IBM Worklight

IBM Worklight V5.0.6

Java client-side API for Java Platform, Micro Edition

15 March 2013

About IBM®

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

Contents

Tables

About this document

This document is intended for Java™ Platform, Micro Edition (Java ME) developers who want to access IBM® Worklight® services from Java ME applications. The document guides you through the available classes and methods.

1 API overview

The IBM Worklight Java client-side API for Java Platform, Micro Edition (Java ME), exposes four main capabilities:

- Calling back-end services to send and 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.

The IBM Worklight Java client-side API for Java ME 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 Worklight SDK.	IBM
Package	`com.worklight.wlclient.api`	All API classes are defined in this package. You must import this package in the Java ME 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	`WLProcedureInvocationData`	Class that contains all data necessary for calling a procedure.	IBM
Class	`WLRequestOptions`	Class that you use to add request parameters, headers, and invocation context.	IBM
Interface	`WLResponseListener`	Interface that defines methods that a listener for the `WLClient invokeProcedure` method implements to receive notifications about the success or failure of the method call.	Application developer
Class	`WLResponse`	Class that contains the result of a procedure invocation.	IBM

Type	Name	Description	Implemented By
Class	`WLFailResponse`	Class that extends `WLResponse`. This class contains error codes and messages in addition to the status in `WLResponse`. This class also contains the original response DataObject from the server.	IBM
Class	`WLProcedureInvocationResult`	Class that extends `WLResponse`. This class contains the result of calling a back-end service, which includes status and data items that the adapter function retrieves from the server.	IBM
Class	`WLProcedureInvocationFailResponse`	Class that extends `WLFailResponse` and that you can use to retrieve the invocation error messages.	IBM
Class	`WLErrorCode`	Class that contains an error code and its message that arrive from the Worklight Server.	IBM
Class	`WLHeader`	Class that contains the name of the header and its value that you send with the request.	IBM
Package	`com.worklight.wlclient.api.challengehandler`	Package that defines Challenge Handler classes that you must use 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 this class to implement your own version of an IBM Worklight Challenge Handler, for example `RemoteDisableChallengeHandler`.	IBM
Class	`ChallengeHandler`	Abstract base class for the custom Challenge Handlers. You must extend this class to implement custom authentication.	IBM

Table 1-1: IBM Worklight Java client-side API for Java ME – 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 Java ME.

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

Initializing the IBM Worklight Client

```
WLClient client = WLClient.createInstance(this);
client.connect(new MyConnectResponseListener());
```

Implementation of a Response Listener for connect

```
public class MyConnectResponseListener implements WLResponseListener{

  public void onFailure(WLFailResponse response) {
    System.out.println("Response fail: " + response.getErrorMsg());
  }

  public void onSuccess(WLResponse response) {
    WLProcedureInvocationData invocationData = new
WLProcedureInvocationData("myAdapterName", "myProcedureName");

    invocationData.setParameters(new Object[]{"stringParam"});

    String myContextObject = new String("This is my context object");

    WLRequestOptions options = new WLRequestOptions();
    options.setInvocationContext(myContextObject);

    WLClient.getInstance().invokeProcedure(invocationData, new
MyInvokeListener(), options);
  }
}
```


Implementation of a Response Listener for Procedure Invocation

```
public class MyInvokeListener implements WLResponseListener {

  public void onFailure(WLFailResponse response) {
    System.out.println("Response failed: " + response.getErrorMsg());
  }

  public void onSuccess(WLResponse response) {
    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) {

            }
      }
  }
}
```

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(MIDlet
midlet)
```

```
public static WLClient createInstance(String
connectionString,MIDlet midlet)
```

Deprecated

```
public static WLClient createInstance()
```

```
public static WLClient createInstance(String con-
nectionString)
```

Description

These methods create 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 application.

If the client device is BlackBerry, connection parameters such as `deviceside=true,interface =wifi`, or any name-value pairs that you can use to identify connection type can be passed as string arguments. For other devices, you can set the string argument to `null`.

Parameters

Type	Name	Description
`String`	`connectionString`	Specifies the connection string to be used to connect to the server from a BlackBerry device. For other devices, it can be set to `null`.
`MIDlet`	`midlet`	This parameter is the midlet instance, for example the midlet 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 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. If the Worklight server runs in secured mode (over `https`), then ensure that the security certificate that the server uses is imported to the device else the connection will fail.

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

2.2.4 Method invokeProcedure

Syntax

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

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 `onSuccess` and `onFailure` are called.
`WLRequestOptions`	`requestOptions`	Optional. Invocation options.

Table 2-3: Method invokeProcedure parameters

2.2.5 Method logActivity

Syntax

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


Description

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

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

Parameters

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

Table 2-4: Method logActivity parameters

2.2.6 Method setHeartBeatInterval

Syntax

```
public void setHeartBeatInterval (int newInterval)
```

Description

This method sets the interval, in seconds, at which the heartbeat signal is sent to the Worklight Server. You use the heartbeat signal to ensure that the session with the server is kept alive when the app does not issue any call to the server, such as `invokeProcedure`. By default, the interval is set to 7 minutes.

Parameters

Type	Name	Description
`int`	`newInterval`	An integer value that defines the interval in seconds between the heartbeat messages that `WLClient` automatically sends to the Worklight Server. To disable the heartbeat, set a value that is less than, or equal to zero.

Table 2-5: Method setHeartBeatInterval parameters

2.2.7 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`.

Parameters

Type	Name	Description
BaseChallengeHandler	challengeHandler	Custom challenge handler instance.

Table 2-6: Method registerChallengeHandler parameters

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 name it with 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 the class `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 closes. You must implement a similar behavior in the `handleFailure` code of the custom Remote Disable Challenge Handler.

2.2.8 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 2-7: Method addGlobalHeader parameters

2.2.9 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.
String	value	The value of the header.

Table 2-8: 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 logic. You use this class mainly 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 2-9: Method isCustomResponse parameters

2.3.2 Method handleChallenge

Syntax

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

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 2-10: 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 2-11: 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 2-12: Method submitSuccess parameters

2.3.5 Method submitLoginForm

Syntax

```
protected void submitLoginForm(String requestURL,
Hashtable requestParameters, Hashtable
requestHeaders,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.
`Hashtable`	`requestParameters`	The request parameters.
`Hashtable`	`requestHeaders`	The request headers.
`String`	`requestMethod`	The `HTTP` method that you must use. Acceptable values are `GET`, `POST`.

Table 2-13: 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. `invocationContext – Object:` An object that is returned with `WLResponse` to the delegate methods. You can use this object to distinguish different `invokeProcedure` calls.

Table 2-14: 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 2-15: Method onSuccess parameters

2.3.8 Method onFailure

Syntax

```
public void onFailure(WLResponse response)
```

Description

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

Parameters

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

Table 2-16: Method onFailure parameters

Example: implementing a form-based Challenge Handler

```
/*
 * Register the custom handler in the midlet
 */
 public class FormBasedAuthentication extends MIDlet {
   public FormBasedAuthentication () {
     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.
  */
 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
  */
  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, "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.
 */
 public void onSuccess(WLResponse response) {
  if (isCustomResponse(response)) {
     handleChallenge(response);
  } else {
```



```
      submitSuccess(response);
   }
  }
  /*
   * onFailure is called in case of any error/exceptions
   * WLErrorCode is set to appropriate error codes
   */
  public void onFailure(WLFailResponse response) {
   submitFailure(response);
  }
}
```

2.4 Class WLProcedureInvocationData

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

- The names 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 2-17: Method setParameters parameters

Example

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

2.5 Class WLRequestOptions

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

2.5.1 Method addParameter

Syntax

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

Description

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

Parameters

Type	Name	Description
String	name	The name of the parameter.
String	value	The value of the parameter.

Table 2-18: Method addParameter parameters

2.5.2 Method addParameters

Syntax

```
public void addParameters(Hashtable parameters)
```

Description

This method adds a table of request parameters.

Parameters

Type	Name	Description
Hashtable	parameters	The request parameters table.

Table 2-19: Method addParameters parameters

2.5.3 Method getParameter

Syntax

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

Description

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

Parameters

Type	Name	Description
String	name	The name of the parameter.

Table 2-20: Method getParameter parameters

2.5.4 Method getParameters

Syntax

```
public Hashtable getParameters()
```

Description

This method returns the parameters table.

2.5.5 Method getResponseListener

Syntax

```
public WLResponseListener getResponseListener()
```

Description

This method returns the response listener for this request.

2.5.6 Method addHeader

Syntax

```
public void addHeader(WLHeader header)

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

Description

You can use these methods to add a header or a header with the given name and value.

Parameters

Type	Name	Description
WLHeader	header	The header to be added.
String	name	The name of the header.
String	value	The value of the header.

Table 2-21: Method addHeader parameters

2.5.7 Method setHeaders

Syntax

```
public void setHeaders(Vector extraHeaders)
```

Description

This method sets the request with the headers of type `WLHeader` from the given vector.

Parameters

Type	Name	Description
`Vector`	`extraHeaders`	The headers to be set.

Table 2-22: Method setHeaders parameters

2.5.8 Method getHeaders

Syntax

```
public Vector getHeaders()
```

Description

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

2.5.9 Methods getInvocationContext, setInvocationContext

Syntax

```
public Object getInvocationContext()
```

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

Parameters

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

Table 2-23: 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 after 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 2-24: 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 2-25: 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.

2.7.4 Method getResponseJSON

Syntax

```
public JSONObject getResponseJSON()
```

Description

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

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 error codes.

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 getResult

Syntax

```
public JSONObject getResult()
```

Description

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

2.9.2 Method isSuccessful

Syntax

```
public boolean isSuccessful()
```

Description

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

2.10 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 Class WLErrorCode

This class contains error codes and their description, which the server returns.

2.11.1 Method getDescription

Syntax

```
public String getDescription()
```

Description

This method returns the description of this error code instance.

2.11.2 Method valueOf

Syntax

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

Description

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

Error Codes

`UNEXPECTED_ERROR` - Unexpected `errorCode` occurred. Try again.

`REQUEST_TIMEOUT` – The request timed out.

`UNRESPONSIVE_HOST` - The service is currently unavailable.

`PROCEDURE_ERROR` - Procedure invocation `errorCode`.

`PROCEDURE_PROTECTED_ERROR` – The procedure is protected.

`APP_VERSION_ACCESS_DENIAL` - Application version denied.

`APP_VERSION_ACCESS_NOTIFY` - Notify application version changed.

2.12 Class WLHeader

This class creates a header that is sent with the request

2.12.1 Method getHeaderName

Syntax

```
public String getHeaderName()
```


Description

This method returns the name of this header.

2.12.2 Method getHeaderValue

Syntax

```
public String getHeaderValue()
```

Description

This method returns the value of this header.

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

