IBM Worklight

IBM Worklight V5.0.5

Objective-C client-side API for native iOS 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.

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 iPhone and iPad developers who want to access IBM® Worklight® services from native iOS applications written in Objective-C. The document guides you through the classes and methods available.

1 API overview

The IBM Worklight Objective-C client-side API for native iOS apps exposes three main capabilities:

- Calling back-end services for retrieving data and performing back-end transactions.
- Writing custom log lines for reporting and auditing purposes.
- Writing custom Challenge Handlers to enable user authentication.

Type	Name	Description	Implemented By
File	`worklight.plist`	Property list file that contains the necessary information for initializing `WLClient`.	Application developer
Class	`WLClient`	Singleton class that exposes methods for communicating with the Worklight Server, in particular `invokeProcedure` for calling a back-end service and push notification subscription services.	IBM
Class	`WLProcedure InvocationData`	Class that holds all data necessary for calling a procedure.	IBM
Protocol	`WLDelegate`	Class that defines methods that a delegate 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 call.	IBM
Class	`WLFailResponse`	Class that is derived from `WLResponse` and 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	`WLProcedure InvocationResult`	Class that contains the result of calling a back-end service, including statuses and data items that are retrieved by the adapter function from the server.	IBM
Class	`WLCookieExtractor`	Class that is used to share cookies between the web code and native code in the application.	IBM

Type	Name	Description	Implemented By
Class	`ChallengeHandler`	Base class that must be overridden to define custom challenge handling for user custom authentication.	Application developer

Table 1-1: IBM Worklight Objective C API for iOS classes, protocols, and files

2 API reference

2.1 Example Code

The following examples show code for using the IBM Worklight Objective-C client-side API. All API classes, methods, and enums are described after these examples.

2.1.1 Example 1: calling a back-end service that does not require authentication

MyClass.m

```
-(void) someMethod{
    …
    WLDelegate *connectDelegate = [MyConnectDelegate new];
    [[WLClient sharedInstance]
   wlConnectWithDelegate:connectDelegate];
}
```

MyConnectDelegate.m <WLDelegate>


```
/**
 * called if connectDelegate succeeded
 */
-(void) onSuccess(WLResponse *)response {
  // initialize a procedureInvocationData object
  WLProcedureInvocationData *invocationData =
       [[WLProcedureInvocationData alloc]
         initWithAdapter:@"demoAdapter"
   procedure:@"getDemoAccount"];
    [invocationData setParameters:
         [NSArray arrayWithObjects:@"123-456-789", @"california",
   nil]];

  // invoke the procedure
  WLDelegate *invokeProcedureDelegate =
       [MyInvokePRocedureDelegate new];
  [[WLClient sharedInstance] invokeProcedure:invocationData
       withDelegate:invokeProcedureDelegate];
}
```

MyInvokeProcedureDelegate <WLDelegate>


```
/**
 * called if invokeProcedure succeeded
 */
-(void)onSuccess:(WLResponse *)response{
    // status
    NSLog(@"Response status is %@", [response status]);

    // print the response data
    NSLog(@"Invocation response success status: %d. Invocation
   result data is %@",
         [[response invocationResult] isSuccessful],
         [[response invocationResult] getResponse]);
}

/**
 * called if invokeProcedure failed
 */
-(void)onFailure:(WLFailResponse *)failResponse{
    // status
    NSLog(@"Response status is %@". Error code %@ (%@)., [response
   status],
    failResponse errorCode],
    [failResponse errorMsg]);
}
```

2.1.2 Example 2: calling a back-end service that requires authentication

```
-(void) someMethod{
   ...

    [[WLClient sharedInstance]
      wlConnectWithDelegate:connectDelegate];
}
```

```
/**
 * called if invokeProcedure succeeded
 */
-(void)onSuccess:(WLResponse *)response{
```



```
   // status
   NSLog(@"Response status is %@", [response status]);

   // print the response data
   NSLog(@"Invocation response success status: %d. Invocation result
      data is %@",
      [[response invocationResult] isSuccessful],
      [[response invocationResult] getResponse]);
}

/**
 * called if invokeProcedure failed
 */
-(void)onFailure:(WLFailResponse *)failResponse{
   // status
   NSLog(@"Response status is %@". Error code %@ (%@)., [response
      status], [failResponse errorCode], [failResponse errorMsg]);
}
```

2.2 Class WLClient

This class exposes methods for communicating with the Worklight Server.

2.2.1 Method wlConnectWithDelegate

Syntax

```
-(void)wlConnectWithDelegate:(WLDelegate *)delegate;
```

Description

This method initializes communication with the Worklight Server by using the connection properties and application ID taken from the `worklight.plist` file. The server checks the validity of the application version.

Note: This method must be called before any other `WLClient` method that calls the server such as `logActivity` and `invokeProcedure`.

If the server returns a successful response, the `onSuccess` method is called. If an error occurs, the `onFailure` method is called.

Parameters

Name	Type	Description
delegate	`WLDelegate`	A class that conforms to the `WLDelegate` protocol.

Table 2-1: Method wlConnectWithDelegate parameters

2.2.2 Method wlConnectWithDelegate:cookieExtractor:

Syntax

```
-(void) wlConnectWithDelegate:(id <WLDelegate>)delegate
   cookieExtractor:(WLCookieExtractor *) cookieExtractor;
```

Description

This method initializes communication with the Worklight Server by using the connection properties and application ID taken from the `worklight.plist` file. The server checks the validity of the application version.

Note: This method must be called before any other `WLClient` method that calls the server such as `logActivity` and `invokeProcedure`.

If the server returns a successful response, the `onSuccess` method is called. If an error occurs, the `onFailure` method is called.

Parameters

Name	Type	Description
delegate	`WLDelegate`	A class that conforms to the `WLDelegate` protocol.
cookieExtractor	`WLcookieExtractor`	Optional; can be `nil`. This parameter is used to share the cookies between the native code and the web code in the application.

Table 2-2: Method wlConnectWithDelegate:cookieExtractor parameters

Example


```
-(void) someMethod{
   …
   WLDelegate *connectDelegate = [MyConnectDelegate new];
   [[WLClient sharedInstance] wlConnectWithDelegate:connectDelegate
      cookieExtractor:[WLCookieExtractor new]];

}
```

2.2.3 Method invokeProcedure:withDelegate:

Syntax

```
-(void)invokeProcedure:(WLProcedureInvocationData
   *)procedureInvocationData
withDelegate:(id <WLDelegate>)delegate;
```

Description

This method calls an adapter procedure. This method is asynchronous. The response is returned to the callback functions of the provided delegate.

If the call succeeds, `onSuccess` is called. If it fails, `onFailure` is called.

Parameters

Name	Type	Description
procedureInvocationData	`WLProcedureInvocationData`	The invocation data for the procedure call.
delegate	`WLDelegate`	The delegate object that is used for the `onSuccess` and `onFailure` callback methods.

Table 2-3: Method invokeProcedure:withDelegate parameters

2.2.4 Method invokeProcedure:withDelegate:options:

Syntax

```
-(void)invokeProcedure:(WLProcedureInvocationData
   *)procedureInvocationData     withDelegate:(id
   <WLDelegate>)delegate    options:(NSDictionary *) options;
```

Description

This method is similar to `invokeProcedure:options`, with an additional `options` parameter for providing more data for this procedure call.

Parameters

Name	Type	Description
options	`NSDictionary`	A map with the following keys and values: `timeout` – `NSNumber`. Time in milliseconds for this `invokeProcedure` to wait before the request fails with `WLErrorCodeRequestTimeout`. The default timeout is 10 seconds. To disable the timeout, set this parameter to 0. `invocationContext` – an object that is returned with `WLResponse` to the delegate methods. This object can be used to identify and differentiate between `invokeProcedure` calls.

Table 2-4: Method invokeProcedure:withDelegate:options parameters

Example

```
NSNumber *invocationContextCounter = [NSNumber numberWithInt:1];
NSNumber *timeout = [NSNumber numberWithInt:3000];
NSDictionary *options = [NSDictionary dictionaryWithObjectsAndKeys:
   invocationContextCounter, @"invocationContext", timeout,
   @"timeout", nil];
```

2.2.5 Method subscribeWithToken:adapter:eventSource: eventSourceID:notificationTypes:delegate:

Syntax

```
-(void) subscribeWithToken:(NSData *)deviceToken adapter:(NSString
   *)adapter eventSource: (NSString *)eventSource eventSourceID:
   (int)id notificationType:(UIRemoteNotificationType) types
   delegate:(id <WLDelegate>)delegate
```

Description

Calling this method is the same as calling the method subscribeWithToken:adapter:eventSource:eventSourceID:notificationTypes:delegate:options: with options: `nil`.

2.2.6 Method subscribeWithToken:adapter:eventSource: eventSourceID:notificationTypes:delegate:options:

Syntax

```
-(void) subscribeWithToken:(NSData *)deviceToken adapter:(NSString
   *)adapter eventSource: (NSString *)eventSource eventSourceID:
   (int)id notificationType:(UIRemoteNotificationType) types
   delegate:(id <WLDelegate>)delegate
options: (NSDictionary *)options
```

Description

This method subscribes the application to receive Push Notifications from the specified event source and adapter.

Parameters

Name	Type	Description
deviceToken	`NSData`	The token received from the method `application:didRegisterForRemoteNotificationsWithDeviceToken:`. Save the device token in case `unsubscribedWithToken:adapter:eventSource:delegate:` is called.
adapter	`NSString`	Name of the adapter
eventSource	`NSString`	Name of the event source
eventSourceID	`int`	An ID that you assign to the event source that is returned by the Worklight Server with each notification from this event source. You can use the ID in your notification callback function to identify the notification event source. The ID is passed on the notification payload. To save space in the notification payload, pass a short integer, otherwise it is used to pass the adapter and event source names.
notificationType	`UIRemoteNotificationType`	Constants that indicate the types of notifications that the application accepts. For more information, see the Apple documentation.

Name	Type	Description
delegate	`id <WLDelegate>`	A standard IBM Worklight delegate with `onSuccess` and `onFailure` methods to indicate success or failure of the subscription with the Worklight Server.
options	`NSDictionary`	Optional. This parameter contains data that is passed to the Worklight Server, which is used by the adapter.

Table 2-5: Method invokeProcedure:withDelegate:options parameters

2.2.7 Method unsubscribeAdapter: eventSource:delegate:

Syntax

```
-(void) unsubscribeAdapter:(NSString *)adapter eventSource:(NSString
 *)eventSource delegate: (id <WLDelegate>)delegate
```

Description

This method unsubscribes to notifications from the specified event source in the specified adapter.

Parameters

Name	Type	Description
adapter	`NSString`	Name of the adapter
eventSource	`NSString`	Name of the event source
delegate	`id <WLDelegate>`	A standard IBM Worklight delegate with `onSuccess` and `onFailure` methods to indicate success or failure of the Worklight Server unsubscription.

Table 2-6: Method unsubscribeAdapter:eventSource:delegate parameters

2.2.8 Method isSubscribedToAdapter:eventSource:

Syntax

```
-(BOOL) isSubscribedToAdapter:(NSString *)adapter
   eventSource:(NSString *)eventSource;
```

Description

This method returns **true** if the current logged-in user on the current device is already subscribed to the adapter and event source. The method checks the information received from the server in the success response for the login request. If the information that is sent from the server is not received, or if there is no subscription, this method returns **false**.

Parameters

Name	Type	Description
adapter	`NSString`	Name of the adapter
eventSource	`NSString`	Name of the event source

Table 2-7: Method isSubscribedToAdapter:eventSource parameters

2.2.9 Method updateDeviceToken:delegate:

Syntax

```
-(void) updateDeviceToken:(NSData *)deviceToken  delegate:(id
   <WLDelegate>)delegate;
```

Description

This method compares the device token to the one registered in the Worklight Server with the current logged-in user and current device. If the device token is different, the method sends the updated token to the server.

The registered device token from the server is received in the success response for the login request. It is considered to be readily available without the need for an additional server call to retrieve. If a registered device token from the server is not readily available in the application, this method sends an update to the server with the device token.

Parameters

Name	Type	Description
deviceToken	`NSData`	The token received from the method `application:didRegisterForRemoteNotificationsWithDeviceToken:`. Save the device token in case `unsubscribedWithToken:adapter:eventSource:delegate` is called.
delegate	`id <WLDelegate>`	A standard IBM Worklight delegate with `onSuccess` and `onFailure` methods to indicate successful or failure of the device token update with the Worklight Server.

Table 2-8: Method updateDeviceToken:delegate parameters

2.2.10 Method getEventSourceIDFromUserInfo:

Syntax

```
-(int)getEventSourceIDFromUserInfo: (NSDictionary *)userInfo
```

Description

This method returns the `eventSourceID` sent by the Worklight Server in the push notification.

Parameters

Name	Type	Description
userInfo	`NSDictionary*`	The `NSDictionary` received in the `application:didReceiveRemoteNotification` method.

Table 2-9: Method getEventSourceIDFromUserInfo parameters

2.2.11 Method logActivity

Syntax

```
-(void)logActivity:(NSString *)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

Name	Type	Description
activityType	`NSString`	A string that identifies the activity

Table 2-10: Method logActivity parameters

2.2.12 Method registerChallengeHandler

Syntax


```
-(void) registerChallengeHandler:(BaseChallengerHandler *)
   challengeHandler;
```

Description

You can use this method to register a custom Challenge Handler, which is a class that inherits from `ChallengeHandler`. See example 1: *Adding a custom Challenge Handler*.

You can also use this method to override the default Remote Disable / Notify Challenge Handler, by registering a class that inherits from `WLChallengeHandler`. See example 2: *Customizing the Remote Disable / Notify*.

Parameters

Name	Type	Description
challengeHandler	`BaseChallengeHandler`	The Challenge Handler to register.

Table 2-11: Method registerChallengeHandler parameters

Example 1: Adding a custom Challenge Handler

To add a custom Challenge Handler, you must first create it, and then register it on the start point of the application.

```
FormChallengeHandler *formChallengeHandler = [[FormChallengeHandler
alloc] initWithRealm:@"myCustomRealm"];
[[WLClient sharedInstance]
registerChallengeHandler:formChallengeHandler];

//
//  FormChallengeHandler.m

#import "FormChallengeHandler.h"

@implementation FormChallengeHandler


-(void) handleChallenge: (WLResponse *)response {
   NSLog(@"FormChallengeHandler :: handleChallenge");
   // Here you can show login form for example

   // Here is code snippet to handle post submit of the login form:
   NSString *username = @"username";
   NSString *password = @"password";

   NSDictionary *headers = [NSDictionary
   dictionaryWithObjectsAndKeys:@"aaa",@"customHeader1",@"bbb",@"customHeader2",
      nil];
   NSDictionary *params = [NSDictionary dictionaryWithObjectsAndKeys:username,
      @"j_username", password, @"j_password", nil];

   // User can use the the api submitLoginForm or his custom function.
   [self submitLoginForm:@"j_security_check" requestParameters:params
   requestHeaders:headers requestTimeoutInMilliSeconds:30000
   requestMethod:@"POST"];

}

//Failure delegate for submitLoginForm
-(void) onFailure:(WLFailResponse *)response {
   [self submitFailure:response];
   NSLog(@"FormChallengeHandler :: onFailureWithResponse");
```



```
}

//Success delegate for submitLoginForm
-(void)onSuccess:(WLResponse *)response{
   [self submitSuccess:response];
   NSLog(@"FormChallengeHandler :: onSuccessWithResponse");
}

-(BOOL) isCustomResponse:(WLResponse *) response {
   NSRange authRange = [response.responseText rangeOfString:@"my unique
   identifier in the response"];
   if (authRange.length > 0) {
      NSLog(@"FormChallengeHandler :: isCustomResponse");
      return YES;
   }
   return NO;
}
@end
```

Example 2: Customizing the Remote Disable / Notify

To customize the Remote Disable / Notify, you must create an instance of type `WLChallengeHandler`, and then register it in the start point of the application with the specific realm name `wl_remoteDisableRealm`.

```
// Register on application start point
[[WLClient sharedInstance]
registerChallengeHandler:[[CustomRemoteChallengeHandler alloc]
initWithRealm:@"wl_remoteDisableRealm"]];

//
//  CustomRemoteChallengeHandler.m

#import "CustomRemoteChallengeHandler.h"

@implementation CustomRemoteChallengeHandler
-(void) handleFailure: (NSDictionary *)failure {
   // here you get the remote disable data
   //message
   NSString * msg = [failure valueForKey:@"message"];
   //downloadLink to market
   NSString * downloadLink = [failure valueForKey:@"downloadLink"];

   // show your block message and exit application
   ...
}


//Notifying an application
-(void) handleChallenge: (NSDictionary *)challenge{
   // here you get the notification data
   NSString * msg = [challenge valueForKey:@"message"];
   NSString * msgId = [failure valueForKey:@"messageId"];

   //Needs to call setMessageId
   [self setMessageId:msgId]

   // show your message
   ...
   //In the end call to submitAnswer
   [self submitAnswer]
```



```
}

@end
```

2.2.13 Method addGlobalHeader

Syntax

```
-(void) addGlobalHeader: (NSString *) headerName
   headerValue:(NSString *)value;
```

Description

This method is used to add a global header. This header is sent on each request.

Parameters

Name	Type	Description
headerName	NSString	The header name/key
headerValue	NSString	The header value

Table 2-12: Method addGlobalHeader parameters

2.2.14 Method removeGlobalHeader

Syntax

```
-(void) removeGlobalHeader: (NSString *) headerName;
```

Description

This method is used to remove a global header. This header is no longer sent with each request.

Parameters

Name	Type	Description
headerName	NSString	The header name to be removed

Table 2-13: Method removeGlobalHeader parameters

2.3 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.

2.3.1 Method initWithAdapter:procedure:

Syntax

```
-(id)initWithAdapter:(NSString *)adapter procedure:(NSString
   *)procedure
```

Description

This method initializes with the adapter name and the procedure name.

Parameters

Name	Type	Description
adapter	`NSString`	The adapter name
procedure	`NSString`	The name of the adapter procedure

Table 2-14: Method initWithAdapter:procedure parameters

2.3.2 Method setParameters

Syntax

```
-(void)setParameters:(NSArray *) parameters;
```

Description

This method sets the procedure parameters.

Parameters

Name	Type	Description
parameters	`NSArray`	The Array contains Objects that can be parsed by using JSON. `NSString` and `NSNumber` work best. For Boolean values, use `[NSNumber numberWithBool:]`

Table 2-15: Method setParameters parameters

Example


```
NSArray *params = [NSarray arrayWithObjects:@"string",
     [NSNumber numberWithInt:7],
     [NSNumber numberWithFloat:65.878],
     [NSNumber numberWithBool: YES]];
```

2.4 Protocol WLDelegate

Description

This protocol defines methods that a delegate for the `WLClient invokeProcedure/wlConnectWithDelegate` method implements to receive notifications about the success or failure of the method call.

2.4.1 Method onSuccess:

Syntax

```
-(void)onSuccess:(WLResponse *)response;
```

Description

This method is called after a successful call to the `WLCLient invokeProcedure` method. `WLResponse` contains the results from the server, along with any context object and status.

2.4.2 Method onFailure:

Syntax

```
-(void)onFailure:(WLFailResponse *)response;
```

Description

This method is called if any failure occurred during the execution of the `WLCLient invokeProcedure`. The `WLFailResponse` instance contains the error code and error message, and optionally, the results from the server and any context object and status.

2.5 Class WLResponse

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

2.5.1 Property status

Parameters

Name	Type	Description
status	NSString	This property retrieves the HTTP status from the response.

Table 2-16: Property status parameters

2.5.2 Property invocationResult

Parameters

Name	Type	Description
invocationResult	WLProcedureInvocationResult	This property is the response data from the server.

Table 2-17: Property invocationResult parameters

2.5.3 Property invocationContext

Parameters

Name	Type	Description
invocationContext	NSObject	This property is the context object that is passed when the invokeProcedure method is called.

Table 2-18: Property invocationContext parameters

2.5.4 Property responseText

Parameters

Name	Type	Description
responseText	NSString	This property is the original response text from the server.

Table 2-19: Property responseText parameters

2.5.5 Method getResponseJson

Syntax

```
-(NSDictionary *)getResponseJson;
```

Description

This method returns the value `NSDictionary` in case the response is a JSON response, otherwise it returns the value `nil`. `NSDictionary` represents the root of the JSON object.

2.6 Class WLFailResponse

This class is derived from `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.

2.6.1 Property errorMsg

Parameters

Name	Type	Description
errorMsg	`NSString`	This property is the error message for the developer, which is not necessarily suitable for the user.

Table 2-20: Property errorMsg parameters

2.6.2 Property errorCode

Parameters

Name	Type	Description
errorCode	`WLErrorCode`	This property is the error code. Possible errors are described in the `WLErrorCode` section.

Table 2-21: Property errorCode parameters

2.6.3 Property invocationResult

Parameters

Name	Type	Description
invocationResult	`NSObject`	This property represents the call results from the server, which can contain a different object for each callback of `WLClient`, as described in a table under `WLResponse` class.

Table 2-22: Property invocationResult parameters

2.6.4 Property invocationContext

Parameters

Name	Type	Description
invocationContext	NSObject	This property is the context object that is passed when the invokeProcedure method is called.

Table 2-23: Property invocationContext parameters

2.6.5 Property responseText

Parameters

Name	Type	Description
responseText	NSString	This property is the message, which is the original response text from the server.

Table 2-24: Property responseText parameters

2.6.6 Method getErrorMessageFromCode

Syntax

```
+(NSString *) getErrorMessageFromCode: (WLErrorCode) code;
```

Description

This method returns a string message from a WLErrorCode.

2.6.7 Method getErrorMessageFromJSON

Syntax

```
+(NSString *) getErrorMessageFromJSON: (NSDictionary *)
   jsonResponse;
```

Description

This method returns an error message from the JSON response.

2.6.8 Method getWLErrorCodeFromJSON

Syntax


```
+(WLErrorCode) getWLErrorCodeFromJSON: (NSDictionary *)
   jsonResponse;
```

Description

This method returns the `WLErrorCode` from the JSON response.

2.7 Class WLProcedureInvocationResult

This class contains the result of calling a back-end service, including statuses and data items that are retrieved by the adapter function from the server.

2.7.1 Method isSuccessful

Syntax

```
-(BOOL)isSuccessful;
```

Description

This method returns `YES` if the call was successful, and `NO` otherwise.

2.7.2 Method procedureInvocationErrors

Syntax

```
-(NSArray *) procedureInvocationErrors;
```

Description

This method returns an `NSArray` of applicative error messages that are collected by the server during the procedure call.

2.7.3 Property response

Parameters

Name	Type	Description
response	NSDictionary	This property is an `NSDictionary`, which represents the JSON response that is returned by the Worklight Server.

Table 2-25: Property response parameters

2.8 Class WLCookieExtractor

This class is used to share cookies between the web code and native code. 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.

This class has no API methods. An object of this class must be passed as a parameter to `wlConnectWithDelegate:cookieExtractor.`

2.8.1 Constructor

Syntax

```
public WLCookieExtractor(Application app)
```

Parameters

Type	Name	Description
`Application`	`app`	An Android application instance.

Table 2-26: Class constructor WLCookieExtractor parameters

2.9 Class ChallengeHandler

This base class is used to create custom Challenge Handler. Application developers must extend this class. This class is used to create custom user authentication.

2.9.1 Method isCustomResponse

Syntax

```
-(BOOL)_isCustomResponse:(WLResponse *)response;
```

Description

This method must be overridden by the extended class of `ChallengeHandler`. In most cases, this method is called to test whether there is a challenge to be answered to. Some responses are handled by default Challenge Handlers. If this method returns `YES` then the Worklight SDK calls the handle challenge method.

Parameters

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

Table 2-27: Method isCustomResponse parameters

2.9.2 Method handleChallenge

Syntax

```
-(void) handleChallenge: (WLResponse *)response;
```

Description

This method is called by the Worklight SDK whenever `isCustomResponse` returns a `YES` value and must be implemented by the application developer. This method must handle the challenge, for example to show the login screen.

Parameters

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

Table 2-28: Method handleChallenge parameters

2.9.3 Method submitSuccess

Syntax

```
-(void) submitSuccess:(WLResponse *) response;
```

Description

This method is called by the application developer when the challenge is answered successfully, for example after the user submits the login form successfully. Then, this method sends the original request.

Parameters

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

Table 2-29: Method submitSuccess parameters

2.9.4 Method submitFailure

Syntax

```
-(void) submitFailure:(WLResponse *) response;
```

Description

This method is called by the application developer whenever the challenge is answered by an error. This method is inherited from `BaseChallengeHandler`.

Parameters

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

Table 2-30: Method submitFailure parameters

2.9.5 Method submitLoginForm

Syntax

```
-(void) submitLoginForm:
     (NSString *)requestUrl
     requestParameters:(NSDictionary *) parameters
     requestHeaders:(NSDictionary *) headers
     requestTimeoutInMilliSeconds:(int) timeout
     requestMethod:(NSString *) method;
```

Description

This method is used to send collected credentials to a specific URL. Developers can also specify request parameters, headers, and timeout.

The success/failure delegate for this request is the instance itself, which is why you must override the `onSuccess` / `onFailure` methods.

Parameters

Name	Type	Description
requestUrl	`NSString`	Absolute URL if the user sends an absolute url that starts with `http://` or `https://` Otherwise, URL relative to the Worklight Server
parameters	`NSDictionary`	The request parameters
headers	`NSDictionary`	The request headers
timeout	`int`	To supply custom timeout, use a number over 0. If the number is under 0, the Worklight SDK uses the default timeout which is 10,000 milliseconds.

Name	Type	Description
method	NSString	The HTTP method to use Acceptable values: GET, POST The default is POST.

Table 2-31: Method submitLoginForm parameters

2.9.6 Method submitAdapterAuthentication

Syntax

```
-(void) submitAdapterAuthentication: (WLProcedureInvocationData *)
   wlInvocationData: options:(NSDictionary *)requestOptions;
```

Description

This method is used to invoke a procedure from the Challenge Handler.

Parameters

Name	Type	Description
wlInvocationData	WLProcedureInvocationData	The invocation data such as the procedure name or the method name.
requestOptions	NSDictionary	A map with the following keys and values: timeout – NSNumber. Time in milliseconds for this invokeProcedure to wait before the request fails with WLErrorCodeRequestTimeout. The default timeout is 10,000 milliseconds. To disable the timeout, set this parameter to 0. invocationContext – an object that is returned with WLResponse to the delegate methods. You can use this object to distinguish different invokeProcedure calls.

Table 2-32: Method submitAdapterAuthentication parameters

2.9.7 Method onSuccess

Syntax


```
-(void) onSuccess:(WLResponse *)response;
```

Description

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

Parameters

Name	Type	Description
response	`WLResponse`	The received response.

Table 2-33: Method onSuccess parameters

2.9.8 Method onFailure

Syntax

```
-(BOOL) isCustomResponse:( (WLFailResponse *)response;
```

Description

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

Parameters

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

Table 2-34: Method isCustomResponse parameters

2.10 Enum WLErrorCode

Definition

```
typedef NSUInteger WLErrorCode;
    enum {
      WLErrorCodeUnexpectedError,
      WLErrorCodeUnresponsiveHost,
      WLErrorCodeRequestTimeout,
      WLErrorCodeProcedureError,
      WLErrorCodeApplicationVersionDenied};
```

Description

The following list shows the various error codes that you can find and their description:

- UNEXPECTED_ERROR: unexpected error
- REQUEST_TIMEOUT: request timed out
- UNRESPONSIVE_HOST: service currently unavailable
- PROCEDURE_ERROR: procedure invocation error
- APP_VERSION_ACCESS_DENIAL: application version denied

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

