IBM Worklight

IBM Worklight V5.0.5

Java client-side API for native Android apps

18 January 2013

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

1 API overview 1

2 API reference 4

2.1 Example Code 4
2.1.1 Example: connecting to the Worklight Server and calling a procedure 4
2.2 Class WLClient 6
2.2.1 Method createInstance 6
2.2.2 Method getInstance 7
2.2.3 Deprecated method init 7
2.2.4 Method connect 7
2.2.5 Method invokeProcedure 8
2.2.6 Method logActivity 8
2.2.7 Method checkForNotifications 9
2.2.8 Method registerChallengeHandler 9
2.2.9 Method addGlobalHeader 12
2.2.10 Method removeGlobalHeader 13
2.3 Class ChallengeHandler 13
2.3.1 Method isCustomResponse 13
2.3.2 Method handleChallenge 14
2.3.3 Method submitFailure 14
2.3.4 Method submitSuccess 14
2.3.5 Method submitLoginForm 15
2.3.6 Method submitAdapterAuthentication 16
2.3.7 Method onSuccess 16
2.3.8 Method onFailure 17
2.4 Class WLProcedureInvocationData 20
2.4.1 Method setParameters 20
2.5 Class WLRequestOptions 20
2.5.1 Methods getTimeout, setTimeout 20
2.5.2 Methods getInvocationContext, setInvocationContext 21
2.6 Interface WLResponseListener 21
2.6.1 Method onSuccess 21
2.6.2 Method onFailure 22
2.7 Class WLResponse 22
2.7.1 Method getStatus 22
2.7.2 Method getInvocationContext 23
2.7.3 Method getResponseText 23
2.8 Class WLFailResponse 23
2.8.1 Method getErrorCode 23
2.8.2 Method getErrorMsg 23
2.9 Class WLProcedureInvocationResult 24
2.9.1 Method isSuccessful 24
2.10 Class WLProcedureInvocationFailResponse 24

2.10.1 Method getProcedureInvocationErrors 24
2.10.2 Method getResult 24
2.11 Enum WLErrorCode 24
2.12 Class WLCookieExtractor 25
2.12.1 Static member cookies 25
3 Adding the IBM Worklight Settings activity to a Native Android Application 26
3.1 Changing the manifest.xml File 26
3.2 Changing your application code 26
3.3 Localizing the Preferences Screen 27
Appendix A - Notices 28
Appendix B - Support and comments 30

Tables

Table 1-1: IBM Worklight Java API for Android packages, classes, interfaces, and files 3
Table 2-1: WLClient instantiation.....7
Table 2-2: Method connect parameters 7
Table 2-3: Method invokeProcedure parameters.....8
Table 2-4: Method logActivity parameters.....8
Table 2-5: Method addGlobalHeader parameters.....13
Table 2-6: Method removeGlobalHeader parameters.....13
Table 2-7: Method removeGlobalHeader parameters.....14
Table 2-8: Method handleChallenge parameters.....14
Table 2-9: Method submitFailure parameters 14
Table 2-10: Method submitSuccess parameters.....15
Table 2-11: Method submitLoginForm parameters 15
Table 2-12: Method submitAdapterAuthentication parameters.....16
Table 2-13: Method onSuccess parameters 17
Table 2-14: Method onFailure parameters.....17
Table 2-15: Method setParameters parameters.....20
Table 2-16: Methods getTimeout, setTimeout parameters 21
Table 2-17: Methods getInvocationContext, setInvocationContext parameters 21
Table 2-18: Method onSuccess parameters 22
Table 2-19: Method onSuccess parameters 22

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 four main capabilities:

- Calling back-end services for retrieving data and performing 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 to allow for a customized authentication process.

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 for using 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 for communicating 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 holds all data necessary for calling 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 are arriving from the Worklight Server.	IBM
Class	`WLCookieExtractor`	Class that provides access to external cookies that `WLClient` can use when it is issuing requests to the Worklight Server. This class is used to share session cookies between a web view and a natively implemented page.	IBM
Class	`WLPreference`	Class that implements a preferences activity for viewing and modifying 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
Package	`com.worklight.wlclient.ui`	Package that holds an activity that is used by the platform to display UI.	IBM

Type	Name	Description	Implemented by
Class	`UIActivity`	Android Activity class that is used by the IBM Worklight platform 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 to 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 examples show code for using 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 class exposes methods for communicating with the Worklight Server. This class is a singleton. It has a single instance which is created only once and accessed statically.

2.2.1 Method createInstance

Syntax

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

Description

This method creates the singleton instance of `WLClient`.

Parameters

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

Table 2-1: WLClient instantiation

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.

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

Note: This method is deprecated. Use connect instead.

2.2.4 Method connect

Syntax

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

Description

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

Important: This method must be called 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

This method is used to check for notifications on the server, such as new block/notify rules, notifications and so on. Calling this method from the `onResume Android Activity` lifecycle event results in the application checking for new notifications when the activity is brought to the foreground.

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 default behavior (implemented in the method `handleFailure` of `RemoteDisableChallengeHandler`) is to show a dialog with the appropriate message. It can also show a link to download the new application version. After the dialog is closed, the application continues to work offline. You must implement a similar behavior in the `handleFailure` code of the customized Remote Disable Challenge Handler.

2.2.9 Method addGlobalHeader

Syntax

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

Description

This method is used 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

This method is used 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

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

2.3.1 Method isCustomResponse

Syntax

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

Description

This method must be overridden by the extending class of `ChallengeHandler`. 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 removeGlobalHeader parameters

2.3.2 Method handleChallenge

Syntax

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

Description

You must implement this method to handle the challenge logics, for example to show the login screen. The method is called by the IBM Worklight framework 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 submits the login form successfully. 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

This method is used 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 to be used. 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

This method is used 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`	Holds the following options. `timeout – int`: 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 holds all data necessary for calling a procedure, including:

- The name of the adapter and procedure to call.
- The parameters that are required by the procedure.

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 currently used request timeout (default is 10 sec).

`setTimeout`: this method sets a new timeout.

Parameters

Type	Name	Description
int	`timeout`	Timeout in milliseconds for waiting for the procedure invocation. If the timeout expires, the `WLResponseListener onFailure` method is called. The value 0 indicates 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
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 is returned from the server, 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
`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 onSuccess 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 calling invokeProcedure.

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 error codes and messages in addition to the status in WLResponse. It contains the original response DataObject from the server as well.

2.8.1 Method getErrorCode

Syntax

```
public WLErrorCode getErrorCode ()
```

Description

The possible errors are described in the WLErrorCode section.

2.8.2 Method getErrorMsg

Syntax

```
public String getErrorMsg()
```

Description

This error message is for the developer and not necessarily suitable for the user.

2.9 Class WLProcedureInvocationResult

This class extends `WLResponse`. It holds statuses and data that are retrieved by an adapter procedure.

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`. It holds statuses and data that are retrieved by an adapter procedure.

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 method is calling the procedure.

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 can be used by `WLClient` when it issues requests to the Worklight Server. This class is used 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 are shared by the `WLCookieExtractor`. They can be accessed statically.

3 Adding the IBM Worklight Settings activity to a Native Android Application

You can add a standard IBM Worklight Preferences screen to your application. This screen enables users to 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

- 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

- Add code to open `WLPreferences` and to receive results from `WLPreferences`. The Intent object that is returned from `WLPreferences` has two properties:
  - `isServerURLChanged` – indicates whether the Worklight Server URL in the Preferences activity changed
  - `serverURL` – 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

You can localize the strings on the Preferences screen by defining 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.

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.

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

