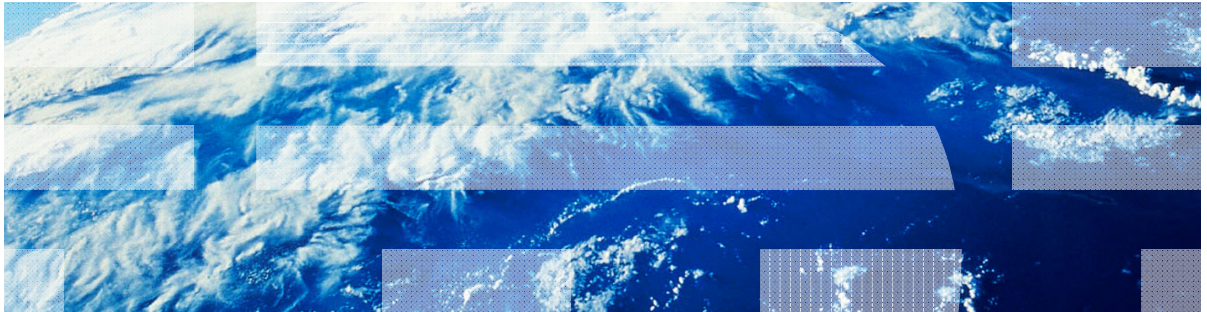*IBM Worklight V5.0.5*
Getting Started

Module 8.2 – iOS Development Using Worklight API in a Native Application

Trademarks

- IBM, the IBM logo, and 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.

- This document may not be reproduced in whole or in part without the prior written permission of IBM.

About IBM®

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

Agenda

- Creating a Worklight native API

- Create and configure an iOS native application

- Initializing the WLCLient

- Invoking a Worklight procedure

- Receiving a procedure response

Creating a Worklight native API

- IBM Worklight® provides ability for native iOS applications to communicate with a Worklight Server by using the IBM Worklight native API library.

- To serve a native iOS application, the Worklight server must be aware of it.

- The Worklight native API is located under the apps folder of your Worklight project.

- The Worklight native API folder serves two purposes:
  - It contains a native API library and configuration file that must be copied to your native iOS project.
  - It contains the **application-descriptor.xml** file, which should be deployed to a Worklight server as an entry point, similar to a Worklight application.

- In this module, you learn how to create a Worklight native API and how to use its components in your native iOS application.

Creating a Worklight native API

- Worklight native API contains several components:
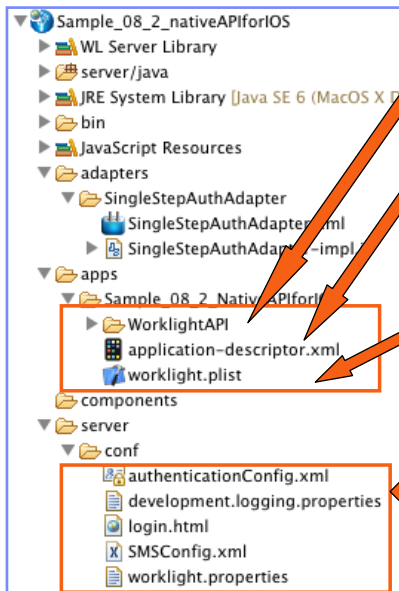
The **WorklightAPI** folder is a Worklight API library that should be copied to your native iOS project.

The **application-descriptor.xml** file is used to define application metadata and to configure security settings to be enforced by a Worklight server.

The **Worklight.plist** file contains connectivity settings to be used by a native iOS application. This file should be copied to your native iOS project.

Like with any Worklight project, you create the server configuration by modifying files under the **server\conf** folder.

Creating a Worklight native API

1. In Worklight Studio, create a Worklight project, and add a Worklight Native API.

2. In the New Worklight Native API dialog, enter your application name, and select **iOS** for the **Environment** field.

3. Right-click the Worklight native API folder and select **Run As ➔ Deploy Native API**.

Creating a Worklight native API

- Edit the *worklight.plist* file that holds the server configuration:
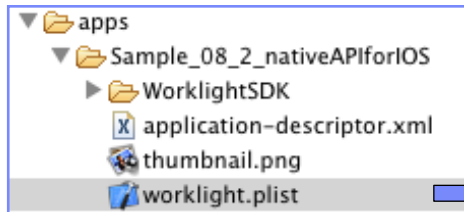
Key	Type	Value
protocol	String	http
host	String	9.148.23.13
port	String	8080
wlServerContext	String	/
application id	String	Sample_08_2_nativeAPIforIOS
application version	String	1.0
environment	String	iosnative

- *protocol* – The communication protocol to the Worklight server can be either *http* or *https.*

- *host* – The hostname of the Worklight server.

- *port* – The port of the Worklight server.

- **wlServerContext** – The context root path of the application on the Worklight server.

- *application id* – The application ID as defined in the *application-descriptor.xml* file.

- *application version* – The application version.

- **environment** – The target environment of the native application (Android or iOS).

Agenda

- Creating a Worklight native API

- Create and configure an iOS native application

- Initializing the WLCLient

- Invoking a Worklight procedure

- Receiving a procedure response

Create and configure an iOS native application

- Create an Xcode project or use an existing one.

- Copy the WorklightAPI folder and the worklight.plist file from the Eclipse Worklight native API to the root of your native project.

- Link the following libraries in your native iOS application: CFNetwork, SystemConfiguration, MobileCoreServices, CoreData, Security, libz.dylib.
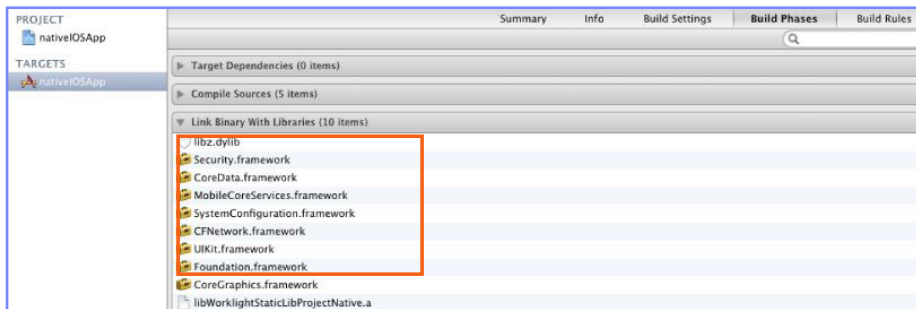
Agenda

- Creating a Worklight native API

- Create and configure an iOS native application

- Initializing the WLCLient

- Invoking a Worklight procedure

- Receiving a procedure response

Initializing the WLClient

- Access the WLClient functionality by using [WLClient sharedInstance] anywhere in your application.

- Initiate the connection to the server by using wlConnectWithDelegate method

- For most actions, you need to specify a delegate object, such as a MyConnectListener instance in the following example:

```
MyConnectListener *connectListener = [[MyConnectListener alloc] initWithController:self];
[[WLClient sharedInstance] wlConnectWithDelegate:connectListener];
[connectListener release];
```

- You learn how to create it in subsequent slides.

- Remember to import **WLClient.h** and **WLDelegate.h** in your header file.

Initializing the WLCLient

- As described on a previous slide, you must to supply a connection delegate (listener) to Worklight server invocation methods.

- Create a delegate to be used in the wlConnectWithDelegate method and receive the response from the Worklight server. Name the class *MyConnectListener*.

- The header file must specify that it implements the *WLDelegate* protocol.

```objc
#import <Foundation/Foundation.h>
#import "WLClient.h"
#import "WLDelegate.h"
#import "ViewController.h"

@interface MyConnectListener : NSObject <WLDelegate> {
    @private
    ViewController *vc;
}

- (id)initWithController: (ViewController *)mainView;
@end
```

- The *WLDelegate* protocol specifies that the class implements the following methods:

  – *The onSuccess (WLResponse *)response*

  – *The onFailure (WLFailResponse *)response*

Initializing the WLCLient

- After wlConnectWithDelegate finishes, the onSuccess method, or the onFailure method, of the supplied MyConnectListener instance is invoked.

- In both cases, the response object is sent as an argument.

- Use this object to operate data that are retrieved from server.

```objc
-(void)onSuccess:(WLResponse *)response{
    NSLog(@"\nConnection Success: %@", response);
    NSString *resultText = @"Connection success. ";

    if ([response responseText] != nil){
        resultText = [resultText stringByAppendingString:[response responseText]];
    }
    [vc updateView:resultText];
}

-(void)onFailure:(WLFailResponse *)response{
    NSString *resultText = @"Connection failure. ";

    if ([response responseText] != nil){
        resultText = [resultText stringByAppendingString:[response responseText]];
    }
    [vc updateView:resultText];
}
```
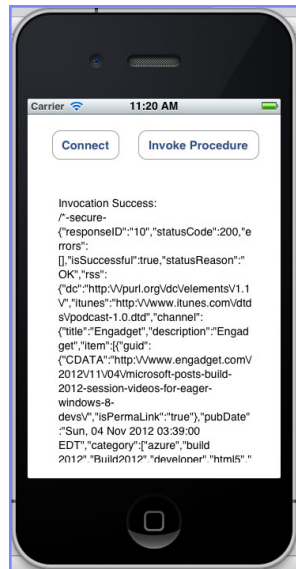
Agenda

- Creating a Worklight native API

- Create and configure an iOS native application

- Initializing the WLCLient

- Invoking a Worklight procedure

- Receiving a procedure response

Invoking a Worklight procedure

- To invoke a procedure, use the following steps:
  - Create a *WLProcedureInvocationData* object and specify the adapter name and the procedure name.
  - Invoke the procedure by using the shared instance of the *WLClient*.

```
WLProcedureInvocationData *myInvocationData = [[WLProcedureInvocationData alloc]
                            initWithAdapterName:@"RSSReader" procedureName:@"getFeeds"];
MyInvokeListener *invokeListener = [[MyInvokeListener alloc] initWithController: self];
[[WLClient sharedInstance] invokeProcedure:myInvocationData withDelegate:invokeListener];
[invokeListener release];
```

- As previously mentioned, you need to supply a delegate object to manage the retrieved data.
  - See subsequent slides to learn how to create this object.

Agenda

- Creating a Worklight native API

- Create and configure an iOS native application

- Initializing the WLCLient

- Invoking a Worklight procedure

- Receiving a procedure response

Receiving a procedure response

- When the procedure invocation is complete, a delegate method of MyInvokeListener class instance is called.

- Remember: any delegate header file must specify that it complies with a *WLDelegate* protocol.

```
#import <Foundation/Foundation.h>
#import "WLClient.h"
#import "WLDelegate.h"
#import "ViewController.h"

@interface MyInvokeListener : NSObject <WLDelegate> {
```

Receiving a procedure response

- After the procedure invocation finishes, the onSuccess method or the onFailure method of the supplied MyInvokeListener instance is called.

- In both cases, a response object is sent as an argument.

- Use this object to operate data that are retrieved from server

```objc
-(void)onSuccess:(WLResponse *)response {
    NSLog(@"Invocation Success: %@", response);
    NSString *resultText = @"Invocation success. ";

    if ([response responseText] != nil){
        resultText = [resultText stringByAppendingString:[response responseText]];
    }
    [vc updateView:resultText];
}

-(void)onFailure:(WLFailResponse *)response{
    NSLog(@"Invocation Failure: %@", response);
    NSString *resultText = @"Invocation failure. ";

    if ([response responseText] != nil){
        resultText = [resultText stringByAppendingString:[response responseText]];
    }
    [vc updateView:resultText];
}
```

Receiving a procedure response

- The sample for this training module can be found in the Getting Started page of the IBM Worklight documentation website at http://www.ibm.com/mobile-docs.

- Sample contains two projects:
  - The **module_08_2_NativeAPIForIOS.zip** contains a Worklight native API to be deployed to your Worklight server.
  - The **module_08_2_iOSNativeApp.zip** contains native iOS application that uses a Worklight native API library to communicate with the Worklight server.

- Make sure to update the **wlclient.plist** file in iOSNativeApp with relevant server settings.

Notices

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

Thank You