

#### Description

This method is called when the device is ready to subscribe to push notifications.

## 2.16 Interface WLEventSourceListener

This interface defines the method that receives the notification message.

### 2.16.1 Method onReceive

#### Syntax

```
void onReceive(String properties, String payload)
```

#### Description

This method is called when the notification arrives from the subscribed event source.

#### Parameters

| Type | Name | Description |
|---|---|---|
| **`String`** | `properties` | A JSON block that contains the notifications properties of the platform. |
| **`String`** | `payload` | A JSON block that contains other data that is sent from the Worklight Server. |

*Table 2-23: Method onReceive parameters*

## 2.17 Interface WLNotificationListener

This interface defines the method that receives the notification message.

### 2.17.1 Method onMessage

#### Syntax

```
void onMessage(String properties, String payload)
```

#### Description

This method is called when a push notification arrives.

#### Parameters

| Type | Name | Description |
|---|---|---|
| **String** | properties | A JSON block that contains the notifications properties of the platform. |
| **String** | payload | A JSON block that contains 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. The default tag name for broadcast notification is Push.ALL. |

*Table 2-24: Method* `onMessage` *parameters*

## 2.18 Class WLPushOptions

This class contains the subscription parameters that can be specified while subscribing to push notifications.

### 2.18.1 Constructor

#### Syntax

```
public WLPushOptions()
```

#### Description

This method creates a WLPushOptions object.

### 2.18.2 Method AddSubscriptionParameter

#### Syntax

```
public void AddSubscriptionParameter(String name,
String value)
```

#### Description

Use this method to add a subscription parameter.

#### Parameters

| Type | Name | Description |
|---|---|---|
| **String** | name | Mandatory. The name of the subscription parameter. |
| **String** | value | Mandatory. The value of the subscription parameter. |

*Table 2-25: Method AddSubscriptionParameter parameters*

## 2.18.3 Method GetSubscriptionParameter

#### Syntax

```
public void GetSubscriptionParameter(String name)
```

#### Description

This method returns the map that contains the subscription parameters.

#### Parameters

| Type | Name | Description |
|---|---|---|
| `String` | `name` | Mandatory. The name of the subscription parameter. |

*Table 2-26: Method AddSubscriptionParameter parameters*

## 2.18.4 Property subscriptionParameters

#### Type

Dictionary <String, String>

#### Access

`Read/Write`

#### Description

This property gets/sets the subscription parameters.

# Appendix A - Notices

**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 section 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.

# Appendix B - Support and comments

For the entire 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