IBM Worklight

IBM Worklight V5.0.6

Java client-side API for native Android apps

17 January 2014

Copyright Notice

Trademarks

IBM, the IBM logo, ibm.com are trademarks or registered trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Worklight is a trademark or registered trademark of Worklight, an IBM Company. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Other company products or service names may be trademarks or service marks of others.

About IBM®

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

Contents

Tables

About this document

This document is intended for Android developers who want to access IBM® Worklight® services from Android applications written in Java™ and from hybrid Android applications. The document guides you through the available classes and methods.

1 API overview

The IBM Worklight Java client-side API for native Android apps exposes five main capabilities:

- Calling back-end services to retrieve data and perform back-end transactions.
- Writing custom log lines for reporting and auditing purposes.
- Authenticating users before they access sensitive data or perform privileged actions.
- Implementing custom Challenge Handlers that you use to create a custom authentication process.
- Subscribing and unsubscribing to push notifications.

The IBM Worklight Java client-side API for native Android apps is available as part of the Worklight Studio.

Type	Name	Description	Implemented by
Properties file	`wlclient.properties`	Properties file that contains the necessary data to use the IBM Worklight API.	IBM
Package	`com.worklight.wlclient.api`	All API classes are defined in this package. You must import this package in the Android code to leverage the capabilities of IBM Worklight.	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	`ChallengeHandler`	Abstract base class for the custom Challenge Handlers. You must extend it to implement custom authentication.	IBM
Class	`WLProcedureInvocationData`	Class that contains all data necessary to call a procedure.	IBM
Class	`WLRequestOptions`	Class that you can use to change the request timeout 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

Type	Name	Description	Implemented by
Class	`WLResponse`	Class that contains the result of a procedure invocation.	IBM
Class	`WLFailResponse`	Class that extends `WLResponse` and that contains error codes and messages in addition to the status in `WLResponse`. This class contains the original response DataObject from the server as well.	IBM
Class	`WLProcedureInvocationResult`	Class that extends `WLResponse` and that contains the result of calling a back-end service, including 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
Enum	`WLErrorCode`	An enumeration of error messages that arrive from the Worklight Server.	IBM
Class	`WLCookieExtractor`	Class that provides access to external cookies that `WLClient` can use when it issues requests to the Worklight Server. You use this class to share session cookies between a web view and a natively implemented page.	IBM
Class	`WLPreference`	Class that implements a preferences activity to view and modify connectivity properties to the Worklight Server.	IBM
Class	`WLDeviceAuthManager`	Class that provides utility functions that help in the implementation of custom provisioning process of a secure device ID.	IBM
Class	`WLPush`	Class that exposes methods to subscribe and unsubscribe to push notifications.	IBM
Class	`WLPushOptions`	Class that contains the subscription parameters.	IBM

Type	Name	Description	Implemented by
Interface	`WLOnReadyToSubscribeListener`	Interface that defines the method that is notified when a device is ready to subscribe.	Application developer
Interface	`WLEventSourceListener`	Interface that defines the method that receives the notification message.	Application developer
Package	`com.worklight.wlclient.ui`	Package that contains an activity that the platform uses to display UI.	IBM
Class	`UIActivity`	Android Activity class that the IBM Worklight platform uses to display UI (dialogs and such) in an Android environment. This class is not exposed to developers, but they must add it to their `AndroidManifest.xml` file.	IBM
Package	`com.worklight.wlclient.api.challengehandler`	Package that defines Challenge Handler classes that must be used in the authentication process.	IBM
Class	`BaseChallengeHandler`	Abstract base class for all the Challenge Handlers.	IBM
Class	`WLChallengeHandler`	Abstract base class for the IBM Worklight Challenge Handlers. You must extend it to implement your own version of an IBM Worklight Challenge Handler, for example `RemoteDisableChallengeHandler.`	IBM

Table 1-1: IBM Worklight Java API for Android packages, classes, interfaces, and files

2 API reference

2.1 Example Code

The following code samples show how to use the IBM Worklight Java client-side API for native Android apps.

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

Initializing the IBM Worklight Client

```
// run this code in your Android activity
WLClient client = WLClient.createInstance(this);
client.connect(new MyConnectResponseListener ());
```

Implementation of a Response Listener for connect

```
public class MyConnectResponseListener implements
   WLResponseListener{

   @Override
   public void onSuccess(WLResponse response) {
       WLProcedureInvocationData invocationData = new
       WLProcedureInvocationData("myAdapterName",
   "myProcedureName");
       invocationData.setParameters(new Object[]{"stringParam",
   true, 1.0, 1});
       String myContextObject = new String("This is my context
   object");
       WLRequestOptions options = new WLRequestOptions();
       options.setTimeout(10000);
       options.setInvocationContext(myContextObject);
       WLClient.getInstance().invokeProcedure(invocationData, new
   MyInvokeListener (), options);
   }

   @Override
   public void onFailure(WLFailResponse response) {
     WLUtils.error("Connection failed:" + response.getErrorMsg()
```



```
    }
}
```

Implementation of a Response Listener for Procedure Invocation

```
public class MyInvokeListener implements WLResponseListener {

   @Override
   public void onSuccess(WLResponse response) {
     WLUtils.debug("Response successful!");
     WLProcedureInvocationResult invocationResponse =
   ((WLProcedureInvocationResult) response);
     JSONArray items;
     try {
           items = (JSONArray)
   invocationResponse.getResult().get("items");
           // do something with the items
           for (int i = 0; i < items.length(); i++) {
                 JSONObject jsonObject = items.getJSONObject(i);
                  (…)
           }
     } catch (JSONException e) {
     }
   }

   @Override
   public void onFailure(WLFailResponse response) {
     WLUtils.error("Response failed: " + response.getErrorMsg());
   }
}
```

2.2 Class WLClient

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

2.2.1 Method createInstance

Syntax

```
public static WLClient createInstance(Context
context)
```

Description

This method creates the singleton instance of `WLClient`.

Note: This method is the first `WLClient` method that you use. It must be called before subsequent calls to `getInstance`. You must invoke this method at the beginning of the main activity of the application.

Parameters

Type	Name	Description
`Context`	`context`	This parameter is the Android context, for example the Android Activity that creates the `WLClient`.

Table 2-1: WLClient instantiation

2.2.2 Method getInstance

Syntax

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

Description

This method gets the singleton instance of `WLClient`.

2.2.3 Deprecated method init

This method is deprecated. Use the `connect` method instead.

2.2.4 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, for example `InvokeProcedure`.

Parameters

Type	Name	Description
WLResponseListener	`responseListener`	When a successful response is returned from the server, the `WLResponseListener onSuccess` method is called. If an error occurs, the `onFailure` method is called.

Table 2-2: Method connect parameters

2.2.5 Method invokeProcedure

Syntax

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

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, `onSuccess` is called. If it fails, `onFailure` is called.

Parameters

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

Table 2-3: Method invokeProcedure parameters

2.2.6 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 application statistics tables (the `GADGET_STAT_N` tables).

Parameters

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

Table 2-4: Method logActivity parameters

2.2.7 Method checkForNotifications

Syntax

```
public void checkForNotifications()
```

Description

You use this method to check for notifications on the server, such as new block/notify rules, or notifications. When you call this method from the `onResume Android Activity` lifecycle event and you bring the activity to the foreground, the application checks for new notifications.

2.2.8 Method registerChallengeHandler

Syntax

```
public void
registerChallengeHandler(BaseChallengeHandler
challengeHandler)
```

Description

You can use this method to register a Challenge Handler in the client. You must use this method when you implement custom challenge handlers, or when you customize the Remote Disable / Notify Challenge Handler.

Important: you must call this method at the beginning of your application after you initialize `WLClient`.

Example 1: registering a customized Remote Disable / Notify Challenge Handler

To customize the Remote Disable / Notify Challenge Handler, you must register an instance of type `WLChallengeHandler` in the client. When you create the Challenge Handler, you must give it the specific realm name `wl_remoteDisableRealm`.

```
// define class
public class MyRemoteDisableCH extends WLChallengeHandler {
      .
      .
      .
}
// create new CH with appropriate realm
MyRemoteDisableCH ch = new
   MyRemoteDisableCH("wl_remoteDisableRealm");

// register CH
WLClient.getInstance().registerChallengeHandler(ch);
```


Example 2: customizing the Remote Disable / Notify Challenge Handler

To customize the Remote Disable / Notify Challenge Handler, you must extend `WLChallengeHandler` and implement the following methods.

```
public void handleSuccess(JSONObject success)
public void handleFailure(JSONObject error)
public void handleChallenge(JSONObject challenge)
```

```
public class MyRemoteDisableCH extends WLChallengeHandler {

   public MyRemoteDisableCH(String realm) {
     super(realm);
   }

   @Override
   /**
    * this method is called after the challenge is answered
   successfully
    */
   public void handleSuccess(JSONObject success) {

   }

   @Override
   /**
    * this method is used to disable the application
    */
   public void handleFailure(JSONObject error) {
     try {

           // get error message
           String message = error.getString("message");

           // get download link
           String downloadLink = error.getString("downloadLink");

           // create and show the disable dialog

     } catch (JSONException e) {
```



```
            // handle exception
      }
    }

    @Override
    /**
     * this method is used to notify the application
     */
    public void handleChallenge(JSONObject challenge) {

      try {

            // get message data from challenge
            String message = challenge.getString("message");
            String messageId = challenge.getString("messageId");

            // do something with the message

            // answer the challenge
            submitChallengeAnswer(messageId);

      } catch (JSONException e) {

            // handle exception
      }

    }

}
```

Note: When the application is disabled, the behavior by default is to open a dialog that displays the appropriate message. You must implement this behavior by default in the method `handleFailure` of `RemoteDisableChallengeHandler`. The dialog can also display a link to download the new version of the application. After the user closes the dialog, the application continues to work offline. You must implement a similar behavior in the `handleFailure` code of the custom Remote Disable Challenge Handler.

2.2.9 Method addGlobalHeader

Syntax


```
public void addGlobalHeader(String headerName,
String value)
```

Description

You use this method to add a global header, which is sent on each request.

Parameters

Type	Name	Description
String	headerName	The name of the header.
String	value	The value of the header.

Table 2-5: Method addGlobalHeader parameters

2.2.10 Method removeGlobalHeader

Syntax

```
public void removeGlobalHeader(String headerName)
```

Description

You use this method to remove a global header. Then, the header is no longer sent on each request.

Parameters

Type	Name	Description
String	headerName	The name of the header.

Table 2-6: Method removeGlobalHeader parameters

2.3 Class ChallengeHandler

You use this abstract base class to create custom Challenge Handlers. You must extend this class to implement your own Challenge Handler logics. You use this class to create custom user authentication.

2.3.1 Method isCustomResponse

Syntax

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

Description

This method must be overridden by extending the `ChallengeHandler` class. In most cases, you call this method to test whether there is a custom challenge to be handled in the response. If the method returns **true**, the IBM Worklight framework calls the `handleChallenge` method.

Parameters

Type	Name	Description
WLResponse	`response`	The response to be tested.

Table 2-7: Method isCustomResponse parameters

2.3.2 Method handleChallenge

Syntax

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

Description

You must implement this method to handle the challenge logic, for example to display the login screen. The IBM Worklight framework calls the method `handleChallenge` whenever the method `isCustomResponse` returns **true**.

Parameters

Type	Name	Description
WLResponse	`challenge`	The response to be handled.

Table 2-8: Method handleChallenge parameters

2.3.3 Method submitFailure

Syntax

```
protected void submitFailure(WLResponse wlResponse)
```

Description

You must call this method when the challenge is answered with an error. The method is inherited from `BaseChallengeHandler`.

Parameters

Type	Name	Description
WLResponse	`wlResponse`	The received `WLResponse`.

Table 2-9: Method submitFailure parameters

2.3.4 Method submitSuccess

Syntax

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

Description

You must call this method when the challenge is answered successfully, for example after the user successfully submits the login form. Then, this method sends the original request.

Parameters

Type	Name	Description
WLResponse	`response`	The received `WLResponse`.

Table 2-10: Method submitSuccess parameters

2.3.5 Method submitLoginForm

Syntax

```
protected void submitLoginForm(String requestURL,
Map<String, String> requestParameters, Map<String,
String> requestHeaders,int
requestTimeoutInMilliseconds, String requestMethod)
```

Description

You use this method to send collected credentials to a specific URL. You can also specify request parameters, headers, and timeout.

The success/failure delegate for this method is the instance itself (the instance of `ChallengeHandler`), so you must override the `onSuccess` / `onFailure` methods.

Parameters

Type	Name	Description
String	`requestURL`	Absolute URL if the user sends an absolute URL that starts with `http://` or `https://` Otherwise, URL relative to the Worklight Server.
Map<String, String>	`requestParameters`	The request parameters.
Map<String, String>	`requestHeaders`	The request headers.
int	`requestTimeoutInMilliseconds`	To supply custom timeout, use a number over 0. If the number is under 0, the IBM Worklight framework uses the default timeout, which is 10,000 milliseconds.
String	`requestMethod`	The `HTTP` method that you must use. Acceptable values are `GET`, `POST`. The default value is `POST`.

Table 2-11: Method submitLoginForm parameters

2.3.6 Method submitAdapterAuthentication

Syntax

```
public void
submitAdapterAuthentication(WLProcedureInvocationDa
ta invocationData, WLRequestOptions requestOptions)
```

Description

You use this method to invoke a procedure from the Challenge Handler.

Parameters

Type	Name	Description
WLProcedureInvocationData	`invocationData`	The invocation data, for example the name of the procedure or the name of the method.
WLRequestOptions	`requestOptions`	It contains the following options. `timeout – int`: The time, in milliseconds, for this `invokeProcedure` to wait before it fails with `WLErrorCodeRequestTimeout`. The default timeout is 10,000 milliseconds. To disable the timeout, set this parameter to 0. `invocationContext – Object`: An object that is returned with `WLResponse` to the delegate methods. You can use this object to distinguish different `invokeProcedure` calls.

Table 2-12: Method submitAdapterAuthentication parameters

2.3.7 Method onSuccess

Syntax

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

Description

This method is the success handler for `submitLoginForm` or `submitAdapterAuthentication`.

Parameters

Type	Name	Description
WLResponse	`response`	The received response.

Table 2-13: Method onSuccess parameters

2.3.8 Method onFailure

Syntax

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

Description

This method is the failure handler for `submitLoginForm` or `submitAdapterAuthentication`.

Parameters

Type	Name	Description
`WLFailResponse`	`response`	The received response.

Table 2-14: Method onFailure parameters

Example: implementing a form-based Challenge Handler

```
/*
*  Register the custom handler in the Main Activity
*/
public class FormBasedAuthentication extends Activity {
   @Override
   public void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     WLClient client = WLClient.createInstance(this);
     client.registerChallengeHandler (new
   SampleAppRealmChallengeHandler ("SampleAppRealm"));
   }
};

/*
* Implementation of Custom Challenge Handler
*/
class SampleAppRealmChallengeHandler extends ChallengeHandler {
   public SampleAppRealmChallengeHandler(String realm) {
     super(realm);
   }
/*
*  Called when the framework needs to identify custom response.
*  In this example is identified by “j_security_check” string
   located in response text.
*/

@Override
   public boolean isCustomResponse(WLResponse response) {
     if (response == null || response.getResponseText() == null ||
         response.getResponseText().indexOf("j_security_check") == -
   1) {
           return false;
     }
     return true;
   }

/*
```



```
* Called to handle custom challenge
/*

   @Override
   public void handleChallenge(WLResponse response) {

// … //
//   Show login form and ask for user name and password
//   When the user name and password are provided by user, pass them
   back to the server using
//   submitLoginForm API.
// … //
     Map<String, String> params = new HashMap<String, String>();
     params.put("j_username", "test");
     params.put("j_password", "pwd");
     super.submitLoginForm("j_security_check", params, null, 0,
   "post");
   }
/*
* onSuccess is always called when the server returns a response. A
   developer is responsible to parse the response
* and display a login form (handle challenge) or submit success
   answer.
*/
   @Override
   public void onSuccess(WLResponse response) {
     if (isCustomResponse(response)) {
           handleChallenge(response);
     } else {
           submitSuccess(response);
     }
   }

/*
* onFailure is called in case of socket/timeout exceptions
   WLErrorCode is set to
* REQUEST_TIMEOUT/UNRESPONSIVE_HOTS codes. In case of general
   exception error code is
* UNEXPECTED_ERROR.
```



```
*/
   @Override
   public void onFailure(WLFailResponse response) {
     submitFailure(response);
   }
}
```

2.4 Class WLProcedureInvocationData

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

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

2.4.1 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 2-15: Method setParameters parameters

Example

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

2.5 Class WLRequestOptions

This class changes the timeout and invocation context.

2.5.1 Methods getTimeout, setTimeout

Syntax

```
public int getTimeout()
```

```
public void setTimeout(int timeout)
```

Description

`getTimeout`: this method gets the request timeout that you currently use. The default request timeout is 10 seconds).

`setTimeout`: this method sets a new timeout.

Parameters

Type	Name	Description
int	`timeout`	The timeout, in milliseconds, to wait for the procedure invocation. If the timeout expires, the `WLResponseListener onFailure` method is called. The value 0 indicates that there is no timeout.

Table 2-16: Methods getTimeout, setTimeout parameters

2.5.2 Methods getInvocationContext, setInvocationContext

Syntax

```
public Object getInvocationContext()
```

```
public void setInvocationContext(Object
invocationContext)
```

Parameters

Type	Name	Description

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 2-17: Methods getInvocationContext, setInvocationContext parameters

2.6 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.

2.6.1 Method onSuccess

Syntax

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

Description

This method is called following 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 2-18: Method onSuccess parameters

2.6.2 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

Type	Name	Description
WLFailResponse	response	A response that contains the error code and error message. Optionally, it can also contain the results from the server and any invocation context object and status.

Table 2-19: Method onFailure parameters

2.7 Class WLResponse

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

2.7.1 Method getStatus

Syntax

```
public int getStatus()
```

Description

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

2.7.2 Method getInvocationContext

Syntax

```
public Object getInvocationContext()
```

Description

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

2.7.3 Method getResponseText

Syntax

```
public Object getResponseText()
```

Description

This method retrieves the original response text from the server.

2.8 Class WLFailResponse

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

2.8.1 Method getErrorCode

Syntax

```
public WLErrorCode getErrorCode ()
```

Description

The `WLErrorCode` section contains a description of the possible errors.

2.8.2 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.

2.9 Class WLProcedureInvocationResult

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

2.9.1 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.

2.10 Class WLProcedureInvocationFailResponse

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

2.10.1 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.

2.10.2 Method getResult

Syntax

```
public JSONObject getResult() throws JSONException
```

Description

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

2.11 Enum WLErrorCode

Description

The Worklight Server can return the following error messages:

```
UNEXPECTED_ERROR
REQUEST_TIMEOUT
REQUEST_SERVICE_NOT_FOUND
UNRESPONSIVE_HOST
PROCEDURE_ERROR
APP_VERSION_ACCESS_DENIAL
APP_VERSION_ACCESS_NOTIFY
```


2.12 Class WLCookieExtractor

This class provides access to external cookies that `WLClient` can use when it issues requests to the Worklight Server. You use this class to share session cookies between a web view and a natively implemented page.

2.12.1 Static member cookies

Syntax

```
public static String cookies
```

Description

The static member cookies are the cookies that the `WLCookieExtractor` share. They can be accessed statically.

2.13 Class Logger

The Logger class is an abstraction of, and pass-through to, `android.util.Log`. The Logger class provides some enhanced capability such as capturing log calls, log level control at both the global and individual package scope, and package filtering. The Logger class also provides a method call to send captured logs to the Worklight Server.

When the capture flag of this `Logger` class is turned on with the `setCapture(true)` method call, all messages passed through the log methods of this class are persisted to file in the following JSON object format:

```
{
   "timestamp" : "17-02-2013 13:54:27:123",  // "dd-MM-yyyy hh:mm:ss:S"
   "level"     : "ERROR",                    // ERROR || WARN || INFO || LOG ||
   DEBUG
   "package"   : "your_tag",                 // typically a class name, app name,
   or JavaScript object name
   "msg"       : "the message",              // a helpful log message
   "metadata"  : {"hi": "world"},            // (optional) additional JSON
   metadata, appended via doLog API call
   "threadid"  : long                        // (optional) id of the current thread
 }
```

Log data is accumulated persistently to a log file until the file size is greater `than FILE_SIZE_LOG_THRESHOLD`. At this point, the log file is rolled over. When both files are full, the oldest log data is pushed out to make room for new log data.

Log file data is sent to the URL, which is by default an IBM Worklight servlet that the user must enable, when the `send()` method of this class is called and the file size is greater than zero. When the server replies with the response code `200 OK`, the file is deleted.

All of the method calls of this class, such as `info(String)`, are pass-throughs to the equivalent method call in `android.util.Logger`, when the `LEVEL` log function that is called is at the `LEVEL`, or above the `LEVEL`.

2.13.1 Method getInstance

Syntax

```
public static Logger getInstance(java.lang.String
tag)
```

Description

This method gets or creates an instance of this logger. If an instance already exists for the `tag` parameter, that instance is returned.

Parameters

Type	Name	Description
String	`tag`	The package or tag that must be printed with log messages. The value is passed through to `android.util.Log` and recorded when log capture is enabled.

Table 2-20: Method getInstance parameters

2.13.2 Method setContext

Syntax

```
public static void
setContext(android.content.Context _context)
```

Description

The `context` object must be set in order to use the `Logger` API.

Parameters

Type	Name	Description
Object	`_context`	Android Context object

Table 2-21: Method setContext parameters

2.13.3 Method setLevel

Syntax

```
public static void setLevel(Logger.LEVEL
desiredLevel)
```

Description

This method sets the level from which log messages must be saved and printed. For example, passing `LEVEL.INFO` logs `INFO`, `WARN`, and `ERROR`. A null parameter value is ignored and has no effect.

Parameters

Type	Name	Description
`Logger.LEVEL`	`desiredLevel`	The valid values of this input parameter are `LEVEL.DEBUG`, `LEVEL.ERROR`, `LEVEL.INFO`, `LEVEL.LOG`, and `LEVEL.WARN`.

Table 2-22: Method setLevel parameters

2.13.4 Method getLevel

Syntax

```
public static Logger.LEVEL getLevel()
```

Description

This method gets the current `Logger.LEVEL` and returns `Logger.LEVEL`.

2.13.5 Method setCapture

Syntax

```
public static void setCapture(boolean _capture)
```

Description

Global setting: turn persisting of the log data passed to the log methods of this class, on or off.

Parameters

Type	Name	Description
`Boolean`	`_capture`	Flag to indicate if the log data must be saved persistently

Table 2-23: Method setCapture parameters

2.13.6 Method getCapture

Syntax

```
public static boolean getCapture()
```

Description

This method gets the current value of the capture flag, indicating that the `Logger` is recording log calls persistently. This method returns the current value of the capture flag.

2.13.7 Method setMaxFileSize

Syntax

```
public static void setMaxFileSize(int bytes)
```

Description

This method sets the maximum size of the local log file. When the maximum file size is reached, no more data is appended. This file is sent to a server.

Parameters

Type	Name	Description
`Int`	`bytes`	The maximum size of the file in bytes, the minimum is 10,000 bytes.

Table 2-24: Method setMaxFileSize parameters

2.13.8 Method getMaxFileSize

Syntax

```
public static int getMaxFileSize()
```

Description

This method gets the current setting for the maximum file size threshold.

2.13.9 Method send

Syntax

```
public static void send()
```

Description

This method sends the log file when the log buffer exists and is not empty.

2.13.10 Method sendIfUnCaughtExceptionDetected

Syntax

```
public static void
sendIfUnCaughtExceptionDetected(android.content.Con
text context)
```

Description

This method sends the log file when there is an uncaught exception detected, which was recorded to the log buffer due to the capture being turned on at the time of the uncaught exception.

This is a convenience method so that callers might place a single line of code at the point in their application where they want to call it.

Parameters

Type	Name	Description
android.content.Context	context	The Android Context in which this method must be run, typically an Activity instance.

Table 2-25: Method sendIfUnCaughtExceptionDetected parameters

2.13.11 Method info

Syntax

```
public void info(java.lang.String message)


public void info(java.lang.String message,
                 org.json.JSONObject
additionalMetadata)
```

Description

This method logs at `INFO` level.

Parameters

Type	Name	Description
String	`message`	The message to log.
org.json.JSONObject	`additionalMetadata`	The metadata to append to the log output.

Table 2-26: Method info parameters

2.13.12 Method debug

Syntax

```
public void debug(java.lang.String message)


public void debug(java.lang.String message,
                  org.json.JSONObject
additionalMetadata)
```

Description

This method logs at `DEBUG` level.

Parameters

Type	Name	Description
String	`message`	The message to log.
org.json.JSONObject	`additionalMetadata`	The metadata to append to the log output.

Table 2-27: Method debug parameters

2.13.13 Method error

Syntax

```
public void error(java.lang.String message)

public void error(java.lang.String message,
                  org.json.JSONObject
additionalMetadata)

public void error(java.lang.String message,
                  java.lang.Throwable t)

public void error(java.lang.String message,
                  org.json.JSONObject
additionalMetadata,
                  java.lang.Throwable t)
```

Description

This method logs at `ERROR` level.

Parameters

Type	Name	Description
`String`	`message`	The message to log.
`org.json.JSONObject`	`additionalMetadata`	The metadata to append to the log output.
`Throwable`	`t`	a Throwable whose stack trace is converted to string and logged, and passed through as-is to `android.util.Log`.

Table 2-28: Method error parameters

2.13.14 Method log

Syntax

```
public void log(java.lang.String message)

public void log(java.lang.String message,
                org.json.JSONObject
additionalMetadata)
```

Description

This method logs at `LOG` (Android VERBOSE) level.

Parameters

Type	Name	Description
`String`	`message`	The message to log.
`org.json.JSONObject`	`additionalMetadata`	The metadata to append to the log output.

Table 2-29: Method log parameters

2.13.15 Method warn

Syntax

```
public void warn(java.lang.String message)

public void warn(java.lang.String message,
                 org.json.JSONObject
additionalMetadata)
```

Description

This method logs at `WARN` level.

Parameters

Type	Name	Description
`String`	`message`	The message to log.
`org.json.JSONObject`	`additionalMetadata`	The metadata to append to the log output.

Table 2-30: Method warn parameters

2.13.16 Method doLog

Syntax

```
public void doLog(Logger.LEVEL calledLevel,
                  java.lang.String message,
                  long timestamp,
                  org.json.JSONObject
additionalMetadata,
                  java.lang.Throwable t)
```

Description

2.14 Enum Logger.LEVEL

Definition

```
public static final Logger.LEVEL DEBUG
public static final Logger.LEVEL ERROR
public static final Logger.LEVEL INFO
public static final Logger.LEVEL LOG
public static final Logger.LEVEL WARN
```

Description

The following list shows the various levels of logging that are supported in the `Logger` class:

- DEBUG
- ERROR
- INFO
- LOG
- WARN

2.14.1 Method values

Syntax

```
public static Logger.LEVEL[] values()
```

Description

This method returns an array that contains the constants of this enum type, in the order they are declared. This method might be used to iterate over the constants as follows:

```
for (Logger.LEVEL c : Logger.LEVEL.values())
    System.out.println(c);
```

2.14.2 Method valueOf

Syntax

```
public static Logger.LEVEL valueOf(java.lang.String
name)
```

Description

This method returns the enum constant of this type with the specified name. The string must match exactly an identifier used to declare an enum constant in this type. Extraneous whitespace characters are not permitted.

Parameters

Type	Name	Description
String	name	The name of the enum constant to be returned.

Table 2-31: Method setLevel parameters

2.14.3 Method fromString

Syntax

```
public static Logger.LEVEL
fromString(java.lang.String level)
```

Description

This method gets the `LEVEL` enum from the level String parameter, or returns `null` if not found.

The following example returns `LEVEL.ERROR` enum:

```
LEVEL.fromString("ERROR");
```

Parameters

Type	Name	Description
String	level	A string that represents the LEVEL enum.

Table 2-32: Method setLevel parameters

3 Adding the IBM Worklight settings activity to a Native Android application

You can add a standard IBM Worklight Preferences screen to your application. With this screen, users can view and modify the URL of the Worklight Server with which the application communicates. Adding the screen is beneficial for demonstrations and testing scenarios with multiple environments, and multiple servers.

Follow these steps to add the standard IBM Worklight settings activity to your application:

3.1 Changing the manifest.xml File

You must declare the activity in your `manifest.xml` file:

```
<!-- Preferences Activity -->
<activity android:name="com.worklight.common.WLPreferences"
   android:label="Worklight Settings">
</activity>
```

3.2 Changing your application code

You must add code to open `WLPreferences` and to receive results from `WLPreferences`. `WLPreferences` returns an Intent object that has two properties:

- `isServerURLChanged`: this property indicates whether the Worklight Server URL in the Preferences activity changed.
- `serverURL`: this property indicates the value of the Worklight Server URL in the Preferences activity

The following sample code uses the `WLPreferences` activity:

```
//code inside parent activity
//Use any code to identify the activity that back from the stack
private static final int WL_PREFERENCES_CODE = 10;


// open the activity
Intent myIntent = new Intent(getApplicationContext(),
   WLPreferences.class);
this.startActivityForResult(myIntent, WL_PREFERENCES_CODE);

//wait for result
```



```
@Override
protected void onActivityResult(int requestCode, int resultCode,
   Intent data) {
   if (requestCode == WL_PREFERENCE_CODE) {
      if (resultCode == RESULT_OK) {
         if (data.getBooleanExtra("isServerURLChanged", false)) {
            // Check here if server changed and init the connection
               to server or reload if necessary
            Log.i("Test Settings","server URL changed to: " +
               data.getStringExtra("serverURL"));
         }
      }
   }
}
```

3.3 Localizing the Preferences Screen

To localize the strings on the Preferences screen, you must define the following strings in your `strings.xml` file:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
...
  <string name="summaryWLServerUrl">Change the Server URL:
   http[s]://[domain or IP address][:port]</string>
  <string name="titleWLServerUrl">Server URL</string>
  <string name="networkSettingsTitleWLServerUrl">Network
   Settings</string>
  <string name="OKTitleWLServerUrl">Ok</string>
  <string name="titleInvalidWLServerUrl">Invalid URL</string>
  <string name="errorInvalidWLServerUrl">is not a valid URL. Valid
   format is http[s]://[domain or IP address][:port]</string>
...
</resources>
```

To learn more about Android localization, see the Android developer website.

4 Push notifications

IBM Worklight provides APIs to subscribe and unsubscribe to push notifications. The Worklight Server sends notifications to the Google push server (GCM). The GCM server then sends the notifications to the relevant phones.

To enable push notifications in your application, follow these steps.

1. Add the `<pushSender>` element to the application descriptor of the Native API application.

```
<nativeAndroidApp>
          ..
        <pushSender key="" senderId=""/>
          ..
</nativeAndroidApp>
```

2. Deploy your Native API application.
3. Copy the `gcm.jar` file from your Native API application.
4. Paste this `gcm.jar` file intro the `libs` folder of your native app for Android.
5. Copy the `push.png` icon file from your Native API application.
6. In the `res` folder of your native app for Android, identify the folders with a name that starts with `drawable` (such as `res/drawable` or `res/drawable-ldpi`), and then paste the `push.png` file into each of these folders. This icon file is the icon for your notifications.
7. Change the `AndroidManifest.xml` file as follow:


```
<manifest ..>
    ...
    <uses-permission android:name="android.permission.INTERNET" />
    <!-- Push permissions -->
    <permission
        android:name="<your_app_package>.permission.C2D_MESSAGE"
        android:protectionLevel="signature" />
    <uses-permission
   android:name="<your_app_package>.permission.C2D_MESSAGE"/>
    <uses-permission
   android:name="com.google.android.c2dm.permission.RECEIVE"/>
    <uses-permission android:name="android.permission.WAKE_LOCK"/>
    <uses-permission
   android:name="android.permission.GET_ACCOUNTS"/>
    <uses-permission
   android:name="android.permission.USE_CREDENTIALS"/>
    <uses-permission
   android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
   <application ..>
        <activity ...>
          ..
          <intent-filter>
           <action
   android:name="<your_app_package>.<main_activity_name>.NOTIFICATIO
   N" />
           <category android:name="android.intent.category.DEFAULT"
   />
          </intent-filter>
        </activity>
        <activity
   android:name="com.worklight.wlclient.ui.UIActivity"/>
        <!-- Push Notification Service and Receiver declaration -->
        <service
   android:name="com.worklight.wlclient.push.GCMIntentService"/>
        <receiver
android:name="com.worklight.wlclient.push.WLBroadcastReceiver"

   android:permission="com.google.android.c2dm.permission.SEND" >
```



```
<!-- Receive the actual message -->
            <intent-filter>
                <action
   android:name="com.google.android.c2dm.intent.RECEIVE" />
                <category android:name="<your_app_package>" />
            </intent-filter>
            <!-- Receive the registration id -->
            <intent-filter>
                <action
   android:name="com.google.android.c2dm.intent.REGISTRATION" />
                <category android:name="<your_app_package>" />
            </intent-filter>
        </receiver>
    </application>
</manifest>
```

8. In the copy of the `wlclient.properties` file, uncomment the `GCMSenderID` property and assign your GCM sender ID to it.
9. The activity lifecycle methods must be overridden as follow:
   a. `onDestroy()` must call `unregisterReceivers()` of push instance to avoid receiver leak exception when the app exits.
   b. `onPause()` must call `setForeground(false)` of push instance to receive the notification in the notification bar when the app is paused.
   c. `onResume()` must call `setForeground(true)` of push instance to receive the notification in the callback of the app.

4.1 Class WLPush

This class exposes the methods that are required for push notifications.

4.1.1 Method setOnReadyToSubscribeListener

Syntax

```
public void
setOnReadyToSubscribeListener(WLOnReadyToSubscribeL
istener listener)
```

Description

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

Parameters

Type	Name	Description
WLOnReadyToSubscribeListener	`listener`	Mandatory. When the device is ready to subscribe to push notifications, the `WLOnReadyToSubscribeListener.onReadyToSubscribe` method is called.

Table 4-1: Method setOnReadyToSubscribeListener parameters

4.1.2 Method registerEventSourceCallback

Syntax

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

Description

This method registers an `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 4-2: Method registerEventSourceCallback parameters

4.1.3 Method subscribe

Syntax

```
public void subscribe(String alias,WLPushOptions
pushOptions,WLResponsListener responseListener)
```

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	Optional. This instance contains the custom subscription parameters that the event source in the adapter supports.
WLResponseListener	responseListener	Optional. The listener object, whose callback methods, onSuccess and onFailure, are called.

Table 4-3: Method subscribe parameters

4.1.4 Method isSubscribed

Syntax

```
public boolean 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 4-4: Method isSubscribed parameters

4.1.5 Method isPushSupported

Syntax

```
public boolean isPushSupported()
```

Description

This method checks whether push notification is supported.

4.1.6 Method unsubscribe

Syntax

```
public void unsubscribe(String
alias,WLResponsListener responseListener)
```

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.
WLResponseListener	responseListener	Optional. The listener object whose callback methods, onSuccess and onFailure, are called.

Table 4-5: Method unsubscribe parameters

4.1.7 Method setForeground

Syntax

```
public void setForeground(boolean isForeground)
```

Description

This method signals the state of the application to the intent service. The Activity class must override the lifecycle methods `onPause()` and `onResume()`, and call this API with the parameter **false**, or **true**.

Parameters

Type	Name	Description
boolean	isForeground	Mandatory. The foreground status of the application. If **true**, the application is in a foreground state, else in a background state.

Table 4-6: Method setForeground parameters

4.1.8 Method isForeground

Syntax

```
public boolean isForeground()
```

Description

This method returns the foreground status of the application.

4.1.9 Method unregisterReceivers()

Syntax

```
public void unregisterReceivers()
```

Description

This method unregisters the receivers that the notification framework set. The Activity class must override the lifecycle method `onDestroy()` and call this API to avoid receiver leak exception when the app exits.

4.2 Class WLPushOptions

This class contains the subscription parameters.

4.2.1 Method addSubscriptionParameter

Syntax

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

Description

You 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 4-7: Method addSubscriptionParameter parameters

4.2.2 Method addSubscriptionParameters

Syntax

```
public void
addSubscriptionParameters(Map<String,String>
parameters)
```

Description

You use this method to add subscription parameters.

Parameters

Type	Name	Description
Map<String,String>	parameters	Mandatory. The map that contains the list of subscription parameters.

Table 4-8: Method addSubscriptionParameters parameters

4.2.3 Method getSubscriptionParameter

Syntax

```
public String getSubscriptionParameter(String name)
```

Description

This method returns the value of the given subscription parameter.

Parameters

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

Table 4-9: Method getSubscriptionParameter parameters

4.2.4 Method getSubscriptionParameters

Syntax

```
public Map<String,String> getSubscriptionParame-
ters()
```

Description

This method returns the map that contains the subscription parameters.

4.3 Interface WLOnReadyToSubscribeListener

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

4.3.1 Method onReadyToSubscribe

Syntax

```
public void onReadyToSubscribe()
```

Description

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

4.4 Interface WLEventSourceListener

This interface defines the method that receives the notification message.

4.4.1 Method onReceive

Syntax

```
public void onReceive(String props,String payload)
```

Description

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

Parameters

Type	Name	Description
String	`props`	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 4-10: Method onReceive parameters

Appendix A - Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Dept F6, Bldg 1
294 Route 100
Somers NY 10589-3216
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

Privacy Policy Considerations

IBM Software products, including software as a service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies that collect session information (generated by the application server). These cookies contain no personally identifiable information and are required for session management. Additionally, persistent cookies may be randomly generated to recognize and manage anonymous users. These cookies also contain no personally identifiable information and are required.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent. For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at http://www.ibm.com/privacy and IBM's Online Privacy Statement at http://www.ibm.com/privacy/details/en/us sections entitled "Cookies, Web Beacons and Other Technologies" and "Software Products and Software-as-a-Service".

Appendix B - 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

