IBM Worklight V5.0.6

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Contents

Experimental IBM Worklight offline documentation. In case of issues, this document is provided "as is". refer to the online user documentation.

Experimental IBM Worklight offline user documentation.
This document is provided "as is".
In case of issues, refer to the online user documentation.

Chapter 1. Starting with IBM Worklight

With IBM® Worklight®, you can simplify and accelerate the development, testing, and delivery of your mobile apps.

IBM Worklight offers an Eclipse-based visual development and a server environment for you to create native, hybrid, and standard web mobile applications, and maximizes code reuse across different mobile phone platforms.

Introducing IBM Worklight

IBM Worklight consists of different components: Worklight Studio, Worklight Server, Worklight device runtime components, Worklight Console, Application Center, and IBM Mobile Application Platform Pattern.

Editions of IBM Worklight

- IBM Worklight Developer Edition is an unsupported and non-warranted program that consists of a single plug-in for the Eclipse integrated development environment (IDE). It is available from the developerWorks® website. You must install it with P2 Eclipse update. It provides every Worklight Studio function available in the Consumer and Enterprise Editions, except for some security-related features.
- IBM Worklight Consumer Edition and IBM Worklight Enterprise Edition are identical binaries that differ in license only. These programs are supported through an IBM International License Agreement. You can install Worklight Studio for these editions with P2 Eclipse update or with IBM Installation Manager. The IBM Worklight Consumer Edition and the IBM Worklight Enterprise Edition also contain a separate Worklight Server component, which is available with IBM Passport Advantage® as an Installation Manager package.

Worklight Studio

In a mobile development platform, cross-platform portability of the application code is critical for mobile device application development. Various methods exist to achieve this portability. With IBM Worklight, you can develop multiplatform applications by using Worklight Studio, which is a mobile development studio, to address the requirements of the organization.

You can use Worklight Studio for the following tasks:

- Develop rich HTML5, hybrid and native applications for all supporting modern devices by using native code, a bidirectional WYSIWYG, and standard web technologies and tools.
- Maximize code sharing by defining custom behavior and styling guidelines that match the target environment.
- Access device APIs by using native code or standard web languages over a uniform Apache Cordova bridge. **Apache Cordova is preinstalled with Worklight, therefore do not download your own Apache Cordova version.**
- Use both native and standard web languages within the same application to balance development efficiency and a rich user experience.
- Use third-party tools, libraries, and frameworks such as JQuery Mobile, Sencha Touch, and Dojo Mobile.

Experimental IBM Worklight user documentation.
This document is provided as is.
In case of issues refer to the main documentation.

- Implement runtime skins to build apps that automatically adjust to environment guidelines such as form factor, screen density, HTML support, and UI input method.

Worklight Server

The Worklight Server is a runtime container for the mobile applications you develop in Worklight Studio. It is not an application server in the Java™ Platform, Enterprise Edition (JEE) sense. It acts as a container for Worklight application packages, and is in fact a collection of web applications (optionally packaged as an EAR file) that run on top of traditional application servers.

Worklight Server is designed to integrate into the enterprise environment and use its existing resources and infrastructure. This integration is based on adapters that are server-side software components responsible for channeling back-end enterprise systems and cloud-based services to the user device. You can use adapters to retrieve and update data from information sources, and to allow users to perform transactions and start other services and applications.

You can use Worklight Server for the following tasks:

- Empower hundreds of thousands of users with transactional capabilities and enable their direct access to back-end systems and cloud-based services.
- Configure, test, and deploy descriptive XML files to connect to various back-end systems by using standard Worklight Studio tools.
- Directly update deployed hybrid and web applications, without going through the different app stores (subject to the terms of service of the vendor).
- Automatically convert hierarchical data to JSON format for optimal delivery and consumption.
- Enhance users interaction with a uniform push notification architecture.
- Define complex mashups of multiple data sources to reduce overall traffic.
- Integrate with the existing security and authentication mechanisms of the organization.

Worklight device runtime components

IBM Worklight provides client-side runtime code that embeds server functionality within the target environment of deployed apps. These runtime client APIs are libraries that are integrated into the locally stored app code. They complement the Worklight Server by defining a predefined interface for apps to access native device functions. Among these APIs, IBM Worklight uses the Apache Cordova development framework. This framework delivers a uniform bridge between standard web technologies (HTML5, CSS3, JavaScript) and the native functions that different mobile platforms provide.

The Worklight device runtime components provide the following functions:

- Mobile data integration: connectivity and authentication APIs
- Security features: on-device encryption, offline authentication, and remote disablement of apps
- Cross-platform support: runtime skins, UI abstractions, and HTML5 toolkits compatibility
- Mobile client functionality: hybrid app framework, access to device APIs and push notification registration
- Reports and analytics: built-in reports and event-based custom reporting

Experimental IBM Worklight user documentation
In case of doubt, refer to the online user documentation.
This document is provided as-is.

- Resource serving: direct update of app web resources and HTML5 caching

Worklight Console

The Worklight Console is used for the control and management of the mobile organization, from managing deployed applications to collecting and analyzing user statistics.

You can use the Worklight Console for the following tasks:
- Monitor all deployed applications, adapters, and push notification rules from a centralized, web-based console.
- Assign device-specific identifiers (IDs) to ensure secure application provisioning.
- Remotely disable applications by using preconfigured rules of app version and device type.
- Customize messages that are sent to users on application launch.
- Collect user statistics from all running applications.
- Generate built-in, pre-configured user adoption and usage reports.
- Configure data collection rules for application-specific events.
- Export raw reporting data to be analyzed by the BI systems of the organization.

Application Center

With the Application Center, you can share mobile applications that are under development within your organization in a single repository of mobile applications. Development team members can use the Application Center to share applications with members of the team. This process facilitates collaboration between all the people who are involved in the development of an application.

Your company can typically use the Application Center as follows:
1. The development team creates a version of an application.
2. The development team uploads the application to the Application Center, enters its description, and asks the extended team to review and test it.
3. When the new version of the application is available, a tester runs the Application Center installer application, which is the mobile client. Then, the tester locates this new version of the application, installs it on their mobile device, and tests it.
4. After the tests, the tester rates the application and submits feedback, which is visible to the developer from the Application Center console.

The Application Center is aimed for private usage within a company, and you can target some mobile applications to specific groups of users. You can use the Application Center as an enterprise application store.

With the Application Center, you can manage native or hybrid applications that are installed on mobile devices. The Application Center supports applications that are built for the Google Android platform and the Apple iOS platform, but does not target mobile web applications.

IBM Mobile Application Platform Pattern

With the IBM Mobile Application Platform Pattern, you can deploy the Worklight Server on IBM PureApplication™ System. With this pattern, administrators and businesses can respond quickly to changes in the business by taking advantage of

Experimental IBM Worklight offline user documentation.
This document is provided "as is".
In case of issues refer to the online user documentation.

on-premises Cloud technologies. This approach simplifies the deployment process, and improves the operational efficiency to cope with increased mobile demand. The demand accelerates iteration of solutions that exceed traditional demand cycles. Deploying the IBM Mobile Application Platform Pattern on IBM PureApplication System also gives access to best practices and built-in expertise, such as built-in scaling policies.

System requirements for using IBM Worklight

Operating systems, Eclipse versions, and SDKs supported by IBM Worklight.

Read the *IBM Worklight and IBM Mobile Foundation detailed system requirements* web page at http://www.ibm.com/support/docview.wss?uid=swg27024838 to identify the system requirements for this release of IBM Worklight, including:

- The operating systems that support IBM Worklight, including mobile device operating systems
- The editions of Eclipse that support Worklight Studio, which is an Eclipse-based integrated development environment (IDE)
- The supported software development kits (SDKs)
- The supported web browsers
- The needed hardware configuration

What's new

This section details the new features and changes in IBM Worklight V5.0.6 and subsequent fix packs.

What's new in IBM Worklight V5.0.6.2

IBM Worklight V5.0.6.2 fixes many problems that were identified in previous versions.

Client-side log collection

Starting with IBM Worklight V5.0.6.2, you can now capture and receive uploaded client-side logs. For more information, see "Client-side log capture" on page 158.

Support for Android 4.4

You can now use IBM Worklight to develop applications that run on Android 4.4. For more information, see IBM Worklight Supports Android 4.4.

Application startup time improvement

Significant improvements have been made on the amount of time the user must wait when IBM Worklight applications are started for the first time. The performance improvement applies to both Android and iOS. In cases where the web resources are not encrypted (that is, when `encryptWebResources` is set to the default value of `false`), a change has been made resulting in a significant improvement in the application startup time.

- Maximum improvement is achieved with `testWebResourcesChecksum` and `encryptWebResources` set to their default value of `false` in the application descriptor file.
- Setting the `testWebResourcesChecksum` value to `true` reduces the improvement slightly.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues refer to the online user documentation.

- Setting the `encryptWebResources` value to `true` results in no improvement.

Changes to the WL.Client.init JavaScript client-side API method

The "WL.Client.init" on page 189 method supports the following new properties and parameters:

- New `showCloseOnDirectUpdateFailure` property.
- New `showCloseOnRemoteDisableDenial` property.
- New `message` and `downloadLink` parameters for the `onErrorRemoteDisableDenial` property.

Change in Remote Disable behavior

When you use the Worklight Console to disable an application's access to the server, the default behavior is no longer to exit the application completely. For more information, see "Remotely disabling application connectivity" on page 363.

Deprecation of WL.OptionsMenu with Android 3.0, API level 11

If your application targets Android 3.0 (API level 11) or higher, `WL.OptionsMenu` might have no effect, depending on the device. For more information, see Creating an Options Menu in the *Android Developers API Guides* and "Options Menu and Application Bar API" on page 251.

Documentation improvements

- Clarification on how to use the `WL.App.BackgroundHandler.hideView` handler to hide the application splash screen. See "WL.App.BackgroundHandler.setOnAppEnteringBackground" on page 175.

List of fixes for IBM Worklight V5.0.6.2

For a complete list of issues that are fixed in IBM Worklight V5.0.6.2, see Fix Pack 5.0.6.2.

What's new in IBM Worklight V5.0.6.1

IBM Worklight V5.0.6.1 fixes many problems that were identified in previous versions.

For more information about the fixed issues, see Fix Pack 5.0.6.1.

What's new in IBM Worklight V5.0.6

This section details the new features and changes in IBM Worklight V5.0.6 compared to the previous version of this product.

Simplified deployment and operational experience

IBM Worklight V5.0.6 defines the Worklight Virtual Application Pattern for IBM PureApplication System. Use this pattern to simply the deployment and the operational experience of IBM Worklight apps.

New Worklight Virtual Application Pattern for PureApplication System

IBM Worklight V5.0.6 defines the Worklight Virtual Application Pattern (VAP) for PureApplication System that you can use to import, configure, deploy, and manage IBM Worklight applications and adaptors directly.

Experimental IBM Worklight online user documentation
This document is provided 'as is'.
In case of issues, refer to the online user documentation.

For more information, see Chapter 8, “Deploying to the cloud by using IBM PureApplication System,” on page 499.

To get started with this new feature, see the new modules *Introducing Worklight Server and Application Center on IBM PureApplication System* and *Integrating Tivoli Directory Server on IBM PureApplication System*, under category 11, *Integrating with other products*, in “Getting started tutorials and samples” on page 29.

Integration of Application Center into Virtual Application Pattern

In IBM Worklight V5.0.6, you can configure and connect the operational components of the Application Center to deploy the enterprise application on PureApplication System.

For more information, see “Deployment of the Application Center on IBM PureApplication System” on page 508.

Enhancement of Worklight Studio for Virtual Application Pattern

In IBM Worklight V5.0.6, you can now deploy your apps to the Worklight Server directly from Worklight Studio.

For more information, see “Working with IBM Worklight PureApplication System Extension for Worklight Studio” on page 504.

Improved security and user experience

IBM Worklight V5.0.6 provides enhanced capabilities in terms of security and user experience.

Important: Some of the changes that are listed in this topic have an impact on how your IBM Worklight applications behave. In some cases, you might have to manually modify your projects or applications that you created with previous versions of IBM Worklight. For more information about this potential impact, and any required manual migration, see “Migrating to a newer version of IBM Worklight” on page 22.

Separate SSL certificate per HTTP adapter

In IBM Worklight V5.0.6, you can configure each HTTP adapter that is based on SSL so that your application can communicate with different back-end services.

For more information, see “SSL certificate keystore setup” on page 409.

Note: With IBM Worklight V5.0.6, the Worklight Server now recognizes self-signed certificates automatically. In previous versions of IBM Worklight, you first had to import them to a Java Runtime Environment (JRE) keystore, by following instructions such as the ones here. With IBM Worklight V5.0.6, you no longer have to follow such instructions, and the corresponding topic is no longer part of the IBM Worklight V5.0.6 Information Center.

Device Single Sign-On (SSO)

In IBM Worklight V5.0.6, the mobile user authentication mechanism now uses Single Sign-On (SSO). With Device SSO, the mobile user authenticates only once for all applications from the same vendor. After the user successfully authenticates

Experimental IBM Worklight online user documentation.
This document is provided "as-is".
In case of issues, refer to the online user documentation.

with an SSO login module, the user gains access to the protected resources that are using the same SSO login module, without having to authenticate again for each of them.

The user remains authenticated while requests to access the resources that are protected by this SSO login module are issued. After an idle period, or after an explicit logout from the SSO login module, the user is no longer authenticated, and must log in to the SSO login module to reobtain access to these resources.

For more information, see "Device single sign-on (SSO)" on page 154.

Enhanced JSONStore capability

In IBM Worklight V5.0.6, JSONStore capability includes the following enhancements:

- Support for multiple users and multiple passwords to provide secure (private) data for each user in a shared app on a device environment
- Simplification of the JavaScript API programming model by using promises
- Better management of concurrency conditions when multiple collections are being used
- Better exception handling
- Support for JSONStore in x86 Android environments

For more information about JSONStore, see "WL.JSONStore" on page 214.

To get started with this feature, see the modules *JSONStore - The client-side JSON-based database overview*, *JSONStore - API basics*, *JSONStore - Synchronizing client and server databases*, *JSONStore - Encrypting sensitive data*, and *JSONStore - Encrypting sensitive data with FIPS 140-2*, under category 5, *Advanced client side development*, in "Getting started tutorials and samples" on page 29.

Enhanced client-side notification subscription and management

In IBM Worklight V5.0.6, support for notification mechanisms is enhanced.

- Push notifications are now supported also in **native** applications for iOS and Android (while they were supported only in hybrid applications for iOS and Android in previous versions).
- Push notifications are now supported also on Windows Phone 8 devices.
  - IBM Worklight V5.0.6 provides API support for Windows Phone 8 push notifications.
  - With this extended API, you can send push notifications to your Windows Phone 8 users through integration of IBM Worklight with the Microsoft Push Notification Service (MPNS).
  - You can send Windows Phone 8 push notifications only through unauthenticated web services. Some limitations therefore apply. For more information about these limitations, see the Microsoft Windows Phone Dev Center website at http://dev.windowsphone.com/en-us/develop, and search for `Sending push notifications for Windows Phone`.
- You can now make HTTP requests to the subscribe SMS servlet to subscribe or unsubscribe to SMS notifications (which are supported on devices that support SMS functions).
- For more information about the IBM Worklight unified push notification mechanism, see "Push Notification" on page 395.

Experimental IBM Worklight offline user documentation.
This document is provided as is.
In case of issues refer to the online user documentation.

- For more information about the push notification API from the client side, see "Mobile push notification methods" on page 243.
- To get started with push notifications in native apps, see the modules *Using Worklight API for push notifications in native iOS applications* and *Using Worklight API for push notifications in native Android applications*, under category 7, *Developing native applications with Worklight*, in "Getting started tutorials and samples" on page 29.
- For more information about the push notification API from the server side, see "JavaScript server-side API" on page 271.
- For more information about SMS notification configuration, see "Subscribe SMS servlet" on page 398.
- To get started with SMS notifications, see the module *SMS notifications*, under category 9, *Advanced topics*, in "Getting started tutorials and samples" on page 29.

Improvements to the Application Center

In IBM Worklight V5.0.6, the Application Center includes the following improvements:

- Support for BlackBerry 6 and BlackBerry 7

  You can now upload BlackBerry `*.jad` and `*.cod` files to the Application Center.
- New BlackBerry client application
- Ability to install the BlackBerry client application over the air
- Improved security and authentication in the Application Center console:
  - Administrators can now log in to the Application Center console by using a login page, which is similar to the login page of the IBM WebSphere® Application Server console.
  - Administrators can now log out of the Application Center console.
- Ability to dump `*.ipa`, `*.apk` and BlackBerry application files to the disk from the Application Center console
- Ability to configure the Service Endpoint to enhance security:
  - You can enhance the security by configuring a secured proxy in front of the Application Center. Requests to the Application Center console and services are run through this proxy first, and then directed to the Application Center.
  - You can do this proxy configuration through the configuration of the Service Endpoint.

In IBM Worklight V5.0.6, the Application Center console includes the following changes:

- The Application Center console was divided into two separate applications:
  - The *Application Center console* application is dedicated to user web interaction. Its URL is: http://<*hostname*>:<*portnumber*>/appcenterconsole
  - One *Application Center services* application is dedicated to responding to the necessary console services. Its URL is: http://<*hostname*>:<*portnumber*>/applicationcenter
- To connect to the back-end server, you must now enter the Application Center services URL in the start (settings) screen of the Application Center mobile client.

For more information about these improvements and changes, see Chapter 7, "Application Center," on page 423.

Experimental IBM Worklight online user documentation.
This content is provided "as is".
In case of issues, refer to the online user documentation.

Improvements to the Worklight Console

In IBM Worklight V5.0.6, the Worklight Console allows device notification messages in multiple languages, not only in English.

For more information, see "Defining administrator messages from the IBM Worklight Console in multiple languages" on page 366.

Changes in the Mobile Browser Simulator

In IBM Worklight V5.0.6, the Windows Phone 8 silhouette and device profiles for the Mobile Browser Simulator are now available.

Enhanced IBM Worklight API

IBM Worklight V5.0.6 enhances and extends the API that you can use to develop mobile applications, in particular provides new API to support push notifications in native apps.

Important: Some of the changes that are listed in this topic have an impact on how your IBM Worklight applications behave. In some cases, you might have to manually modify your projects or applications that you created with previous versions of IBM Worklight. For more information about this potential impact, and any required manual migration, see "Migrating to a newer version of IBM Worklight" on page 22.

Modified way to add custom code to an Android app

In IBM Worklight V5.0.6, adding custom code to your Android app in the onCreate method is deprecated. You now add custom code to your Android app in the onWLInitCompleted method instead.

For more about information about adding custom code to an Android app, see "Adding custom code to an Android app" on page 78.

Updated version of external libraries

In IBM Worklight V5.0.6, the following libraries, which are used as part of the product, have upgraded versions:
- Cordova library is upgraded to version 2.3.
- Dojo library is upgraded to version 1.8.3.
- jQuery library is upgraded to version 1.8.1.

If you are using other JavaScript libraries such as jQuery Mobile, you might have to upgrade them as well.

To get started with using Cordova in IBM Worklight, see the module *Apache Cordova overview*, under category 6, *Adding native functionality to hybrid applications with Apache Cordova*, in "Getting started tutorials and samples" on page 29.

Updated JavaScript client-side API

IBM Worklight V5.0.6 updated its JavaScript client-side API. For more information, see "JavaScript client-side API" on page 173.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

In particular, the WL.OptionsMenu.isEnabled and WL.OptionsMenu.isVisible methods now take a callback function as a parameter. The callback function is called by Cordova after the request is processed, and it receives the current enabled or visible state.

Updated JavaScript server-side API

IBM Worklight V5.0.6 complements its JavaScript server-side API with further elements that you can use to extend the Worklight Server.

In particular, this API now includes the WL.Server.createDefaultNotification method that you can use to create a notification JSON block for the supplied values, for all supported environments.

In IBM Worklight V5.0.5, using query parameters on the URL (for example,`http://some.server/action?someparam=somevalue`) with the WL.Server.invokeHttp method was impossible for HTTP methods other than GET. In IBM Worklight V5.0.6, using query parameters is now possible for other HTTP methods, with some restrictions. For more information about this WL.Server.invokeHttp method, see "Method WL.Server.invokeHttp" on page 272.

For more information about the IBM Worklight JavaScript server-side API, see "JavaScript server-side API" on page 271.

Updated Objective-C client-side API to support push notifications in native apps on iOS

IBM Worklight V5.0.6 includes some changes to its Objective-C client-side API to develop native apps on iOS.

In particular, this API now includes elements that you can use to manage push notifications in native apps on iOS, such as the WLPush class and associated interfaces to subscribe and unsubscribe to push notifications.

For more information, see "Objective-C client-side API for native iOS apps" on page 271.

To get started with push notifications in native apps on iOS, see the module *Using Worklight API for push notifications in native iOS applications*, under category 7, *Developing native applications with Worklight*, in "Getting started tutorials and samples" on page 29.

Updated Java client-side API to support push notifications in native apps on Android

IBM Worklight V5.0.6 includes some changes to its Java client-side API to develop native apps on Android.

In particular, this API now includes elements that you can use to manage push notifications in native apps on Android, such as the class WLPush and associated interfaces to subscribe and unsubscribe to push notifications.

For more information, see "Java client-side API for native Android apps" on page 271.

Experimental IBM Worklight offline user documentation. This document is provided as-is. In case of issues refer to the online user documentation.

To get started with push notifications in native apps on Android, see the module *Using Worklight API for push notifications in native Android applications*, under category 7, *Developing native applications with Worklight*, in “Getting started tutorials and samples” on page 29.

Updated Java client-side API to develop Java Platform, Micro Edition (Java ME) apps

IBM Worklight V5.0.6 includes some changes to its Java client-side API to develop Java ME apps.

In particular, note the following changes:
- The two WL.createInstance methods, with or without a `String` argument, are deprecated. These methods are now replaced with two WL.createInstance methods, with or without a `String` argument, and with a supplementary and mandatory `MIDlet midlet` argument.
- The WLClient.setHeartBeatInterval method is now defined. You use it to set the period of the heartbeat signal that is sent to the Worklight Server to ensure that the session with the server is kept alive when the app does not issue any call to the server.

For more information, see “Java client-side API for Java ME apps” on page 271.

Updated Java server-side API

In IBM Worklight V5.0.6 Java server-side API, the WorkLightLoginModule is now deprecated, and replaced with the WorkLightAuthLoginModule interface.

For more information, see “Java server-side API” on page 301 and “Interface WorkLightAuthLoginModule” on page 305

Integration with complementary products

IBM Worklight V5.0.6 integrates with complementary offers, such as Tealeaf® CX and SiteMinder.

Integrating Tealeaf CX with IBM Worklight

In IBM Worklight V5.0.6, you can integrate IBM Worklight with Tealeaf CX. Tealeaf CX gives visibility, insight, and answers for companies that do business online.

For more information about this integratin , see “Integrate Tealeaf CX with IBM Worklight” on page 63.

Integrating SiteMinder with IBM Worklight

In IBM Worklight V5.0.6, you can integrate IBM Worklight with SiteMinder. SiteMinder is a system that enables user authentication.

For more information about how you can integrate SiteMinder with IBM Worklight to create SiteMinder-based authentication, see the module *Integrating with SiteMinder*, under category 11, *Integrating with other products*, in “Getting started tutorials and samples” on page 29.

Miscellaneous modifications

Additional changes in IBM Worklight V5.0.6 include changes that are related to federal requirements and changes in the available documentation.

Experimental IBM Worklight offline user documentation. This document is provided "as-is". In case of issues, refer to the online user documentation.

Support of federal requirements

In IBM Worklight V5.0.6, the following federal requirements are now supported:

- IPv6.
- Federal Desktop Core Configuration (FDCC) and United States Government Configuration Baseline (USGCB). For more information about FDCC, see “FDCC and USGCB support” on page 372.
- Federal Information Processing Standards (FIPS) 140-2. IBM Worklight now offers the option to use a FIPS 140-2 validated cryptographic module for the protection (encryption) of data stored locally in the JSONStore feature. For more information about FIPS 140-2, see “FIPS 140-2 support” on page 372. To get started with protecting data in JSONStore with FIPS 140-2, see the module *JSONStore - Encrypting sensitive data with FIPS 140-2*, under category 5, *Advanced client side development*, in “Getting started tutorials and samples” on page 29.

Augmented getting started tutorials and samples

The list of tutorials and samples to get started with IBM Worklight is augmented and enhanced.

- The categories are reorganized to help you find more rapidly the tutorial that you require.
- All tutorial modules and samples are updated to match the modified or new features of IBM Worklight V5.0.6.
- New tutorials and samples are available for you to learn how to get started with the new features of IBM Worklight V5.0.6.

To identify the new or highly modified tutorials and samples, and to get more information about how you can use to get started with IBM Worklight, see “Getting started tutorials and samples” on page 29.

Miscellaneous changes in project files and behaviors

In IBM Worklight V5.0.6, several changes have an impact on different aspects of your work, such as the content or the organization of IBM Worklight projects, and how applications that are based on IBM Worklight API behave. In some cases, you might have to manually update your projects or applications. For a comprehensive description of these changes and their potential impact, see “Migrating to a newer version of IBM Worklight” on page 22.

Known limitations

This topic describes general limitations to the current version of IBM Worklight. Limitations to a specific feature are explained in the topic that describes the feature.

In this documentation, you can find the description of IBM Worklight known limitations in different locations:

- When the known limitation applies to a specific feature, you can find its description in the topic that explains this specific feature. You can then immediately identify how it affects the feature.
- When the known limitation is general, that is, applies to different and possibly not directly related topics, you can find its description here.

Experimental IBM Worklight online user documentation.
This document is provided "as is".
In case of issues refer to the online user documentation.

Note: You might find complementary information about product known limitations or issues in the product Technotes.

Globalization

If you are developing apps for international users, be aware of the following restrictions:

- Translated versions of the product are not supplied.
- The Worklight Studio and Worklight Console provide only partial support for bidirectional languages.
- In Worklight Studio and Worklight Console, dates and numbers might not be formatted according to the locale.
- Names of projects, apps, and adapters must be composed only of the following characters:
  - Uppercase and lowercase letters (A-Z and a-z)
  - Digits (0-9)
  - Underscore (_)
- There is no support for Unicode characters outside the Basic Multilingual Plane.

You might also experience restrictions or anomalies in various aspects of globalization because of limitations in other products, such as the database management system and software development kits in use.

Rich Page Editor

The Rich Page Editor fails to show your page when the code that initializes it attempts to communicate with Worklight Server.

The Rich Page Editor simulates the mobile device environment without any connection to a real server. If the code that initializes your page tries to communicate with Worklight Server, a failure occurs. Because of this failure, the page content remains hidden, and you cannot use the Design pane of the Rich Page Editor.

As an example, a failure occurs if your page calls an adapter procedure in the `wlCommonInit()` function or the `wlEnvInit()` function.

In general, however, the initialization code is not strictly necessary to get a reasonable visual rendering of your page. To avoid this limitation, temporarily remove the `"display: none"` style from the `body` element in your page. Your page then renders even if the initialization functions do not execute completely.

Setting up IBM Worklight Studio

To set up the development environment that you need to create mobile applications with IBM Worklight, you install Worklight Studio and required SDKs, change the port number of the internal server, and upgrade your development environment.

About this task

This collection of topics is intended for developers who want to set up the development environment that you need to create mobile applications with IBM Worklight. It guides you through the steps of installing the software prerequisites

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

and the Eclipse-based IBM Worklight Studio.

Procedure

1. Review the list of supported operating systems and software development kits (SDK).
2. Install Worklight Studio.
3. Install the SDKs that you require.
4. Change the port number of the internal server.
5. Upgrade your development environment.

Installing IBM Worklight Studio

The installation method varies, depending on which version of IBM Worklight you use, and whether you want to install it into an existing Eclipse instance.

Before you begin

- You must install Worklight Studio Developer Edition with P2 Eclipse update. It provides every Worklight Studio function available in the Consumer and Enterprise Editions, except for some security-related features.
- You must install Worklight Studio Consumer Edition and Worklight Studio Enterprise Edition either with IBM Installation Manager or with P2 Eclipse update.

  To know more about the specificities of IBM Worklight Developer Edition, IBM Worklight Consumer Edition, and IBM Worklight Enterprise Edition, see the section *Editions of IBM Worklight* in "Introducing IBM Worklight" on page 1.

About this task

Notes:

1. If the `root` user installs the product but a non-root user runs the product, then the non-root user must be able to access the directory where the Java Runtime Environment (JRE) is located.
2. If you are using Apple Mac OS X, you must install Worklight Studio from the Eclipse update site.

Procedure

- To install Worklight Studio Developer Edition, go to the IBM Mobile development website at https://www.ibm.com/developerworks/mobile/worklight.html.
- To install Worklight Studio Consumer Edition or Worklight Studio Enterprise Edition:
  1. From the DVD or an image downloaded from Passport Advantage® using IBM Installation Manager, see "Installing IBM Worklight Studio from the DVD or from an image downloaded from Passport Advantage."
  2. Into an existing Eclipse IDE from an update site or the installation disk, see "Installing IBM Worklight Studio into an existing Eclipse IDE from an update site or the installation disk" on page 15.

Installing IBM Worklight Studio from the DVD or from an image downloaded from Passport Advantage

You can install Worklight Studio from a DVD or from an electronic image downloaded from IBM Passport Advantage using IBM Installation Manager.

Experimental IBM Worklight offline user documentation
This document is provided "as-is".
In case of issues, refer to the online user documentation.

Before you begin

- Ensure that your computer meets the system requirements for the software that you are installing.
- If you downloaded the software from IBM Passport Advantage, ensure that you extracted the contents of the compressed files. The extracted contents are in the IWS/disk1 directory where you extracted the contents.
- If you want Installation Manager to search for the latest available version of the software that you are installing, ensure that you are connected to the Internet.
- If you do not have IBM Installation Manager 1.6.1 or later, perform the additional installation steps of inserting the IBM Rational® Enterprise Deployment DVD and install the IBM Installation Manager from it.

About this task

This method is not applicable for Apple Mac OS X. If you are using Mac OS X, use the Eclipse update site as described in "Installing IBM Worklight Studio into an existing Eclipse IDE from an update site or the installation disk."

Procedure

1. Insert the IBM Worklight or IBM Mobile Foundation DVD.
2. Start IBM Installation Manager.
3. On the **File** menu, click **Preferences**.
4. Click **Add Repository**.
5. Enter the fully qualified path to the following directory: `*dvd-root*/IWS/disk1`
6. In the Add Repository window, click **OK**.
7. In the Preferences window, click **OK**.
8. In the main IBM Installation Manager window, click **Install**.
9. Follow the steps presented to complete the installation.

Installing IBM Worklight Studio into an existing Eclipse IDE from an update site or the installation disk

If you already have an Eclipse IDE installed, you can install Worklight Studio from your Eclipse IDE workbench.

Before you begin

- Ensure that your computer meets the system requirements for the software that you are installing.

  Restriction: You cannot install into an existing Eclipse 3.6.2 instance that uses a version 1.7 JRE; the installation does not complete.
- If you downloaded the software from IBM Passport Advantage, ensure that you extracted the contents of the compressed files. The extracted installation files are in the *target directory*/IWS/disk1 directory.

Procedure

1. Start your Eclipse IDE workbench.
2. Click **Help** > **Install new software**.
3. Optional: If your Eclipse workbench is Eclipse Classic, version 3.6.2:
   a. Beside the **Work with** field, click **Add**.

Experimental IBM Worklight online user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

   b. In the Add Repository window, enter the following web address in the **Location** field and then click **OK**: `http://download.eclipse.org/webtools/repository/helios/`.
4. Beside the **Work with** field, click **Add**.
5. In the Add Repository window, click **Archive**.
6. Browse to the update-site directory on the installation disk or download installation files (for example, *`target directory`*`/IWS/disk1/update_site`).
7. Select the update site .zip file and then click **Open**.
8. Select the features of IBM Worklight Studio that you want to install, and then click **Next**.
9. On the Install Details page, review the features that you install, and then click **Next**.
10. On the Review Licenses page, review the license text. If you agree to the terms, click **I accept the terms of the license agreement** and then click **Finish**. The installation process starts.
11. When the installation process completes, restart the workbench.

What to do next

Before you run Worklight Studio, determine whether you need to run additional post-installation tasks.

Important: After you finish installing IBM Worklight Developer Edition, when you restart Eclipse, click **Window** > **Open perspective** > **Other**, and select **Design** in the Open Perspective window. If you do not take this action, when you click **File** > **New**, you do not see the Worklight entries. After taking the action, when you click **File** > **New**, you do see the Worklight entries.

Running post-installation tasks:

Before you run IBM Worklight Studio, you may need to run additional post-installation tasks. Which tasks you need to run depends on whether you are installing into an existing Eclipse environment, and whether you have IBM Rational Team Concert™ installed.

Updating `eclipse.ini` for IBM Worklight Studio:

If you install Worklight Studio in to an existing Eclipse installation, you must manually update the `eclipse.ini` file to successfully run Worklight Studio.

Procedure
1. Stop the workbench.
2. Locate the `eclipse.ini` file in *`eclipse_installation_directory`*`\eclipse\eclipse.ini`.
3. Make a copy of the `eclipse.ini` file, or back it up.
4. Open the `eclipse.ini` file in a text editor.
5. Search for a line that starts with **`-XX:MaxPermSize`**.
6. If this line does not exist, insert a line in the `eclipse.ini` file, and start it with **`-XX:MaxPermSize=`**.
7. Edit the content of this (existing or new) line to make sure that it matches what is indicated next for the JDK that you are using:
   - For an Oracle 32-bit Java Runtime Environment (JRE): `-XX:MaxPermSize=320m`

Experimental IBM Worklight offline user documentation
This document is provided "as is"
In case of issues, refer to the online user documentation.

- For an Oracle 64-bit Java Runtime Environment (JRE): `-XX:MaxPermSize=512m`

8. If you are using a Java Development Kit 1.7 (JDK) with Eclipse 3.6.2, add the following line to the end of the `eclipse.ini` file:
   `-Djava.util.Arrays.useLegacyMergeSort=true`
9. Save and close the file.

Results

You can now start and work with the product.

What to do next

If your Eclipse workbench has IBM Rational Team Concert, version 4.0, Eclipse Client already installed, the Worklight Studio plug-ins might not be properly activated when you open an existing workbench. For example, the wizard **New > IBM Worklight Project** might not be available. To work around this problem, follow the instructions in “Running additional tasks for Rational Team Concert V4.0.”

Running additional tasks for Rational Team Concert V4.0:

You might need to clean the Eclipse environment before you run Worklight Studio.

About this task

If your Eclipse workbench has IBM Rational Team Concert, version 4.0, Eclipse Client already installed, the IBM Worklight Studio plug-ins might not be properly activated when you open an existing workbench. For example, the wizard **New > IBM Worklight Project** might not be available. To work around this problem, follow these instructions.

Note: You need to perform these steps only the first time that you start the product.

Procedure

1. Stop the workbench.
2. Locate the `eclipse.ini` file in *`eclipse_installation_directory`*`\eclipse\eclipse.ini`.
3. Make a copy of the `eclipse.ini` file, or back it up.
4. Open the `eclipse.ini` file in a text editor.
5. Append the following text on a new line: `-clean`
6. Save and close the file.
7. Start the product and select a workspace. You should be able to successfully open the workspace.
8. Remove the `-clean` line from the `eclipse.ini` file and save the file.

Troubleshooting IBM Worklight Studio installation

You can troubleshoot to find the cause of IBM Worklight Studio installation failure.

IBM Worklight Studio installation errors with Eclipse 3.6.2

If an error message is displayed when you install Worklight Studio on Eclipse 3.6.2, install all available Eclipse updates first, and then install Worklight Studio.

Experimental IBM Worklight offline user documentation
This document is provided as is.
In case of issues refer to the online user documentation.

About this task

Use this procedure if the following message is displayed shortly after you start to install Worklight Studio on Eclipse 3.6.2:

```
Cannot complete the install because one or more required items could not be found.
  Software being installed: IBM jQuery Mobile Tools 5.0.5001.v20121214_1542 (com.ibm.webtools.jquery
  Missing requirement: IBM Web Editor Common 1.1.0.v20121012_0113 (com.ibm.etools.webtools.webedit.co
  Cannot satisfy dependency:
    From: Rich Page Editor Base 1.0.200.v20121101_1546 (com.ibm.etools.rpe.feature.feature.group 1.0
    To: com.ibm.etools.rpe.html [1.0.200.v20121019_2352]
  Cannot satisfy dependency:
    From: Rich Page Editor - HTML 1.0.200.v20121019_2352 (com.ibm.etools.rpe.html 1.0.200.v20121019_2
    To: bundle com.ibm.etools.webtools.webedit.common [1.0.0,2.0.0)
  Cannot satisfy dependency:
    From: IBM Worklight Studio 5.0.5001.v20121217_1857 (com.ibm.imp.tools.feature.feature.group 5.0.5
    To: com.ibm.etools.rpe.feature.feature.group 1.0.200
  Cannot satisfy dependency:
    From: IBM jQuery Mobile Tools 5.0.5001.v20121214_1542 (com.ibm.webtools.jquery.tools.feature.feat
    To: com.ibm.imp.tools.feature.feature.group 5.0.5001
```

This issue does not arise when installing on Eclipse 3.7.2 or Eclipse 4.2.1.

Procedure

1. In Eclipse, run **Help** > **Check for Updates**.
2. Install all available updates.
3. Install Worklight Studio.

Upgrading IBM Worklight Studio in Eclipse

If you have an earlier version of Worklight Studio, you can use Eclipse to upgrade to the current version.

Procedure

1. Start your Eclipse IDE workbench.
2. If you previously installed Worklight Studio from a local archive, perform these steps:
   a. Download the latest Worklight version archive file.
   b. In Eclipse, click **Window** > **Preferences** > **Install/Update** > **Available Software Sites**
   c. Click **Add**.
   d. In the Add Repository window, click **Archive**.
   e. Browse to the update-site directory on the installation disk or download installation files *target directory*/`IWS/disk1/update_site`.
3. Click **Help** > **Check for updates**.
4. Follow the instructions that are provided by the Eclipse IDE Workbench to update your installation of Worklight Studio.

Results

Worklight Studio is updated.

Note:
If the update appears to hang, it might be because you are using a bad mirror. Add this line to your `eclipse.ini` file to resolve the problem.

```
-Declipse.p2.mirrors=false
```

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

Starting IBM Worklight Studio installed with IBM Installation Manager

If you installed IBM Worklight Consumer Edition or Enterprise Edition with IBM Installation Manager, you can start Worklight Studio in one of two ways.

Procedure

Attention: When you install Worklight Studio with IBM Installation Manager, do not start Worklight Studio by running the `eclipse.exe` file on Windows, or `eclipse` on Linux (you only run these files when you install Worklight Studio with P2 Eclipse update. See "Starting IBM Worklight Studio installed with P2 Eclipse update"). Though Worklight Studio appears to start normally, you might later experience problems such as the Android emulator failing to start. Use one of the following techniques:

- Click **Start** > **All Programs** > **IBM Software Delivery Platform** > **IBM Worklight Studio 5.0.6** > **IBM Worklight Studio**.
- Double-click the *`INSTALL_DIR`*`/sdp/Worklight.cmd` file (on Windows) or `Worklight.sh` (on Linux), where *INSTALL_DIR* is your Worklight installation directory.

Starting IBM Worklight Studio installed with P2 Eclipse update

If you use IBM Worklight Developer Edition, or if you installed IBM Worklight Consumer or Enterprise Edition with P2 Eclipse update, start Worklight Studio by running the Eclipse executable file.

Procedure

- On Linux systems, run the `eclipse` file.
- On Windows systems, run the `eclipse.exe` file.

Installing mobile specific tools

When you develop mobile applications, you must install and use specialized tools (such as SDKs). These tools depend on the operating system that you develop the applications for (such as iOS or Android).

This collection of topics details the required tools for each operating system.

Installing tools for Adobe AIR

To build and sign applications for Adobe AIR, you must install the Adobe AIR SDK.

Procedure

1. Download the Adobe Air SDK from the Air SDK on Adobe website.
2. Unpack the archive into a folder of your choice.
3. Set an environment variable (either locally or on the central build server) named `AIR_HOME`, pointing to the place where you opened the SDK. The Worklight Builder uses this environment variable to run the build and sign tool when building AIR applications.

Installing tools for iOS

To build and sign applications for iOS, you must install the latest Xcode IDE (including the iOS simulator) on a Mac.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Procedure

1. Register as an Apple developer on the Apple Registration Center website at https://developer.apple.com/programs/register/.
2. Download Xcode from the Mac App Store at http://www.apple.com/osx/apps/app-store.html.
3. Install Xcode on your Mac.

   For more information about the iOS development environment, see the module *Setting Up Your iOS Development Environment*, under category 1, *Setting up your development environment*, in "Getting started tutorials and samples" on page 29.

Installing tools for Android

To build and sign applications for Android, you must install the Android SDK and the Android Development Tools plug-in for Eclipse.

Procedure

1. Install the Android SDK available at http://developer.android.com/sdk/.
2. Install the Android Development Tools plug-in for Eclipse available at https://dl-ssl.google.com/android/eclipse/.
3. Add SDK Platform and Virtual Devices to the SDK.

   For more information about the Android development environment, see the module *Setting Up Your Android Development Environment*, under category 1, *Setting up your development environment*, in "Getting started tutorials and samples" on page 29.

Installing tools for BlackBerry

To build and sign applications for BlackBerry OS 6, 7 or 10, you must install the WebWorks tools.

Procedure

1. Download Ripple emulator available at https://developer.blackberry.com/html5/download/ and install it.
2. Download WebWorks SDK from the same site, at https://developer.blackberry.com/html5/download/, and install it to the folder of your choice.
3. (Only for BlackBerry 10) Set an environment variable (either locally or on the central build server) named *WEBWORKS_HOME*, pointing to the SDK root folder. The Worklight® Builder uses this environment variable when it builds BlackBerry 10 applications. On each build, the environment variable value is transferred to `native\project.properties`.

   Note: *WEBWORKS_HOME* must be set before you start Worklight Studio. This variable is important for the normal operation of the client. If you use Ant scripts to build and deploy the application to the device, and the *WEBWORKS_HOME* value is set incorrectly, your file structure might become corrupted, and produce a new directory with the incorrect *WEBWORKS_HOME* value name.
4. Download and install a simulator.

   For more information about the BlackBerry development environment, see the module *Setting Up Your BlackBerry 6 and 7 Development Environment* and *Setting up your BlackBerry 10 development environment*, under category 1, *Setting up your development environment*, in "Getting started tutorials and samples" on page 29.

Experimental IBM Worklight offline documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Installing tools for Windows Phone 7.5

To build and sign applications for Windows Phone 7.5, you must install Microsoft Visual Studio Express 2010 or 2012 for Windows Phone, and Zune Software.

Procedure

1. Download and install Microsoft Visual Studio Express, available at http://www.microsoft.com/visualstudio/eng/products/visual-studio-2010-express for the 2010 edition, and at http://www.microsoft.com/visualstudio/eng/products/visual-studio-express-for-windows-phone for the 2012 edition.
2. Download Zune software from http://www.zune.net/enUS/products/software/download/default.htm and install it.

   For more information about the Windows Phone 7.5 development environment, see the module *Setting Up Your Windows Phone 7.5 Development Environment*, under category 1, *Setting up your development environment*, in "Getting started tutorials and samples" on page 29.

Installing tools for Windows Phone 8

To build and sign applications for Windows Phone 8, you must install Microsoft Visual Studio Express 2012 for Windows Phone.

Procedure

Download Microsoft Visual Studio Express 2012 from http://www.microsoft.com/visualstudio/eng/products/visual-studio-express-for-windows-phone and install it. For more information about the Windows Phone 8 development environment, see the module *Setting Up Your Windows Phone 8 Development Environment*, under category 1, *Setting up your development environment*, in "Getting started tutorials and samples" on page 29.

Installing tools for Windows 8

Windows Store apps run only on Windows 8, so to develop Windows Store apps, you need Windows 8 and some developer tools.

Procedure

1. After you install Windows 8, go to http://msdn.microsoft.com/en-us/windows/apps/br229516.aspx.
2. Click **Download now** under "Download the tools and SDK".
3. Download Microsoft Visual Studio Express® 2012 from http://www.microsoft.com/visualstudio/eng/products/visual-studio-express-for-windows-phone and install it. Microsoft Visual Studio Express 2012 for Windows 8 includes the Windows 8 SDK. It also gives you tools to code, debug, package, and deploy a Windows Store app.
4. Start Visual Studio Express 2012. You will be prompted to obtain a developer license. You need a developer license to install, develop, test, and evaluate Windows Store apps.
5. For more information about obtaining a developer license, see http://msdn.microsoft.com/en-us/library/windows/apps/hh974578.aspx.

Changing the port number of the internal application server

If the default port number is already in use, edit the `eclipse.ini` file to change to a different port.

Experimental IBM Worklight online user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

About this task

When you start Eclipse with Worklight Studio, an embedded application server is started automatically to host a Worklight Server instance for your adapters and apps. This internal server uses port 8080 by default.

If port 8080 is occupied by another application that is running on your development computer, you might need to configure the Worklight Studio internal server to use a different port. To do so, follow these instructions:

Procedure

1. In the Eclipse home folder, open the `eclipse.ini` file.
2. After the `-vmargs` line, add: `-Dworklight.port=8085` (or any other port number of your choice).
3. Restart Eclipse.

What to do next

If you develop Facebook apps or mobile web apps, also set this port as the value of the parameter `publicWorkLightPort` in the *`project-name`*`/server/conf/worklight.properties` file of each of your Worklight projects. Doing this ensures that the Worklight Console features **Get URL for Facebook** and **Embed in Web Page** work correctly.

The development of Facebook apps is deprecated in Worklight version 5.0.5. Support might be removed in any future version. Use standard desktop web apps.

Migrating to a newer version of IBM Worklight

When you open your IBM Worklight project with IBM Worklight Studio V5.0.6, your project is automatically updated to this new version. However, some parts of your application require manual updates that are related to new versions of some software and to some changes in the IBM Worklight environment. Be aware of some modifications, such as changes in file names and structure.

This topic focuses on the migration process from IBM Worklight V5.0.5 to V5.0.6. To know about the migration process from IBM Worklight V5.0.0.3 to V5.0.5, see “Migrating from Worklight V5.0.0.3 to V5.0.5” on page 26.

Worklight Server

For instructions on how to upgrade Worklight Server to V5.0.6 in your development environment, see Installing Fix Packs for IBM Worklight V5.0.

Important:

If you are upgrading Worklight Server in a production environment, the process can be longer and more complicated, especially if you have existing IBM Worklight applications that run in a Worklight Server environment. For instructions on how to upgrade your production Worklight Server, see “Upgrading IBM Worklight Server in a production environment” on page 28.

Usage of existing applications

If you want to use existing applications with a new server version, see “Setting up existing applications with a new server version” on page 371.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

WL.App.close:

Note: The WL.App.close API quits the application and the iOS Human Interface Guidelines now indicate that the code for an iOS app must not contain a call to exit the app. Consider no longer using WL.App.close API in your apps. For more information, see "WL.App.close" on page 177

Third-party libraries

Cordova: for Android, iOS, Windows Phone 8, BlackBerry 10 and Windows 8, IBM Worklight V5.0.6 is now based on Cordova 2.3. Cordova 2.3 includes deprecated and modified items compared to the earlier version. The upgrade process for the Cordova 2.3 configuration is automated when the IBM Worklight project is built in Studio or with the Ant tasks. To view the Cordova change log, go to https://issues.apache.org/jira/browse/CB# and click *Change Log*. To know more about the migration steps in Cordova, see http://cordova.apache.org/docs/en/2.3.0/ and click *Upgrading Guides*.

- **Deprecated item**

  Cordova 2.3 deprecates the **`device.name`** property for all platforms. This property returned both the name of the user and of the device model (for example, `Jane's iPhone 5`). Now it returns the name of the device (for example, `iPhone`). For all platforms, the new property **`device.model`** returns the specific device model name (for example, `iPhone 5`).

  To know more about deprecation in Cordova, see http://wiki.apache.org/cordova/DeprecationPolicy.

- **Modified items**
  - The iOS configuration `Cordova.plist` file was changed to the `config.xml` file. It now comes in the same format as the Android `config.xml` file.

    The following `Cordova.plist` configuration elements are automatically migrated into the `config.xml` file by the Worklight upgrade:
    - The Cordova built-in plug-ins, now under the `<!--Cordova-->` section
    - The IBM Worklight plug-ins, now under the `<!--Worklight-->` section
    - The user/custom plug-ins, now under the `<!--User-->` section
    - All the State of Cordova preferences

    You must manually update the following `Cordova.plist` configuration elements because they are not migrated automatically:
    - Changes to the `<access origin>` element from the default setting
    - Custom preferences
  - For Windows Phone 7.5 applications, the namespace of the Cordova classes changed from `WP7CordovaClassLib` to `WPCordovaClassLib`. If you wrote a custom plug-in that relies on this namespace (for example, `using WP7CordovaClassLib`), you must change this C# code to address the new namespace (for example, `using WPCordovaClassLib`).
  - The optional **`callback`** parameter is added to the `WL.App.copyToClipboard` method as a new way of invocation.

Dojo: IBM Worklight Studio now ships with Dojo V1.8.3, which has a number of iOS fixes. There is no automated upgrade process available, so you must make these updates manually.

For more information about upgrading to Dojo V1.8.3, see "Dojo iOS fixes" on page 25.

Experimental IBM Worklight first user documentation
This document is provided as is.
In case of issues refer to the online user documentation.

If you created your current project with an earlier version of IBM Worklight Studio, consider migrating the code to the new Dojo module loading technique in addition to upgrading the Dojo toolkit. It ensures that the code performs more reliably and that the page continues to work when it makes further changes in RPE.

Specifically, the Dojo layers are no longer loaded from HTML elements, but instead they are loaded by **`require()`** calls inside the **`wlCommonInit()`** method. The individual modules are loaded from **`require()`** calls inside the **`dojoInit()`** method.

For more information about migrating your code to Dojo 1.8.3, see "Dojo 1.8.3 code migration" on page 26.

Changes in projects

Native SDK:

- For iOS: manually upgrade your native iOS project to use IBM Worklight V5.0.6 iOS Native Runtime libraries (`WorklightAPI` folder), which supports push notifications.
- For Android: manually upgrade your native Android project to use IBM Worklight V5.0.6 Android Native Runtime libraries (`worklight-android.jar`), which supports push notifications.

  For more information, see "Development guidelines for using native API" on page 84.

Custom code for Android app: the `onCreate` method to add custom code to your Android app is deprecated. It is now replaced with the `onWLInitCompleted` method. To know more about custom code for an Android app, see "Adding custom code to an Android app" on page 78.

iOS apps: add the following code to your main iOS project `m` file, `${projectName}.m`:

```
-(void) didFinishWLNativeInit:(NSNotification *)notification {
}
```

Java ME: manually upgrade your native Java ME project to use IBM Worklight V5.0.6 Java ME Native Runtime libraries (`worklight-javame.jar` and `json4javame.jar`), which support authentication and application management. To see the messages from the admin console, the application must update its `WLClient.createInstance()` API. See "Java client-side API for Java ME apps" on page 271 and the module *Using Worklight API in Native Java ME applications*, under category 7, *Developing native applications with Worklight*, in the "Getting started tutorials and samples" on page 29.

Windows Phone 8 `applicationBar` folder: when you migrate a Windows Phone 8 project to IBM Worklight 5.0.6, the `images/applicationBar` folder under the root of the IBM Worklight project becomes the `nativeResources/applicationBar` folder and stays under the root of the Worklight project. See "WL.OptionsMenu.addItem" on page 252.

Changes in features

Push notifications: the **`notificationOptions`** parameter has a new JSON structure for push notifications. The old JSON block is now deprecated. When this deprecated JSON block is used:

Experimental IBM Worklight offline use documentation.
This document is provided "as is".
In case of doubt, refer to the online user documentation.

- The Studio console displays a deprecation warning message.
- All the previously supported environments (iOS, Android, SMS) receive a notification message. The Windows Phone 8 environment receives two notifications: a tile message, which contains the badge and the alert, and a raw message, which contains the payload.

Changes in API

JavaScript client-side API: the `WL.OptionsMenu.isEnabled` and `WL.OptionsMenu.isVisible` methods now take a callback function as a parameter. The callback is called by Cordova after the request is processed, and it receives the current enabled or visible state.

Interface WorkLightLoginModule: the interface WorkLightLoginModule is now deprecated and is replaced with the interface WorkLightAuthLoginModule, where the new `createIdentity` method replaces the previous `createIdenity` method.

Changes in sessions configuration

Default sessions settings: the default value of **`serverSessionTimeout`**, after which the IBM Worklight session is invalidated, changed from 30 to 10 minutes.

The default value of **`heartBeatIntervalInSecs`** sent by `WLClient` to Worklight Server changed from 1200 (20 minutes) to 420 (7 minutes).

New SSL properties: the `worklight.properties` file contains new common SSL properties: `ssl.keystore`. Now the properties of **`ws-security`** are deprecated and they are linked by default to the common SSL properties.

Changes in Application Center

Application Center console: the URL of the Application Center console changed. You can now start the Application Center console by entering this address in your browser: `http://localhost:9080/appcenterconsole/`.

Dojo iOS fixes

IBM Worklight Studio V5.0.5 ships with Dojo V1.8.1 and Worklight Studio V5.0.6 ships with Dojo V1.8.3. These versions have a number of iOS fixes. There is no automated upgrade process available, so you must make these updates manually.

About this task

Complete the following procedure to manually implement the iOS fixes shipped with Dojo.

Procedure

1. Open your workspace using IBM Worklight Studio.
2. Right-click the project you want to migrate and click **Properties**.
3. Click **Project Facets**.
4. Uncheck **Web 2.0-> Dojo Toolkit** to uninstall the tooling for Dojo. It does not uninstall Dojo.
5. Click **OK** to close the project **Properties** page.

Experimental IBM Worklight online user documentation
This document is provided as is.
In case of issues, refer to the online user documentation.

6. Find the folder *{PROJECT_NAME}* > **dojo** and delete all the children under this folder. Do not delete the *{PROJECT_NAME}* > **dojo** folder. It must be empty.
7. Right-click the project again and go to the **Properties** page again.
8. Click **Project Facets**.
9. Check **Web 2.0-> Dojo Toolkit** to install the tooling for Dojo. You have now upgraded to a new Dojo version.
10. You might have to upgrade some application content to work with Dojo. Run and test your application to make sure things work with this version.

Dojo 1.8.3 code migration

The Dojo layers are now loaded by **`require()`** calls inside the **`wlCommonInit()`** method, so you must modify the existing code in the HTML file that loads the layers and the modules.

About this task

Make the following changes to migrate your existing code.

Procedure

1. Remove the `<script>` elements from the HTML file that loads the layers and replace them with a **`require()`** call in the **`wlCommonInit()`** method (see the code snippet later in this section).
2. If you have the **`require()`** call in the HTML file that loads the individual modules, move it into the **`dojoInit()`** method (see the code snippet later in this section).
3. If you use the deviceTheme module (`dojox/mobile/theme`), remove it from the **`require()`** call and instead use a `<script>` element to load it from inside the HTML file. Make sure this element comes before the `<script>` element for `dojo.js`.

Example

The following code snippet shows the new technique:

```
function wlCommonInit(){
        require([ "dojo/core-web-layer", "dojo/mobile-ui-layer"], dojoInit);
        // Common initialization code goes here
}
function dojoInit() {
        require([ "dojo", "dojo/parser", "dojox/mobile", "dojox/mobile/ScrollableView"],
                        function(dojo, dijit) {
                                dojo.ready(function() {

                                });
                        });
}
```

Migrating from Worklight V5.0.0.3 to V5.0.5

When you open your IBM Worklight project with a newer version of IBMWorklight Studio, your project is automatically updated to this new version. However, there are some parts of your application that require manual updates.

Updates in IBM Worklight V5.0.5 include modifications related to new versions of some software, and some changes to the IBM Worklight environments. If you are using the IBM Worklight Application Center, there are also some configuration changes. Some environments and files were deprecated.

Experimental IBM Worklight online user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

IBM Worklight Core Development

In IBM Worklight V5.0.5, the Embedded environment is now called the Desktop Web App environment. Everything is upgraded automatically, however, the application URL contains the name of the environment, and therefore:

- For new environments, the URL is `.../desktopbrowser/...` and NOT `.../embedded/...`.
- For old environments, both URLs are supported.

Deprecated environments

With IBM Worklight V5.0.5, the following environments are now deprecated:

- Facebook
- iGoogle
- Windows 7 Gadgets
- Mac OS dashboard widgets

Changed default behavior for connection on startup

The **`connectOnStartup`** property in the `initOptions.js` file now defaults to `false`, rather than `true` as in earlier versions. Applications that do not need to connect to the server when they start might now start more quickly. However, if your application must connect to the server when it starts, you must change the value of **`connectOnStartup`**. For more information, see "Connecting to Worklight Server" on page 77.

IBM Worklight Application Center

When migrating from version 5.0.0.3 to 5.0.5, the Derby database for the Application Center is migrated to the new format as part of the installation.

When migrating from V5.0.0.3 to V5.0.5, you must adapt the security roles and configuration, because the application center in V5.0.5 has a new J2EE security role named **`appcenteruser`**, which consists of the group of users authorized to use the mobile client. See "Configuration of the Application Center after installation" on page 429 to learn how to set up these security roles.

Cordova

IBM Worklight 5.0.5 is based on Cordova 2.2.

Since Cordova 2.0, Cordova deprecates the `cordova.xml` and `plugins.xml` files. Cordova replaces these two files with a single `config.xml` file, which combines the two deprecated files. You can find the new `config.xml` file in the same `native/res/xml` folder as the deprecated `cordova.xml` and `plugins.xml` files.

For example, if you develop a native application for the Android environment, you can find the `config.xml` file in the `android/native/res/xml` folder of your application folder.

The upgrade process for Cordova configuration is automated. However, if you have applicative code that is calling Cordova API, consider checking for changes in the new Cordova API and manually fix your code. IBM Worklight version 5.0.0.3 was bundled with Cordova 1.6.1. For information about Cordova changes, review

Experimental IBM Worklight online user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

the release notes in the Apache Cordova Change Log site: The Apache Software Foundation.

jQuery

A new release of IBM Worklight's internal tool `jQuery` version 1.8.1 means that you must manually upgrade your JavaScript UI libraries, for example `jQuery Mobile`.

Dojo

IBM Worklight Studio V5.0.5 ships with Dojo V1.8.1, which has a number of iOS fixes. There is no automated upgrade process available, so you must make these updates manually.

For more information about upgrading to Dojo V1.8.1, see "Dojo iOS fixes" on page 25.

If you have existing IBM Worklight Dojo projects created with a previous version, it is highly recommended that you migrate the code to the new architecture. This ensures that the code performs more reliably and that the page continues to work when making further changes in RPE, as the new tools insert **`require()`** calls into a method called **`dojoInit()`**

The layers are no longer loaded from HTML elements, but instead they are loaded by **`require()`** calls inside the **`wlCommonInit()`** method. The individual modules are not loaded from **`require()`** calls inside the HTML page, but from **`require()`** calls inside the **`dojoInit()`** method.

Upgrading IBM Worklight Server in a production environment

Upgrading to IBM Worklight Server V5.0.6.x in a production environment is a more exacting process than in your development environment because you must back up your data and prepare for the upgrade carefully to minimize production downtime.

When you upgrade from Worklight Server V5.0.5.x to V5.0.6.x in a production environment, the process can be more complicated than upgrading to a new version in your development environment. The procedure is also longer if you have existing Worklight applications that run in a production Worklight Server environment. For step-by-step instructions on how to upgrade your production Worklight Server to V5.0.6, see Instructions for Upgrade from Worklight Server 5.0.5.x to 5.0.6.1 in a Production Environment.

The document at this link provides essential information about migrating your existing IBM Worklight projects and applications to the new version, backing up any existing databases or Application Server data, and performing other preparation tasks that must be completed before you install the new version of Worklight Server. These preparation steps are followed by post-installation, verification, and configuration tasks that must be completed before you restart the new Worklight Server and finish migrating your updated Worklight applications.

The upgrade procedure can take some time, several hours in fact, and so these activities must be scheduled to create the least disruption and downtime to production servers and the applications that run on them.

Experimental IBM Worklight offline user documentation. This document is provided "as-is". In case of issues refer to the online user documentation.

Setting up IBM Worklight Server

To deploy your mobile applications to the test or production environments, you must install and configure IBM Worklight Server.

For information about Worklight Server, and instructions on how to install and configure it, see Chapter 6, "IBM Worklight Server administration," on page 311.

Getting started tutorials and samples

Getting started tutorials and samples help you learn about IBM Worklight and evaluate what the product can do for you.

You can get started with IBM Worklight by following the "Tutorials."

You can further learn how to develop mobile applications with IBM Worklight by studying the following samples:
- "Worklight Starter application samples" on page 36
- "JavaScript framework-based application samples" on page 37

You can find links to download compressed files that contain the materials for the tutorials and samples in "Additional resources" on page 37.

Important: These materials have been created for use with only the IBM Worklight Developer Edition and the Worklight Server inside Eclipse. If your configuration differs, you might have to adapt the exercise instructions, the code samples, or both.

Terms and conditions: The following resources are subject to these "Terms and conditions" on page 37, and may include applicable third-party licenses. Please review the third-party licenses before using any of the resources. The third-party licenses applicable to each sample are available in the `notices.txt` file that is included with each code sample.

Tutorials

Use the tutorials to learn the most important features of IBM Worklight.

Each tutorial is composed of one module and generally one companion sample:
- The module is a PDF presentation file that provides step-by-step guidance on how to get started with an important feature of IBM Worklight.
- The sample, if any, is a compressed (.zip) file that provides pieces of code or script files that accompany and support the module. If a module has some exercises, you also have a companion sample that provides the solutions to these exercises.

The modules and companion samples of the tutorials are organized in the following categories:
1. Setting up your development environment: With this category, you learn how to set up your development environment to work with IBM Worklight.
2. Hello Worklight: With this category, you learn how to create your first IBM Worklight app and preview it in different mobile operating systems.
3. Worklight client-side development basics: With this category, you learn how to use basic IBM Worklight APIs to develop your apps, how to build a

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

multi-page application, how to work with the user interface framework, how to debug and optimize your app, and some general information that you must know to work in each specific environment.

4. Worklight server-side development: With this category, you learn how to develop the server code (adapters) that your mobile application requires to integrate with enterprise back-end applications and cloud services.
5. Advanced client-side development: With this category, you learn how to implement different features in your mobile application, such as controls, skins, offline access, translation, encryption of sensitive data. You also learn how to develop your client application by using native APIs
6. Adding native functionality to hybrid applications with Apache Cordova: With this category, you learn how to use Apache Cordova with IBM Worklight, and how to use native pages in hybrid applications.
7. Developing native applications with Worklight: With this category, you learn how to develop native applications with IBM Worklight.
8. Authentication and security: With this category, you learn how to protect your applications and adapter procedures against unauthorized access by using authentication, login modules, and device provisioning.
9. Advanced topics: With this category, you learn advanced topics that you can use with IBM Worklight, such as how to develop by using shells or how to handle notifications.
10. Moving to production: With this category, you learn how to move the apps that you create from your development environment to the production environment.
11. Integrating with other products: With this category, you learn how IBM Worklight integrates with some other IBM products, such as IBM PureApplication System or Tivoli® Directory Server.

Note: Compared to the previous version of IBM Worklight, some modules are new or highly revised. To help you identify these new or revised modules, their names are introduced with either *NEW* or *Highly revised* in the following table.

Table 1. Getting Started modules and samples

Module	Sample (if any)	Description
1. Setting up your development environment		
Setting up your Worklight development environment		This module explains how to set up your environment.
Setting up your iOS development environment		This module complements the module "Setting up your Worklight development environment" with further steps that are required for iOS application development.
Setting up your Android development environment		This module complements the module "Setting up your Worklight development environment" with further steps that are required for Android application development.
Setting up your BlackBerry 6 and 7 development environment		This module complements the module "Setting up your Worklight development environment" with further steps that are required for BlackBerry 6 and BlackBerry 7 application development.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

Table 1. Getting Started modules and samples (continued)

Module	Sample (if any)	Description
Setting up your BlackBerry 10 development environment		This module complements the module "Setting up your Worklight development environment" with further steps that are required for BlackBerry 10 application development.
Setting up your Windows Phone 7.5 development environment		This module complements the module "Setting up your Worklight development environment" with further steps that are required for Windows Phone 7.5 application development.
Setting up your Windows Phone 8 development environment		This module complements the module "Setting up your Worklight development environment" with further steps that are required for Windows Phone 8 application development.
2. Hello Worklight		
Creating your first Worklight application	Exercise and code sample	This module explains how to set up your first mobile application.
Previewing your application on iOS		This module explains how to preview your application in the iOS environment.
Previewing your application on Android		This module explains how to preview your application in the Android environment.
Previewing your application on BlackBerry 6 and 7		This module explains how to preview your application in the BlackBerry 6 and BlackBerry 7 environments.
Previewing your application on BlackBerry 10		This module explains how to preview your application in the BlackBerry 10 environment.
Previewing your application on Windows Phone 7.5		This module explains how to preview your application in the Windows Phone 7.5 environment.
Previewing your application on Windows Phone 8		This module explains how to preview your application in the Windows Phone 8 environment.
3. Worklight client-side development basics		
Learning Worklight client side API	Exercise and code sample	This module explains the basics of the IBM Worklight Client API.
Building a multi page application	Exercise and code sample	This module explains how to build a multi-page application with IBM Worklight.
Working with UI frameworks		This module explains how to work with the user interface (UI) frameworks of IBM Worklight.
Debugging your applications		This module explains how to debug the client applications.
Optimizing your application for various environments		This module explains how to optimize the application code for specific environments.

Experimental IBM Worklight offline user documentation.
This document is provided as is.
In case of issues, refer to the online documentation.

Table 1. Getting Started modules and samples (continued)

Module	**Sample (if any)**	**Description**
General information when developing for iOS		This module gives some general information that you must know to develop apps for the iOS environment.
General information when developing for Android		This module gives some general information that you must know to develop apps for the Android environment.
General information when developing for BlackBerry 6 and 7		This module gives some general information that you must know to develop apps for the BlackBerry 6 and BlackBerry 7 environments.
General information when developing for BlackBerry 10		This module gives some general information that you must know to develop apps for the BlackBerry 10 environment.
General information when developing for Windows Phone 7.5		This module gives some general information that you must know to develop apps for the Windows Phone 7.5 environment.
General information when developing for Windows Phone 8		This module gives some general information that you must know to develop apps for the Windows Phone 8 environment.
General information when developing Mobile Web applications		This module gives some general information that you must know to develop mobile web applications.
General information when developing desktop applications		This module gives some general information that you must know to develop desktop applications.
4. Worklight server-side development		
Adapter framework overview		This module explains what adapters are in IBM Worklight, and how to work with them.
HTTP adapter - Communicating with HTTP back-end systems	Exercise and code sample	This module explains how to work with adapters to communicate with HTTP back-end systems.
SQL adapter - Communicating with SQL database	Exercise and code sample	This module explains how to work with adapters to communicate with SQL databases.
Cast Iron® adapter - Communicating with Cast Iron		This module explains how to work with adapters to communicate with Cast Iron.
JMS adapter - Communicating with JMS		This module explains how to work with adapters to communicate by using Java Message Service (JMS).
Invoking adapter procedures from client applications	Exercise and code sample	This module explains how to call the adapter procedures from the client application.
Advanced adapter usage and mashup	Exercise and code sample	This module explains advanced details on how to use adapters.

Experimental IBM Worklight offline user documentation.
This document is provided as is.
In case of issues, refer to the online user documentation.

Table 1. Getting Started modules and samples (continued)

Module	**Sample (if any)**	**Description**
Using Java in adapters	Exercise and code sample	This module explains how to use Java in adapters.
5. Advanced client side development		
Highly revised: Overview of client technologies		This module explains the technologies that support IBM Worklight clients.
Common UI controls	Exercise and code sample	This module explains the common user-interface controls in IBM Worklight.
Supporting multiple form-factors using Worklight skins	Exercise and code sample	This module explains how you can support multiple form factors by working with skins in IBM Worklight.
Working offline	Exercise and code sample	This module explains how to detect application connectivity failures and corresponding actions.
Enabling translation	Exercise and code sample	This module explains how to enable translation of the client applications.
Using Direct Update to quickly update your application		This module explains how to automatically update your applications with new versions of their web resources.
Storing sensitive data in Encrypted Cache	Exercise and code sample	This module explains how to work with the encrypted cache of the mobile device.
JSONStore - The client-side JSON-based database overview		This module introduces the JSONStore, and how you can work with JSON documents.
JSONStore - API basics	Exercise and code sample	This module explains the basic tasks that you can perform on a local JSON collection.
JSONStore - Synchronizing client and server databases	Exercise and code sample	This module explains how you can synchronize a local JSON collection with a server-side database.
JSONStore - Encrypting sensitive data	Exercise and code sample	This module explains how you can encrypt the sensitive data of your local JSON collection.
NEW: JSONStore - Encrypting sensitive data with FIPS 140-2	Exercise and code sample	This module explains how you can encrypt the sensitive data of your local JSON collection by using FIPS 140-2.
6. Adding native functionality to hybrid applications with Apache Cordova		
Highly revised: Apache Cordova overview		This module explains what Apache Cordova is, and how to use it with IBM Worklight.
iOS - Using native pages in hybrid applications	Exercise and code sample	This module explains how to use native pages in hybrid applications that are developed for the iOS environment.
Highly revised: iOS - Adding native functionality to hybrid application with Apache Cordova plugin	Exercise and code sample	This module explains how to use Apache Cordova plugs-in to add native functionality to hybrid applications that are developed for the iOS environment.

Experimental IBM Worklight offline user documentation.
This document is provided as is.
In case of issues, refer to the online user documentation.

Table 1. Getting Started modules and samples (continued)

Module	Sample (if any)	Description
Android - Using native pages in hybrid applications	Exercise and code sample	This module explains how to use native pages in hybrid applications that are developed for the Android environment.
Highly revised: Android - Adding native functionality to hybrid application with Apache Cordova plugin	Exercise and code sample	This module explains how to use Apache Cordova plug-ins to add native functionality to hybrid applications that are developed for the Android environment.
NEW: Windows Phone 8 - Adding native functionality to hybrid application with Apache Cordova plugin	Exercise and code sample	This module explains how to use Apache Cordova plug-ins to add native functionality to hybrid applications that are developed for the Windows Phone 8 environment.
7. Developing native applications with Worklight		
Using Worklight API in native iOS applications	Exercise and code sample (app) and Exercise and code sample (native API)	This module explains how to create a Worklight native API, and to use its components in a native iOS application.
Using Worklight API in native Android applications	Exercise and code sample (app) and Exercise and code sample (native API)	This module explains how to create a Worklight native API, and to use its components in a native Android application.
Using Worklight API in native Java ME applications	Exercise and code sample (app) and Exercise and code sample (native API)	This module explains how to use Java API to develop Java Platform, Micro Edition (Java ME) applications.
NEW: Using Worklight API for push notifications in native iOS applications	Exercise and code sample (app) and Exercise and code sample (native API)	This module explains how to use Worklight API to manage push notification a native iOS application.
NEW: Using Worklight API for push notifications in native Android applications	Exercise and code sample (app) and Exercise and code sample (native API)	This module explains how to use Worklight API to manage push notification a native Android application.
8. Authentication and security		
Authentication concepts		This module explains how to protects your applications and adapter procedures against unauthorized access by using authentication.
Form-based authentication	Exercise and code sample	This module explains how to work with the form-based authentication.

Experimental IBM Worklight online user documentation.
This document is provided "as-is".
In case of issues, refer to the online user documentation.

Table 1. Getting Started modules and samples (continued)

Module	**Sample (if any)**	**Description**
Adapter-based authentication	Exercise and code sample (single step) and Exercise and code sample (double step)	This module explains how to work with the adapter-based authentication.
Custom Authenticator and Login Module	Exercise and code sample	This module explains how to work with custom login modules and authenticators when the default ones do not suffice.
NEW: Using LDAP Login Module to authenticate users with LDAP server	Exercise and code sample	This module explains how to work with the LDAP login module to authenticate users with LDAP servers.
WebSphere LTPA-based authentication		This module explains how to work with the WebSphere LTPA-based authentication.
NEW: Device provisioning concepts		This module explains the basics of device provisioning.
Custom device provisioning	Exercise and code sample	This module explains how to create a custom provisioning that uses a certificate from an external service to authenticate a device. This module also explains how to implement a custom authenticator that connects to that service.
9. Advanced topics		
Shell development concepts	Exercise and code sample	This module explains the concepts that support the shell development and the inner applications.
Android shell development		This module explains how to develop Android applications by using shells.
iOS shell development		This module explains how to develop iOS applications by using shells.
Push notifications	Exercise and code sample	This module explains how to allow mobile device to receive messages that are pushed from a server.
NEW: SMS notifications	Exercise and code sample	This module explains how to configure mobile devices to receive notifications through SMS messages that are pushed from a server.
Integrating server-generated pages in hybrid applications	Exercise and code sample	This module explains how to remotely load dynamic content, where the code (HTML, CSS, and JavaScript) is hosted externally.
10. Moving to production		
Moving from development environment to stand-alone QA and production servers		This module explains how to move the components from the development environment into the test or production environment.

Experimental IBM Worklight offline user documentation
This document is provided as-is.
In case of issues, refer to the online documentation.

Table 1. Getting Started modules and samples (continued)

Module	Sample (if any)	Description
Reports and analytics		This module explains the BIRT reports that help collect the analytics data that are pertaining to applications and to devices that are accessing the Worklight Server.
11. Integrating with other products		
Using Rational Team Concert to build your applications	Exercise and code sample	This module explains how to develop as a team by using Rational Team Concert.
NEW: Introducing Worklight Server and Application Center on IBM PureApplication System		This module introduces how you can integrate IBM Worklight Server and Application Center with IBM PureApplication System.
NEW: Integrating IBM Tivoli Directory Server on IBM PureApplication System		This module introduces how you can integrate IBM Tivoli Directory Server with IBM PureApplication System.
Using Worklight application as a container for server-generated pages	Exercise and code sample	This module explains how to remotely load dynamic content, where the code (HTML, CSS, and JavaScript) is hosted externally.
Container for advanced pages	Exercise and code sample and Exercise and code sample (WAR)	This module complements the module "Using Worklight application as a container for server-generated pages" with advanced information about how you can remotely load dynamic content.
NEW: Integrating with SiteMinder	Exercise and code sample	This module explains how you can integrate IBM Worklight with SiteMinder.

Worklight Starter application samples

Study the Worklight Starter application samples to learn how to use IBM Worklight to create mobile applications. These samples have no associated modules.

Table 2. Worklight Starter applications

Sample	Description
Worklight Starter application	This file contains the sample code of the IBM Worklight Starter application.
Worklight Starter application with jQuery Mobile	This file contains the sample code of the IBM Worklight Starter application with jQuery Mobile.
Worklight Starter application with Sencha	This file contains the sample code of the IBM Worklight Starter application with Sencha Touch.
Worklight Starter application with Dojo Mobile	This file contains the sample code of the IBM Worklight Starter application with Dojo Mobile.

Experimental IBM Worklight online use. This document is provided as is. In case of issues, refer to the online user documentation.

JavaScript framework-based application samples

IBM Worklight provides several support materials for developing with JavaScript frameworks, such as Dojo, Dojo Mobile, and jQuery Mobile. You can study the following samples to learn how to use IBM Worklight to develop applications that are based on such frameworks. These samples have associated modules that describe them.

Table 3. JavaScript framework-based applications

Module	Sample	Description
Running the Dojo-based sample	Exercise and code sample	This module and its companion sample show how to develop an application that is based on Dojo through a basic sample application.
Running Dojo-based Mysurance end-to-end sample	Exercise and code sample	This module and its companion sample show how to develop an application that is based on Dojo through the end-to-end "MySurance" sample application.
Running Dojo Mobile-based Apache Cordova sample	Exercise and code sample	This module and its companion sample show how to develop an application that is based on Dojo Mobile through an Apache Cordova sample application.
NEW: Running jQuery Mobile-based Flight Ticket sample	Exercise and code sample	This module and its companion sample constitute an end-to-end application in the flight booking domain that is based on jQuery Mobile.

Additional resources

The following compressed files contain all the materials for the tutorials and samples:

- All IBM Worklight tutorial modules
- All IBM Worklight tutorial companion samples and application samples

Terms and conditions

Use of the IBM Worklight V5.0.6 Getting Started modules, exercises, and code samples available on this page is subject to you agreeing to the terms and conditions set forth here:

This information contains sample code provided in source code form. 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 code is written. Notwithstanding anything to the contrary, IBM PROVIDES THE SAMPLE SOURCE CODE ON AN "AS IS" BASIS AND IBM DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, SATISFACTORY QUALITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, AND ANY WARRANTY OR CONDITION OF NON-INFRINGEMENT. IBM SHALL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY OR ECONOMIC CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR OPERATION OF THE SAMPLE SOURCE CODE. IBM SHALL NOT BE LIABLE FOR LOSS OF, OR DAMAGE TO, DATA, OR FOR LOST PROFITS, BUSINESS REVENUE, GOODWILL, OR ANTICIPATED SAVINGS. IBM HAS NO OBLIGATION TO

Experimental IBM Worklight offline documentation
This document is provided as is.
In case of issues refer to the online user documentation.

PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS OR MODIFICATIONS TO THE SAMPLE SOURCE CODE.

Please review the third party licenses before using any of the resources. The third party licenses applicable to each sample are available in the `notices.txt` file included with each sample.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Chapter 2. Developing IBM Worklight applications

To develop an IBM Worklight application you must have a Worklight project, adapt for a mobile, desktop, or web environment, and authenticate.

About this task

This information is designed to help users develop and optimize applications for various channels by using the IBM Worklight Platform. It is intended for web developers who know Ajax and are familiar with application development.

It also covers server-side development topics, such as integration and push notifications.

The Worklight Platform provides a framework that enables the development, optimization, integration, and management of secure applications that run on high-end smartphones and other consumer environments. This framework provides the following features:

- Guidelines and design patterns that promote compatibility across multiple consumer environments.
- Automatic packaging and provisioning of application resources to multiple consumer environments.
- A flexible UI optimization and globalization scheme.
- Tools that provide uniform access to back-end enterprise data, processes, and transactions.
- Uniform persistence.
- A uniform personalization model.
- A flexible authentication model and automatic application protection from web attacks.

Worklight does not introduce a proprietary programming language or model that users have to learn. You can develop apps using HTML5, CSS3, and JavaScript. You can optionally write native code (Java or Objective-C) if you need to do so and Worklight provides an SDK that includes libraries that you can access from native code.

Starting with IBM Worklight projects, applications, environments, and skins

With Worklight Studio, you can develop mobile applications within projects, build your applications for different environments, and create skins for specific devices.

This collection of topics introduces how you can start your mobile application development with IBM Worklight Studio.

- It introduces the concepts of projects, applications, environments, and skins.
- It gives the required steps for you to create a project and applications.
- It describes the content of a project and of applications.

Experimental IBM Worklight offline user documentation.
This document is provided "as-is".
In case of issues refer to the online user documentation.

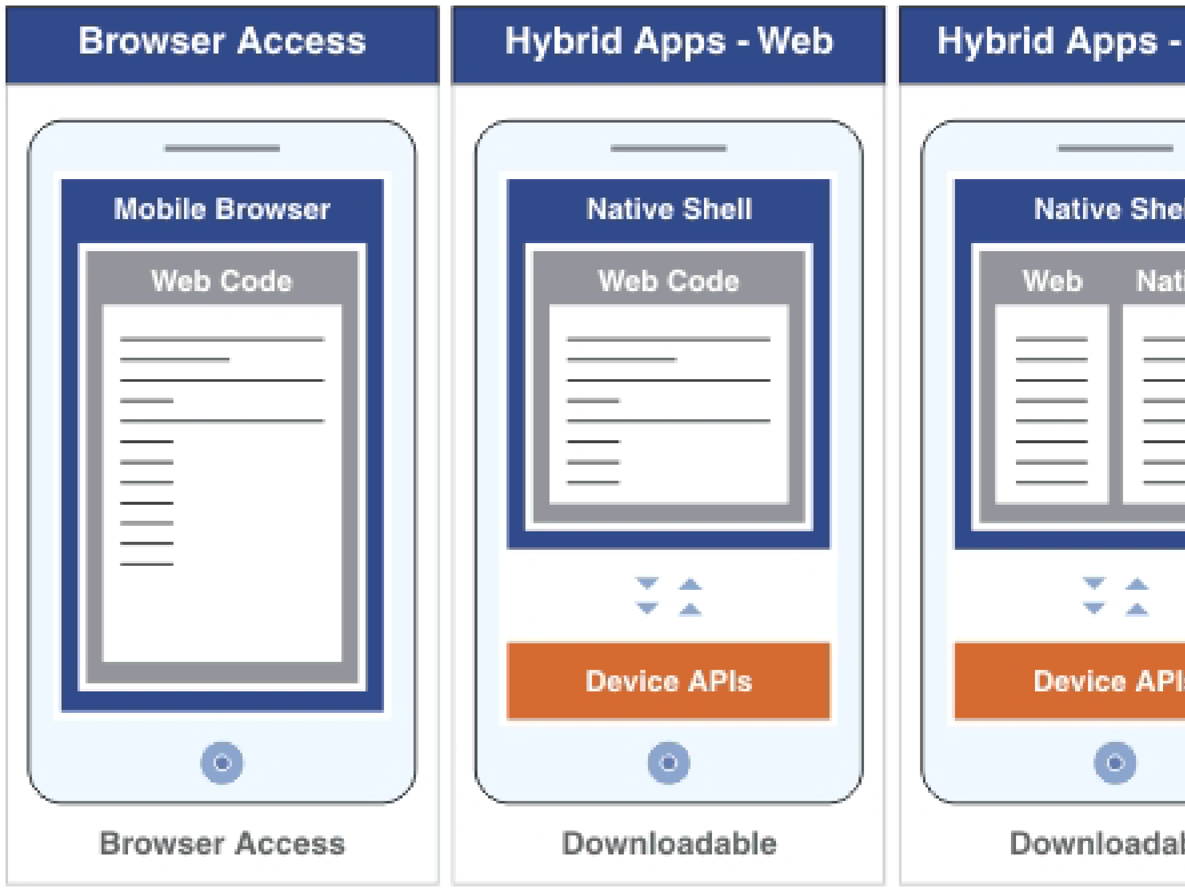
Overview: IBM Worklight projects, applications, environments, and skins

With Worklight Studio, you can develop mobile applications within projects, build your applications for different environments, and create skins for specific devices.

IBM Worklight applications

With IBM Worklight, you can develop mobile applications by using any of four different approaches:

- Web Applications: they are written entirely in HTML5, CSS, and JavaScript code. Web applications are executed by the mobile browser and are cross-platform by default.
- Hybrid Applications (Web): the source code of the application consists of web code that is executed within a native container (which is provided by IBM Worklight) and consists of native libraries.
- Hybrid Applications (Mixed): the developer augments the web code with native language to create unique features and access native APIs that are not available in JavaScript.
- Native Applications: this type of application is platform-specific and requires expertise and knowledge of the platform.



By using one or all of these approaches to implement mobile applications, you can:

- Create mobile applications that are designed specifically for the needs of their organization.
- Use multiple distribution channels such as public and private enterprise application stores.
- Manage the growing portfolio of deployed apps and integration adapters in a secure and centralized manner.

IBM Worklight environments

You can build your mobile applications for different environments, such as:
- Mobile environments, which include iPhone, iPad, Android phones and tablets, BlackBerry 6 and 7, BlackBerry 10, Windows Phone 7.5 and Window Phone 8.
- Desktop environments, which include Adobe AIR and Windows 8.
- Web environments, which include Mobile web app and Desktop Browser web page.

There is a difference between the Mobile web app environment and the Desktop Browser web page environment.
- Mobile web apps are only used in a mobile device browser. Choose the Mobile web app environment when you want your users to surf to your application by using their mobile device.
- Desktop browser web pages are used only in a desktop web browser. With the Desktop Browser web page environment, you can develop an application that you then embed inside your website, but this application is not meant for use in a mobile device.

Note: You cannot combine the Mobile web app environment and the Desktop browser web page environment in the same application.
If your web application is not based on Worklight, you must first port it to Worklight. If your web application is based on Worklight, you can add the Desktop Browser web page environment to your existing project.

IBM Worklight projects

To develop your mobile applications with IBM Worklight, you must first create a project in IBM Worklight Studio.

A project in Worklight Studio is a place for you to develop one or several mobile applications, which you can build for different environments.

In your project, when you create an application, you have a main application folder, in which you can find several subfolders:
- A `common` folder, for you to store the code that is shared between all environments, such as HTML, CSS, or JavaScript code.
- One folder for each environment that is supported by the application, and where you store the code that is specific to this environment, such as Java code for Android or Objective-C code for iOS.
- An `adapter` folder, for you to store the code of the adapters that your application requires to collect data from back-end systems.

Within your project, you can create the graphical user interface of your mobile application by using the Rich Page Editor. The Rich Page Editor is a WYSIWYG editor in Worklight Studio.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Note: If you use non-Latin characters in your application, you must make sure that your Eclipse editor uses UTF-8 encoding. To set the Eclipse text file encoding to UTF-8:

1. In Worklight Studio, go to **Window** > **Preferences** > **General** > **Workspace**.
2. In **Text file encoding**, select **Other**, and select **UTF-8** from the list.

When the application is finished, you can test it with the Mobile Browser Simulator in Worklight Studio. However, you cannot test native code with Worklight Studio. To test native code, you must test it with a real device or with the development kit of the appropriate environment. To test your application:

1. Build and deploy your application: Worklight Studio creates the project with your native code that you can then view and update.
2. Test it with the Mobile Browser Simulator, which emulates the target device, or with a real device.

IBM Worklight skins

Different types of devices exist for a same environment. If you want to write a piece of code that is specific to a certain device, you must create a skin. Skins are subvariants of an environment and they provide support for multiple form factors in a single executable file for devices of the same OS family. Skins are packaged together in one app. At run time, only the skin that corresponds to the target device is applied.

Creating IBM Worklight projects

You use Worklight Studio to create an IBM Worklight project.

About this task

With Worklight Studio, you create an IBM Worklight project as a place where you develop your apps. When you create an IBM Worklight project, you create a first app in it. This first app can be of the following types:

- *Hybrid application*: A Hybrid application can target multiple environments. You can write it primarily in HTML5, CSS, and JavaScript. It can access device capabilities by using the IBM Worklight JavaScript API. You can also extend it with native code.
- *Inner application*: An Inner application contains the HTML, CSS, and JavaScript parts that run within a Shell component. Before you can deploy this application, you must package it within a shell component to create a full hybrid application.
- *Shell component*: A Shell component provides custom native capabilities and security features that an Inner application can use.
- *Native application*: A Native application targets a specific environment, and can use the IBM Worklight API for integration, security, and application management.

After you created an IBM Worklight project, you can later add further apps to it.

Procedure

To create an IBM Worklight project and a first app in it:

1. Select **File** > **New** > **Worklight Project**.
2. In the **Project Name** field, enter a name for your new project.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

3. From the list of project templates, select the template that applies to the first application in your Worklight project:

Option	Description
`Hybrid Application`	To create a Worklight project with an initial hybrid application.
`Inner Application`	To create a Worklight project with an initial inner application and point to a built shell component
`Shell Component`	To create a Worklight project with an initial shell component application
`Native Application`	To create a Worklight project with an initial Native application

4. Optional: Select any of the following options to add the corresponding support to the application:

Option	Description
Add jQuery Mobile	To add jQuery Mobile support to the application. You must identify the directory where the required files for jQuery Mobile are located.
Add Sencha Touch	To add Sencha Touch support to the application. You must identify the directory where the required files for Sencha Touch are located.
Add Dojo Toolkit	To add the Dojo facet and Dojo support to the application. When you build a mobile web application, Dojo is included to create the native application, such as an iPhone or Android application. **Note:** If you intend to add a Windows Phone 8 environment, note that Dojo Mobile is not yet supported on Windows Phone 8.

Anatomy of an IBM Worklight Project

The file structure of an IBM Worklight project helps you organize the code that is required for your apps.

When you develop mobile apps with the IBM Worklight platform, all development assets including source code, libraries, and resources are placed in an IBM Worklight project folder.

A Worklight project has the following structure:

`<project-name>`		Root project folder
	`adapters`	Source code for all adapters belonging to the project
	`apps`	Source code for all applications belonging to the project

Experimental IBM Worklight offline user documentation.
This document is provided as is.
In case of issues refer to the online user documentation.

	bin		Artifacts resulting from building adapters, apps, and server-side configuration and libraries
	components		Source code for all shell components belonging to the project
	dojo		Source code of the Dojo JavaScript framework, if installed as part of the IBM Worklight Studio
	server		
		conf	IBM Worklight Server configuration files, such as `worklight.properties` and `authenticationConfig.xml`
		java	Java code that must be compiled and packaged into jar files deployable to the IBM Worklight Server
		lib	Pre-compiled libraries that must be deployed to the IBM Worklight Server

Experimental IBM Worklight offline user documentation.
This document is provided "as is".
In case of issue, refer to the online user documentation.

Initialization options

The `initOptions.js` file is included in the project template. It is used to initialize the Worklight JavaScript framework. It contains a number of tailoring options, which you can use to change the behaviour of the JavaScript framework. These options are commented out in the supplied file. To use them, uncomment and modify the appropriate lines.

The `initOptions.js` file calls WL.Client.init, passing an `options` object that includes any values you have overridden.

Content of the `dojo` folder

If you have installed the Dojo JavaScript framework, the `dojo` folder contains several layers of JavaScript files that provide the Dojo widgets that you can use in your apps.

When you create your app, your pages by default link only to the JavaScript files that are required to use the `dojo.mobile` widgets. To use widgets that are not defined in `dojo.mobile`, such as the widgets defined in `dojox`, you must explicitly add the required links to the proper JavaScript layers in your app pages.

If you do not add these links, you cannot have any runtime visualization of the result when using the Mobile Browser Simulator and the Mobile Browser Simulator might even not properly display other contents.

For example, to work with the `dojox.charting.widget.Chart` widget, your page must link to the `charting-layer.js` layer that defines the `dojox.charting.widget.Chart`. Your page must also link to the `graphics-layer.js` layer that the`charting-layer.js` layer requires. You must therefore:

1. Add the following links to your page:
   ```
   <script type="text/javascript" src="dojo/charting-layer.js"></script>
   <script type="text/javascript" src="dojo/graphics-layer.js"></script>
   ```
2. Edit the `build-dojo.xml` file in your project to ensure that the JavaScript files are part of your mobile resources:
   a. Search for the `target` section that copies the mobile resources: it starts with this tag: `<target name="-copy-mobile-resources" if="mobile.mgt">`.
   b. In this section, add the following lines next to `mobile layers`:
      ```
      <include name="dojo/charting-layer.js.compressed.js">
      <include name="dojo/graphics-layer.js.compressed.js">
      ```

Creating the client side of an IBM Worklight application

You use Worklight Studio to create the client side of an IBM Worklight application.

In Worklight Studio, you have two methods to create the client side of an IBM Worklight application:
- Use an existing Worklight project, and create your application in it, as described in “Creating an application in an IBM Worklight project” on page 46.
- Create a Worklight project, and your application in it as its first application, as described in “Creating IBM Worklight projects” on page 42

After you create your Worklight application, you can develop its code by using different APIs:
- JavaScript client-side API for hybrid apps
- Objective-C client-side API for native iOS apps
- Java client-side API for native Android apps
- Java client-side API for Java Platform, Micro Edition (Java ME) apps

You can also use your own custom libraries or third-party libraries when you create mobile applications in Worklight Studio.

You can find guidelines on how to develop your applications in the following topics:
- “Development guidelines for mobile environments” on page 61
- “Development guidelines for desktop and web environments” on page 79
- “Development guidelines for using native API” on page 84

JavaScript client-side API for hybrid apps

With the JavaScript client-side API, you can develop hybrid applications that target all environments. You can use the capabilities of the IBM Worklight runtime client API for mobile applications, desktop, and web to develop your applications.

Experimental IBM Worklight offline user documentation
This document is provided [illegible]
In case of issues refer to the online user documentation.

For more information, see "JavaScript client-side API" on page 173.

Objective-C client-side API for native iOS apps

IBM Worklight provides the IBM Worklight Objective-C client-side API that you can use to develop native iOS applications. This API provides 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.

For more information, see "Objective-C client-side API for native iOS apps" on page 271.

Java client-side API for native Android apps

IBM Worklight provides the IBM Worklight Java client-side API that you can use to develop native Android applications. This API provides 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.

For more information, see "Java client-side API for native Android apps" on page 271.

Java client-side API for Java ME apps

IBM Worklight provides the IBM Worklight Java client-side API that you can use to develop native Java ME applications. This API provides two main capabilities:

- Calling back-end services for retrieving data and performing back-end transactions.
- Writing custom log lines for reporting and auditing purposes.

For more information, see "Java client-side API for Java ME apps" on page 271.

Creating an application in an IBM Worklight project

With Worklight Studio, you can create different types of applications within an existing IBM Worklight project.

About this task

With Worklight Studio, you can create and develop an IBM Worklight application in an existing IBM Worklight project.

Procedure

1. From the **Worklight** menu, select the type of application you want to create:
   - **Worklight Hybrid Application**
   - **Worklight Inner Application**
   - **Worklight Native API**

Experimental IBM Worklight offline user documentation
This document is provided "as-is".
In case of issues, refer to the online user documentation.

- **Worklight Shell Component**

A dialog opens that depends on the type of application that you selected. Depending on the selected type of application, set the properties of your application, as described next.

2. Hybrid Application:
   a. In the field **Project name**, select your existing project.
   b. In the field **Application name**, set the name of your application.
   c. Select the check-boxes that correspond to the layers that you need (jQuery Mobile, Sensa Touch, Dojo Mobile). Note: If you intend to add a Windows Phone 8 environment, note that Dojo Mobile is not yet supported on Windows Phone 8.
3. Inner Application:
   a. In the field **Project name**, select your existing project.
   b. In the field **Application name**, set the name of your application.
   c. In the field **Shell archive name**, set the path of your Shell archive file. The path can be either absolute or relative in case Shell archive exists within your project.
4. Native API:
   a. In the field **Project name**, select your existing project.
   b. In the field **Application name**, set the name of your application.
   c. In the field **Environment**, select the environment that you need: Android, iOS, or Java ME.
5. Shell Component:
   a. In the field **Project name**, select your existing project.
   b. In the field **Component name**, set the name of your component.
   c. Select the check-boxes that correspond to the layers that you need (jQuery Mobile, Sensa Touch, Dojo Mobile). Note: If you intend to add a Windows Phone 8 environment, note that Dojo Mobile is not yet supported on Windows Phone 8.
6. Click **Finish** to save your choices. An application of the type of application that you selected is now visible in your Worklight project.

Anatomy of an IBM Worklight Application

This collection of topics describes the files within a Worklight Application

By using the IBM Worklight Platform, you can write applications by using web technologies or native technologies, or combine both types of technology in a single app. All client-side application resources, both web and native, must be located under a common file folder with a predefined structure. The IBM Worklight Studio builds these resources into various targets, depending on the environments supported by the application.

The application folder

The application folder contains all application resources.

The folder has the following structure:

`<app-name>`	Main application folder

Experimental IBM Worklight offline user documentation. This document is provided "as-is". In case of issues refer to the online user documentation.

	`common`		Application resources common to all environments
		`css`	Style sheets to define the application view
		`images`	Thumbnail image and default icon
		`js`	JavaScript files
		`<app-name>.html`	An HTML5 file that contains the application skeleton
	`android`		Web and native resources specific to Android
	`blackberry10`		Web and native resources specific to BlackBerry 10
	`blackberry`		Web and native resources specific to BlackBerry 6 and 7
	`ipad`		Web and native resources specific to iPad
	`iphone`		Web and native resources specific to iPhone
	`windowsphone8`		Web and native resources specific to Windows Phone 8
	`windowsphone`		Web and native resources specific to Windows Phone 7.5
	`air`		Resources specific to Air
	`dashboard`		Resources specific to OS X dashboard
	`desktopbrowser`		Resources specific to desk top browsers
	`facebook`		Resources specific to Facebook. The use of Facebook is deprecated in Worklight version 5.0.5. Support might be removed in any future version.
	`igoogle`		Resources specific to iGoogle. The use of iGoogle is deprecated in Worklight version 5.0.5. Support might be removed in any future version.

Experimental IBM Worklight offline user documentation.
This document is provided "as is".
In case of issues, refer to the online user documentation.

	mobilewebapp	Web resources specific to mobile web applications
	vista	Resources specific to Windows 7 and Vista
	windows8	Resources specific to Windows 8
	legal	License documents for the application or third-party software used in the application
	application-descriptor.xml	

Application resources

You must provide various types of resources if you are to create applications that can run in multiple environments.

You must provide the following resources to create applications that can run in multiple environments. IBM Worklight automatically generates any missing resources that are not supplied. However, for production quality, you must provide all resources that are required by the environments in which the application runs.

Application descriptor

The application descriptor is a mandatory XML file that contains application metadata, and is stored in the root directory of the app. The file is automatically generated by Worklight Studio when you create an application, and can then be manually edited to add custom properties.

Main file

The main file is an HTML5 file that contains the application skeleton. This file loads all the web resources (scripts and style sheets) necessary to define the general components of the application, and to hook to required document events. This file is in the `\common` folder of the `app` directory and optionally in the `optimization` and `skin` folders.

The main file contains a `<body>` tag. This tag must have an `id` attribute that is set to `content`. If you change this value, the application environment does not initialize correctly.

Style sheets

The app code can include CSS files to define the application view. Style sheets are placed under the `\common` folder (normally under `\common\css`) and optionally in the `optimization` and `skin` folders.

Scripts

The app code can include JavaScript files that implement interactive user interface components, business logic and back-end query integration, and a message dictionary for globalization purposes. Scripts are placed under the `\common` folder (normally under `\common\js`) and optionally in the `optimization` and `skin` folders.

Experimental IBM Worklight online user documentation. This document is provided "as is". In case of issues refer to the online user documentation.

Thumbnail image

The thumbnail image provides a graphical identification for the application. It must be a square image, preferably of size 128 by 128 pixels. It is used to identify the app in the IBM Worklight catalog.

Worklight Studio creates a default thumbnail image when the app is created. You can override this default image (using the same file name) with a replacement image that matches your application. The file is in the `\common\images` folder of the app.

Splash image

The splash image applies for mobile environments and Windows 8 apps. The splash image (or *splash screen*) is displayed while the application is being initialized. It must be in the exact dimensions of the app.

Worklight Studio creates a default splash image when you create an application environment. These **default** images are stored in the following locations:

- For Apple iOS platforms, the default splash images are stored:
  - For iPhone, under `iphone\native\Resources`
  - For iPad, under `ipad\native\Resources`

  The file names of the default splash images are as follows, and vary according to iOS version and target device:
  - For iPhone Non-Retina display (iOS6.1 and earlier): `Default~iphone.png` 320 by 480 pixels
  - For iPhone Retina display (iOS6.1 and earlier): `Default@2x~iphone.png` 640 by 960 pixels
  - For iPhone 4-inch Retina display (iOS6.1 and earlier): `Default568h@2x~iphone.png` 640 by 1136 pixels
  - For iPhone Retina display (iOS7): `Default@2x~iphone.png` 640 by 960 pixels
  - For iPhone 4-inch Retina display (iOS7): `Default568h@2x~iphone.png` 640 by 1136 pixels
  - For iPad (iOS6.1 and earlier): `Default-Portrait~ipad.png` 768 by 1004 pixels
  - For iPad Retina display (iOS6.1 and earlier): `Default-Portrait@2x~ipad.png` 1536 by 2008 pixels
  - For iPad (iOS6.1 and earlier): `Default-Landscape~ipad.png` 1024 by 748 pixels
  - For iPad Retina display (iOS6.1 and earlier): `Default-Landscape@2x~ipad.png` 2048 by 1496 pixels
  - For iPad (iOS7): `Default-Portrait~ipad.png` 768 by 1004 pixels
  - For iPad Retina display (iOS7): `Default-Portrait@2x~ipad.png` 1536 by 2008 pixels
  - For iPad (iOS7): `Default-Landscape~ipad.png` 1024 by 748 pixels
  - For iPad Retina display (iOS7): `Default-Landscape@2x~ipad.png` 2048 by 1496 pixels
- For Android platforms, the file name of the default splash image is `splash.9.png`; it is stored:
  - For all resolutions, under `android\native\res\drawable`
- For BlackBerry 10, under `blackberry10\native`. The file must be in `.png` format and there are four different splash screen sizes:
  - splash 1024 pixels width by 600 pixels height: `splash-1024x600.png`

Experimental IBM Worklight documentation. This document is provided "as-is". In case of issue, refer to the online user documentation.

  - splash 1280 pixels width by 768 pixels height: `splash-1280x768.png`
  - splash 600 pixels width by 1024 pixels height: `splash-600x1024.png`
  - splash 768 pixels width by 1280 pixels height: `splash-768x1280.png`
- For BlackBerry 6 and 7, the file name of the splash image is `splash.png`, stored under `blackberry\native`.
- For Windows Phone 8, the file name of the splash image is `SplashScreenImage.jpg`, stored under `windowsphone8\native`. This file must be in `.jpg` format, with a width of 768 pixels and height of 1280 pixels.
- For Windows 8, the file name of the splash image is `splashscreen.png`, stored under `windows8\native\images`. This file must be in `.png` format, with a width of 620 pixels and height of 300 pixels.

Adding a custom splash image

You can override the default images that are created by Worklight Studio with a splash image that matches your application.

The procedures for doing this differ, depending on the target platform. But in all cases, your custom splash image must match the size of the default splash image you are replacing, and must use the same file name.

- For Apple iOS platforms:
  - There are two ways of creating a custom splash image:
    1. Replace the default image in `ipad\native\Resources` (or `iphone\native\Resources`), **OR**
    2. Add the new (replacement) image to `ipad\nativeResources\Resources` (or `iphone\nativeResources\Resources`).
    3. Run **Build All and Deploy** for the project.

    The second method (step 2) is preferable because it does not delete any files from the `native` directory, which is often not backed up in a source code control system. When you add your image to the `nativeResources` directory, it is copied to the `native` directory during the build. The replacement splash image must not be placed in any folder other than `Resources`.
- For Android:
  - There are two ways of creating a custom splash image:
    1. Replace the default image in `android\native\res\drawable`, **OR**
    2. Add the new (replacement) image to `android\nativeResources\res\drawable`.
    3. Run **Build All and Deploy** for the project.

    The second method (step 2) is preferable because it does not delete any files from the `native` directory, which is often not backed up in a source code control system. When you add your image to the `nativeResources` directory, it is copied to the `native` directory during the build. The replacement splash image must not be placed in any folder inside the `res` folder other than `drawable`.
- For BlackBerry 10:
  - There are two ways of creating a custom splash image:
    1. Replace the default image in `blackberry10\native` (or `iphone\native\Resources`), **OR**
    2. Add the new (replacement) image to `blackberry10\nativeResources\www`.
    3. Run **Build All and Deploy** for the project.

Experimental IBM Worklight offline user documentation.
This document is provided as is.
In case of issues, refer to the online user documentation.

The second method (step 2) is preferable because it does not delete any files from the `native` directory, which is often not backed up in a source code control system. When you add your image to the `nativeResources` directory, it is copied to the `native` directory during the build.

- For BlackBerry 6 and 7:
  1. Replace the default image in `blackberry\native`. If the original splash image is not backed up in a source code control system, it is advisable to rename or back up the original image first.
  2. Run **Build All and Deploy** for the project.
- For Windows Phone 8:
  - There are two ways of creating a custom splash image:
    1. Replace the default image in `windowsphone8\native`, **OR**
    2. Add the new (replacement) image to `windowsphone8\nativeResouces`.
    3. Run **Build All and Deploy** for the project.

    The second method (step 2) is preferable because it does not delete any files from the `native` directory, which is often not backed up in a source code control system. When you add your image to the `nativeResources` directory, it is copied to the `native` directory during the build.
- For Windows 8:
  1. Replace the default image in `windows8\native\images`. If the original splash image is not backed up in a source code control system, it is advisable to rename or back up the original image first.
  2. Run **Build All and Deploy** for the project.

Application icons

Worklight Studio creates default application icons when you create the app. You can override them with images that match your application. For Android, iPad, and iPhone, put your replacement icons (using the same file names, except as noted with an asterisk * below) in the location indicated by the `Location of overriding icon` column in the following table.

The following table summarizes the sizes and location of each application icon.

Table 4. Application icons

Environment	File name	Description	Location of default icon	Location of overriding icon

Adobe AIR	`icon16x16.png`
	`icon32x32.png`
	`icon48x48.png`
	`icon128x128.png`
		Application icons of various sizes that are attached to the AIR version of the application.
		
		The dimensions of each icon are specified in its name.
			`air\images`	

Experimental "IBM Worklight" offline user documentation.
This document is provided "as is".
In case of issues, refer to the online user documentation.

Table 4. Application icons (continued)

Environment	File name	Description	Location of default icon	Location of overriding icon
Android	`icon.png`	An icon that is displayed on the device springboard. You can provide a different icon for each device density that you want to support.	`android\native\res\drawable`	`⌂\android\nativeResources\res\drawable` or `android\nativeResources\res\drawable-l` *`-hdpi -or other options`*
BlackBerry 10	`icon.png`	An icon that is displayed on the device. Its dimensions are 114 by 114 pixels. Application icons: for best practices on creating icons see https://developer.blackberry.com/devzone/design/application_icons.htmlnati	`blackberry10\native\www`	`blackberry10\nativeResources\www`
BlackBerry 6 and 7	`icon.png`	An icon that is displayed on the device. Its dimensions are 80 by 80 pixels.	`blackberry\native`	
iPad	`icon-`*`xxxx`*`.png` * Filename varies by size and target device. Exact file name may change as long as it is listed in the `plist` file.	An icon that is displayed on the device springboard. Size depends on iOS version and target device. iOS6.1 and earlier: • Non-Retina display: 72 by 72 pixels • Retina display: 144 by 144 pixels iOS7: • Non-Retina display: 76 by 76 pixels • Retina display: 152 by 152 pixels	`ipad\native\resources`	`\ipad\nativeResources\Resources`

Experimental IBM Worklight offline user documentation.
This document is provided "as is".
In case of issues, refer to the online user documentation.

Table 4. Application icons (continued)

Environment	File name	Description	Location of default icon	Location of overriding icon
iPhone	`icon-xxxx.png` * Filename varies by size and target device. Exact file name may change as long as it is listed in the `plist` file.	An icon that is displayed on the device springboard. Size depends on iOS version and target device. iOS6.1 and earlier: • Non-retina display: 57 by 57 pixels • Retina display: 114 by 114 pixels iOS7: • 120 by 120 pixels	`iphone\native\resources`	`\iphone\nativeResources\Resources`

Windows Phone 8	`Background.png`
	
	`ApplicationIcon.png`
		Both icons are used to identify the application.
		
		`Background.png` is displayed on the device home screen, and must be 300 by 300 pixels.
		
		`ApplicationIcon.png` is displayed in the list of applications, and must be 100 by 100 pixels.
			`windowsphone8\native`	

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Table 4. Application icons (continued)

Environment	File name	Description	Location of default icon	Location of overriding icon

Windows 8	`storelogo.png`
	
	`logo.png`
	
	`smalllogo.png`
		All icons are used to identify the application.
		
		`storelogo.png` is the image the Windows Store uses when it displays the app listing in search results and with the app description in the listing page. The image must be 50 by 50 pixels.
		
		`logo.png` represents the square tile image of the app in the Start screen. The image must be 150 by 150 pixels.
		
		`smalllogo.png` is displayed with the app display name in search results on the Start screen. `smalllogo.png` is also used in the list of searchable apps and when the Start page is zoomed out. The image must be 30 by 30 pixels.
			`windows8\native\images`	

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

The application descriptor

The application descriptor is a metadata file that is used to define various aspects of the application. It is located in the application root directory.

General structure

The application descriptor is a metadata file that is used to define various aspects of the application. It is located in the application root directory and has the name `application-descriptor.xml`.

The following example shows the format of the file:

```
<?xml version="1.0" encoding="UTF-8"?>
<application id="fcb" platformVersion="5.0">
xmlns="http://www.example.com/application-descriptor" xmlns:xsi=http://www.w3.org/2001/XMLSchema-ins
xsi:schemaLocation="http://www.example.com/application-descriptor
../../../../../gadgets/application-descriptor/src/main/resources/schema/application-descriptor.xsd">
```

The `<application>` element is the root element of the descriptor. It has two mandatory attributes:

id
Contains the ID of the application. The ID must be identical to the application folder name. It must be an alphanumeric string that starts with a letter. It can also contain underscore ("_") characters. It must not be a reserved word in JavaScript.

platformVersion
Contains the version of the IBM Worklight Platform on which the app was developed.

```
<displayName>First Country Bank</displayName>
<description>Conveniently and securely manage your checking, savings, and credit card accounts using
```

The `<displayName>` and `<description>` elements contain the name and description of the application. They are displayed in the IBM Worklight Console and are copied to the descriptor files of various web and desktop environments.

```
<author>
<name>ACME</name>
<email> info@acme.com </email>
<homepage> acme.com </homepage>
<copyright> (C) ACME 2011 </copyright>
</author>
```

You can use the `<author>` element and its subelements to provide information about the application author. This data is copied to the descriptor files of the web and desktop environments that require it.

```
<height>410</height>
<width>264</width>
```

The `<height>` element is used to determine the height of the application on iGoogle and desktop environments.

The use of iGoogle is deprecated in IBM Worklight version 5.0.5. Support might be removed in any future version.

The `<width>` element is used to set the width of the application on desktop environments.

```
<mainFile>fcb.html</mainFile>
<thumbnailImage>common/images/thumbnail.png</thumbnailImage>
```

The `<mainFile>` element contains the name of the main HTML file of the application.

The `<thumbnailImage>` element contains the path to and the name of the thumbnail image for the application. The path is relative to the main application folder.

```
<worklightServerRootURL> https://www.acme.com/mobile-services </worklightServerRootURL>
```

The `<worklightServerRootURL>` is used to define the URL for accessing the IBM Worklight Server for mobile and desktop apps. It can be any URL of the form: *`protocol`*`://`*`domain`*`[:`*`port`*`][/`*`path`*`]`.

Experimental IBM Worklight offline user documentation.
This document is provided "as is".
In case of issues, refer to the online user documentation.

Note: The <worklightServerRootURL> replaces the former <worklightRootURL> element. <worklightRootURL> is deprecated and might not be supported in any future release.

```
<smsGateway id="kannelgw"/>
```

The <smsGateway> element defines the SMS gateway to be used for SMS Push Notifications. It has one mandatory attribute:

id Contains the ID of the SMS gateway. The ID must match one of the gateway IDs defined in the SMSConfig.xml file.

```
<iphone version="1.0" />
<android version="1.0" />
<blackberry10 version="1.0" />
<blackberry version="1.0" />
<windowsPhone8 version="1.0">
  <uuid>87e096eb-6882-4cef-9f66-e68769de3926</uuid>
</windowsPhone8>
<windowsPhone version="1.0">
  <uuid>62a2a2cf-0092-448e-8e7b-130687ca2938</uuid>
</windowsPhone>
<windows8 version="1.0">
  <certificate PFXFilePath="Path to certificate file" password="certificate password"/>
  <uuid>556a98a3-63fb-4602-827c-0b6bd9d00490</uuid>
</windows8>
<ipad version="1.0" />
<mobileWebApp version="1.0" />
<vista version="1.0" />
<dashboard version="1.0" />
<air version="1.0" />
```

Each environment on which the application can run must be declared with a dedicated XML element. Each such element has one mandatory attribute, **version**. The value of this version is a string of the form x.y, where *x* and *y* are digits (0-9).

- For mobile apps, the version is exposed to users who download the app from the app store or market.
- For desktop apps, the version determines whether the IBM Worklight Server automatically downloads a new version of the app to the user's desktop
- For web apps, the value of the version has no functional meaning and is available for documentation purposes only.

```
<iphone version="1.0" bundleId="com.mycompany.myapp"> (or <ipad>)
<pushSender password="${push.apns.senderPassword}"/>
<worklightSettings include="true"/>
<security> ... </security>
</iphone>
```

In the <iphone> and <ipad> elements, you must provide the bundle ID of the application in the **bundleId** attribute. Each time the IBM Worklight builder builds your application, it copies the value of this attribute to the appropriate native configuration file in the Xcode project of the application. Do not modify this value directly in the native configuration file as it is overridden by the builder with the value you indicate in this attribute.

For iOS apps that use the Apple Push Notification Service (APNS), use the <pushSender> element to define the password to the SSL certificate that encrypts the communication link with APNS. The **password** attribute can refer to a property in the worklight.properties file and can thus be encrypted.

The app user can use the IBM Worklight settings screen to change the address of the IBM Worklight Server with which the app communicates. To enable it for the

Experimental IBM Worklight offline user documentation.
This document is provided "as is".
In case of issues refer to the online user documentation.

app, specify the `<worklightSettings>` element. When enabled, the settings screen is accessible by using the settings app on the iOS device.

See "The <security> element" on page 60 for details of this element.

```
<android version="1.0" sharedUserId="com.mycompany">
<pushSender key="AIzaSyDcSz7OvxQwr7XKg_0UdOaNJz0pYXuaS_c" senderId="54385266031"/>
<worklightSettings include="true"/>
<security> ... </security>
</android>
```

The **`sharedUserId`** attribute is optional; it is required only when device provisioning is activated on the application by specifying the `<authentication>` element. **`sharedUserId`** allows multiple applications with the same value for this attribute to access the same keystore item on the device. The applications can thus use the same secure device ID assigned to the device by the IBM Worklight app.

Note: : Android apps that have the same **`sharedUserId`** but are signed with a different certificate cannot be installed on the same device.

For Android apps that use Google Cloud Messaging (GCM), use the `<pushSender>` element to define the connectivity details to GCM. The **`key`** is the GCM API key, and the **`senderId`** is the GCM Project Number. For more information about GCM API key and GCM Project Number, see http://developer.android.com/google/gcm/gs.html#gcm-service.

The app user can use the IBM Worklight settings screen to change the address of the IBM Worklight Server with which the app communicates. To include it in the app, specify the `<worklightSettings>` element. When the screen is included in the app, a menu item is automatically appended to the options menu of the app. Users can tap this menu item to reach the screen.

See "The <security> element" on page 60 for details of this element.

```
<windowsPhone8 version="1.0">
<uuid>87e096eb-6882-4cef-9f66-e68769de3926</uuid>
<pushSender/>
<allowedDomainsForRemoteImages>
 <domain>http://icons.aniboom.com</domain>
 <domain>http://media-cache-ec2.pinterest.com</domain>
</allowedDomainsForRemoteImages>
</windowsPhone8>
```

The `<windowsPhone8>` element has three subelements:

- The `<uuid>` subelement is used to uniquely identify a Windows Phone 8 application on the device. It is automatically generated by the IBM Worklight Studio when you create the Windows Phone 8 environment for the application.
- For Windows Phone 8 apps that use the Microsoft Push Notification Service (MPNS), use the `<pushSender>` subelement to indicate that the app is a "pushable" application, that is, it subscribes to event sources and receives push notifications.
- The `<allowedDomainsForRemoteImages>` subelement is used to enable the application tile to access remote resources. Use subelement `<domain>` within `<allowedDomainsForRemoteImages>` to define the list of allowed remote domains from which to access remote images. Each domain in the list is limited to 256 characters.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

Note: The `<allowedDomainsForRemoteImages>` subelement cannot be added to the application descriptor by using the Design editor. You must use the Source editor instead.

```
<windowsPhone version="1.0">
<uuid>62a2a2cf-0092-448e-8e7b-130687ca2938</uuid>
</windowsPhone>
```

The `<uuid>` element is used to uniquely identify a Windows Phone 7.5 application on the device. It is automatically generated by the IBM Worklight Studio when you create the Windows Phone 7.5 environment for the application.

```
<windows8 version="1.0">
<certificate PFXFilePath="Path to certificate file" password="certificate password"/>
<uuid>556a98a3-63fb-4602-827c-0b6bd9d00490</uuid>
</windows8>
```

The `<windows8>` element contains the following subelements:

`<certificate>`
Use the `<certificate>` subelement to sign the Windows 8 application before you publish it. See "Signing Windows 8 apps" on page 82 for more details.

`<uuid>` Use the `<uuid>` subelement to uniquely identify a Windows 8 application. It is automatically generated by the IBM Worklight Studio when you create the Windows 8 environment for the application.

```
<mobileDeviceSSO join="true" />
```

When this element is specified, device SSO is enabled for the application. Thus, when a session requires authentication in a realm and there is already an active session from the same device authenticated in that realm, the authentication details from the existing session are copied to the new session. The user experience implications are that the user does not have to reauthenticate when starting the new session.

```
<air version="1.0" showOnTaskbar="always">
<certificate password="password" PFXFilePath="path-to-pfx"/>
</air>
```

The optional `<air>` element has the following structure:

- The **`showOnTaskbar`** attribute determines behavior of the AIR application on the taskbar. See "Specifying the application taskbar for Adobe AIR applications" on page 79 for more details.
- Use the `<certificate>` element to sign the AIR application before you publish it. See "Signing Adobe AIR applications" on page 82 for more details.

```
<facebook version="1.0" loginDisplayType="type" />
<igoogle version="1.0" loginDisplayType="type" />
```

The use of the `<facebook>` and `<igoogle>` elements is deprecated in IBM Worklight version 5.0.5. Support might be removed in any future version.

The `<facebook>` and `<igoogle>` elements receive a **`loginDisplayType`** attribute, which can be used to specify how the app login screen is displayed. *type* can be one of the following values:

`popup` To log in to a separate browser window.

`embedded`
To embed the login form within the application frame.

Experimental IBM Worklight offline user documentation. This document is provided "as-is". In case of issue refer to the online user documentation.

```
<loginPopupHeight> Height in pixels </loginPopupHeight>
<loginPopupWidth> Width in pixels </loginPopupWidth>
```

When login is configured as popup, you must provide the dimensions of the login window.

```
<vista version="1.0" />
<dashboard version="1.0" />
```

The use of the <vista> and <dashboard> elements is deprecated in IBM Worklight version 5.0.5. Support might be removed in any future version.

```
</application>
```

The closing tag.

The <security> element

The <security> element occurs under the <iphone>, <ipad>, and <android> elements. It is used to configure security mechanisms for protecting your iOS and Android apps against various malware and repackaging attacks. The element has the following structure:

```
<security>
<encryptWebResources enabled="false"/>
<testWebResourcesChecksum enabled="false" ignoreFileExtensions="png, jpg, jpeg, gif, mp4, mp3"/>
<publicSigningKey> value </publicSigningKey>
</security>
```

The element <encryptWebResources> controls whether the web resources associated with the application are packaged and encrypted within the application binary file (a file with the extension .apk or .app). If its enabled attribute is set to true, the IBM Worklight builder encrypts the resources. They are then decrypted by the application when it first runs on the device.

The element <testWebResourcesChecksum> controls whether the application verifies the integrity of its web resources each time it starts running on the mobile device. If its enabled attribute is set to true, the application calculates the checksum of its web resources and compares it with a value stored when it was first run. Checksum calculation can take a few seconds, depending on the size of the web resources. To make it faster, you can provide a list of file extensions to be ignored in this calculation.

The element <publicSigningKey> is valid only in the Android environment, under <android>/<security>. This element contains the public key of the developer certificate that is used to sign the Android app. For instructions on how to extract this value, see “Extracting a public signing key” on page 74

Deprecated elements

The following elements have been deprecated since version 4.1.3:

```
<provisioning>
<viralDistribution>
<adapters>
<mobile>
```

The following elements have been deprecated since version 5.0:

```
<worklightRootURL>
```

The following elements have been deprecated since version 5.0.0.3:

Experimental IBM Worklight online user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

```
<usage>
```

The following elements have been deprecated since version 5.0.5:

```
<dashboard>
<facebook>
<igoogle>
<vista>
```

Login form and authenticator

Your application might need a login form. A default is provided, which you can change as necessary.

Applications that require user authentication might have to display a login form as part of the authentication process. In web widgets, the login form is not part of the widget resources. It can be triggered by the authentication infrastructure used by the organization or by the IBM Worklight Server. For more information about authentication, see the module *Authentication Concepts*, and the following modules under category 8, *Authentication and security*, in "Getting started tutorials and samples" on page 29.

Development guidelines for mobile environments

This collection of topics describes a standard IBM Worklight project hierarchy, with Android, iOS, and Blackberry environments.

Integrating with source control systems

Some source code files should be held in a version control system: others should not.

There are two types of files and folders in a standard IBM Worklight project hierarchy:

- Your own source code files and some source code files that are provided in the IBM Worklight device runtime libraries.

  You should commit these files to a version control system.
- Files that are generated from your web source code and some JavaScript files that are provided with the IBM Worklight platform (such as `wlclient.js`).

  These files and folders are added to the file system every build.

  You should not commit them to a version control system.

  In the next figure, these files and folders are marked with a star (*) after their names.

Experimental IBM Worklight online user documentation
This document is provided as is.
In case of issues, refer to the online user documentation.

```
Project Name
|   .classpath
|   .project
|   tree.txt
|
+---.settings
+---adapters
+---apps
|   \---   \Application Name
|       |   application-descriptor.xml
|       |
|       +---android
|       |   +---css
|       |   +---images
|       |   +---js
|       |   +---native
|       |   |   |   .classpath
|       |   |   |   .project
|       |   |   |   .wldata
|       |   |   |   AndroidManifest.xml
|       |   |   |   project.properties
|       |   |   |
|       |   |   +---.externalToolBuilders
|       |   |   +---.settings
|       |   |   +---assets
|       |   |   |   |   icudt46l.zip
|       |   |   |   |   wlclient.properties
|       |   |   |   |
|       |   |   |   +---www (*)
|       |   |   +---bin (*)
|       |   |   +---gen (*)
|       |   |   +---res
|       |   |   +---src
|       |   +---nativeResources
|      |
|       +---blackberry
|       |   +---css
|       |   +---images
|       |   +---js
|       |   \---native
|       |       |   config.xml
|       |       |   icon.png
|       |       |   splash.png
|       |       |   .wldata
|       |       |
|       |       +---ext
|       |       |       WLExtension.jar
|       |       |
|       |       +---www (*)
|       |
|       +---blackberry10
|       |   +---css
|       |   +---images
|       |   +---js
|       |   +---native
|       |   |   |   build.xml
|       |   |   |   buildId.txt
|       |   |   |   project.properties
|       |   |   |   qnx.xml
|       |   |   |
|       |   |   +---build (*)
|       |   |   +---www (*)
|       |   |
|       +---common
|       |   |
|       |   +---css
|       |   +---images
|       |   +---js
|      +---jqueryMobile
|       |
|       +---ipad
|       |   +---css
|       |   +---images
```

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

To ensure that your source code is always synchronized with your source control system, add the (*) files and folders to the ignore list in your source control system. For Subversion, for example, perform the following steps:

- **Step 1**: Using the Tortoise extension for Subversion, right-click each file or folder that is to be ignored and add it to the ignore list.
- **Step 2**: Go up one level in the file system and commit the change to the SVN repository. The changes take effect from now on for every developer who updates the code.

For more information about the folders that are shown in the figure, see "Anatomy of an IBM Worklight Application" on page 47.

Integrate Tealeaf CX with IBM Worklight

Existing Tealeaf CX Mobile customers can integrate Tealeaf CX Mobile iOS and Android SDK into an IBM Worklight mobile application. The IBM Worklight app is enabled with insightful analytics data collection from the client side to be analyzed by using Tealeaf CX on the server-side.

Tealeaf CX gives visibility, insight, and answers for companies that do business online. It provides digital customer experience management, and customer behavior analysis solutions. Companies are enabled to better understand the purpose of a customer's online and mobile interactions, and be able to enhance the customer experience.

Here are the key benefits of using Tealeaf CX:

- Being able to discover previously unknown site experience problems so you can improve success rates and increase online revenue.
- Being able to quantify the magnitude of any identified site issue (numbers of affected customers, and impact to revenue) to prioritize corrective actions.
- Quickly understand and diagnose site problems by visually analyzing customer and site behavior.
- Dramatically reduce the time that is required to reproduce and resolve site issues.

For more information, see http://www.tealeaf.com/

Configuring Tealeaf by using the Android environment

Configuring an IBM Worklight Hybrid App to collect and send data to Tealeaf CX by using the Android environment.

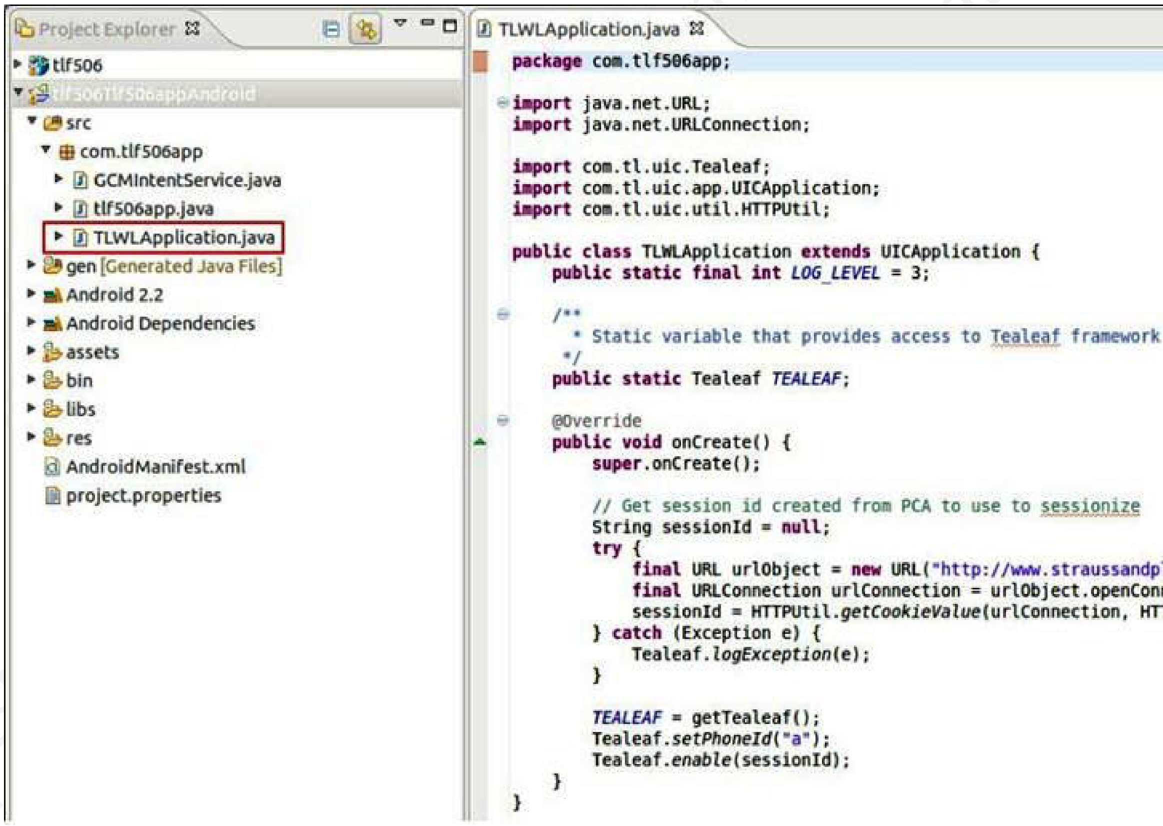
Procedure

1. Extract the Tealeaf Android SDK file (`UICAndroid.zip`) obtained from your Tealeaf installation or your Tealeaf CX administrator, to a temporary location. The compressed file contains:
   - `common/js/Tealeaf.js` Enables Tealeaf logging at the JavaScript layer
   - Android includes `android/native/assets/TLFConfigurableItems.properties` and `android/native/libs/uicandroid.jar`
2. In IBM Worklight Studio 5.0.6
   a. Create a Hybrid application, then an Android environment
   b. Add `common/js/Tealeaf.js` to the app `common/js` folder
   c. In the common folder, there is an `AppName.HTML` file, which is named after your app name, for example `common/tlfapp.html`. Update the HTML file to

Experimental IBM Worklight online user documentation. This document is provided as is. In case of issues, refer to the online user documentation.

include `js/Tealeaf.js`. Add a script tag in the main HTML file to load `Tealeaf.js <script src="js/Tealeaf.js"></script>`

3. Copy the `uicandroid.jar` to the `android/native/libs` folder of your IBM Worklight application.
4. Copy `TLFConfigurableItems.properties` to the `android/native/assets` folder of your IBM Worklight application.
5. Right-click your IBM Worklight application folder (**tlf506app**) and run as **`Build all and Deploy`** to generate the Android native project.
6. In the Android native project, create a Java class in the **src** folder. You can create it in the same Java package as the other Java classes, and call it `TLWLApplication.java`. It has the following code:



```
package com.tlf506app;

import java.net.URL;
import java.net.URLConnection;

import com.tl.uic.Tealeaf;
import com.tl.uic.app.UICApplication;
import com.tl.uic.util.HTTPUtil;

public class TLWLApplication extends UICApplication {
    public static final int LOG_LEVEL = 3;

    /**
     * Static variable that provides access to Tealeaf framework
     */
    public static Tealeaf TEALEAF;

    @Override
    public void onCreate() {
        super.onCreate();

        // Get session id created from PCA to use to sessionize
        String sessionId = null;
        try {
            final URL urlObject = new URL("http://www.straussandp
            final URLConnection urlConnection = urlObject.openCon
            sessionId = HTTPUtil.getCookieValue(urlConnection, HT
        } catch (Exception e) {
            Tealeaf.logException(e);
        }

        TEALEAF = getTealeaf();
        Tealeaf.setPhoneId("a");
        Tealeaf.enable(sessionId);
    }
}
```

   You must also add a reference to your TLWLApplication class in the application tag of the `AndroidManifest.xml` file. Add the following attribute: android:name=".TLWLApplication">.
7. Double-click the `AndroidManifest.xml` file in the Android native project to open the Android manifest editor. Click the `AndroidManifest.xml` tab in the editor to view the XML source. Add the `android:name=".TLWLApplication">` attribute to the **application** tag in the `AndroidManifest.xml` file.
8. To see Tealeaf logging in the Android ADB console, start your Android application as you normally would, and observe ADB console in Eclipse, or by running `adb logcat` from your Android SDK platform-tools directory.

Configuring Tealeaf by using the iOS environment

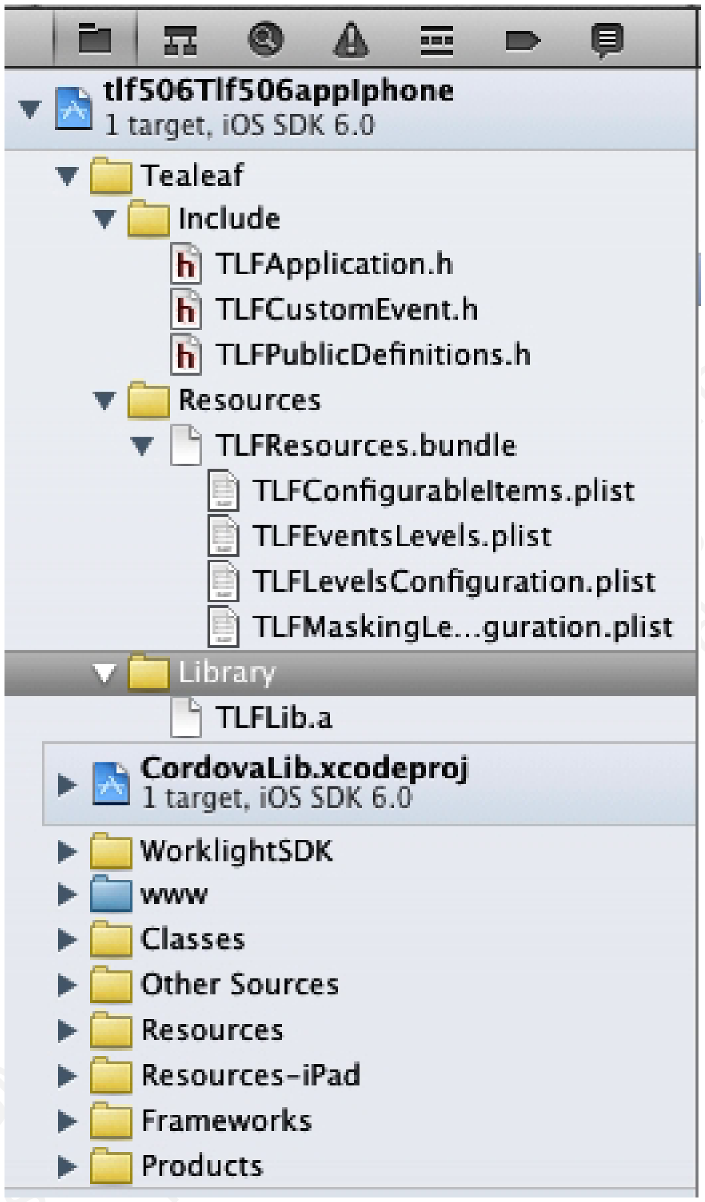
Configuring an IBM Worklight Hybrid App to collect and send data to Tealeaf CX by using the iOS environment.

Procedure

1. Extract the Tealeaf iOS SDK file `iOSMobile.zip` obtained from your Tealeaf installation or from your Tealeaf CX administrator, to a temporary location. The compressed file contains:
   - `js/Tealeaf.js` Enables Tealeaf logging at the JavaScript layer
   - iOS include files `TLFApplication.h`, `TLFCustomEvent.h`, `TLFPublicDefinitions.h`
   - iOS Tealeaf library `TLFLib.a`
   - iOS Tealeaf configuration files `TLFResource.bundle`
2. In IBM Worklight Studio 5.0.6
   a. Create a Hybrid application, then an iOS environment (use iPhone or iPad)
   b. Add `js/Tealeaf.js` (provided in the compressed file) to the apps `common/js` folder
   c. In the common folder, there is an `AppName.HTML` file, which is named after your app name, for example `common/tlfapp.html`. Update the HTML file to include `js/Tealeaf.js`. Add a script tag in the main HTML file to load `Tealeaf.js <script src="js/Tealeaf.js" type= "text/JavaScript"></script>`
3. Right-click your IBM Worklight application folder and **`Run As > Build all and Deploy`**.
4. Open the iPhone environment in Xcode
5. Add the CoreTelephony framework to the Build Phases under Link Binary With Libraries (all the other required frameworks are added by IBM Worklight when the project is created).
6. Add the Tealeaf group and the subgroups Include, Resources, and Library to your project.
7. Add the files from the temporary folder directly into the appropriate group. You can do that by selecting the group, for example, Include, then right-click > Add Files to <project name>. To add the files to the Include group highlight them, select the **Copy items into destination group's folder** check box. Select the **Add to targets** check box, which specifies the name, for example `tlf506Tlf506applphone`. Verify that the targets of your application are selected in the **Add to targets** section. After you add the files to each group, here is an example of how your project looks.

Figure 2. Example

Experimental IBM Worklight offline user documentation.
This document is provided "as is".
In case of issues, refer to the online user documentation.



8. In your application's `main.m` file, you must instruct **UIApplicationMain** to use the Tealeaf subclass of `UIApplication`. Go to the **Other Sources** folder, open your `main.m` file and add the Include files

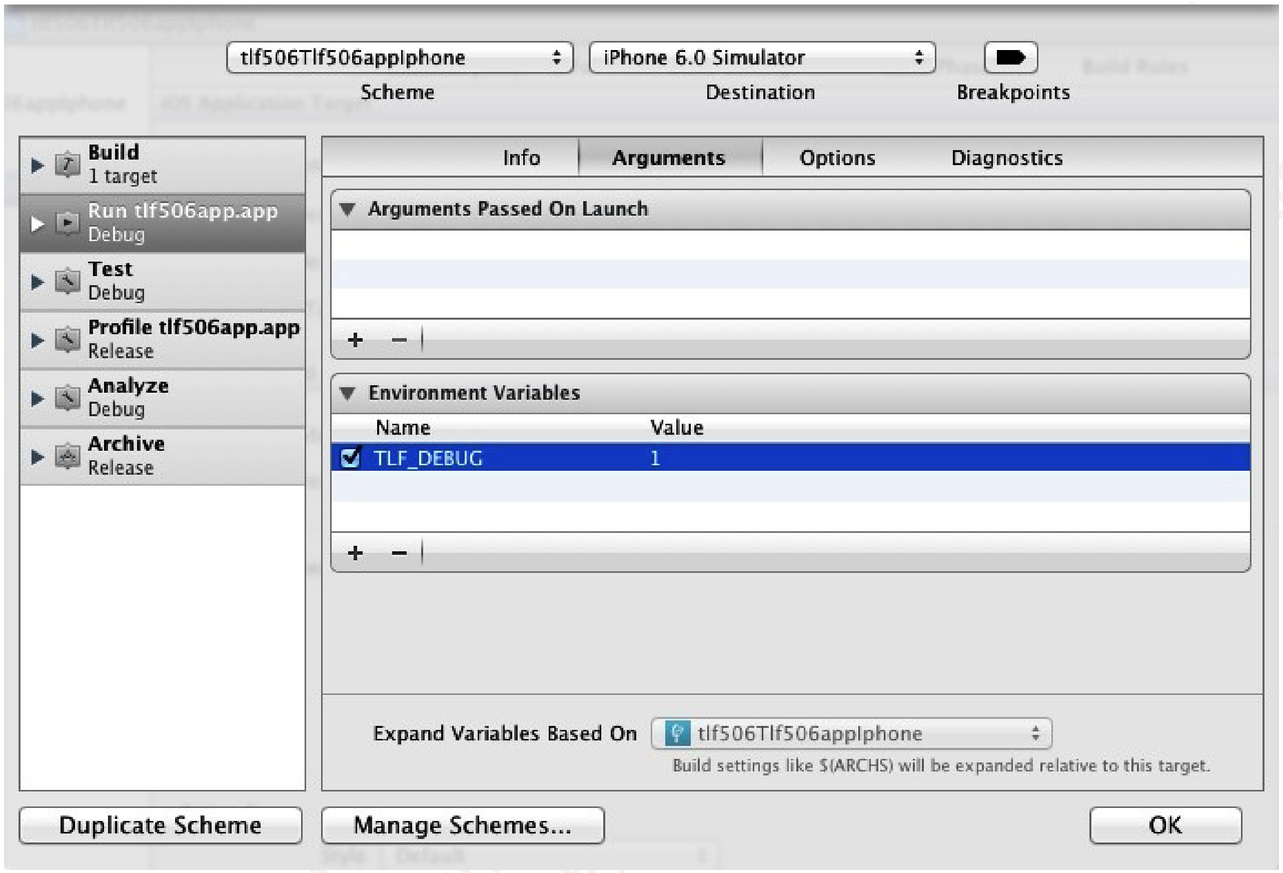
   ```
   #import "TLFPublicDefinitions.h"
   #import "TLFApplication.h"
   #import "TLFCustomEvent.h"
   ```

   Then you add code to initialize the Tealeaf runtime by passing a reference to the **TLFApplication class** into **UIApplicationMain**:

   ```
   NSString appClass = NSStringFromClass([TLFApplication class]);

   // int retVal = UIApplicationMain (argc, argv, nil, @"MyAppDelegate");
   int retVal = UIApplicationMain (argc, argv, appClass @"MyAppDelegate");
   ```

9. The URL of the Tealeaf Server to which the data is sent, is configured in `Tealeaf>Resources>TLFResources.bundle` in the file `TLFConfigurableItems.plist` in the property *PostMessageURL*. There are other configurable options in this file as well. For more information, see the Tealeaf documentation on these configuration options.

10. Optional. If you want to see the Tealeaf logs in the Xcode console, edit the Scheme and set the *TLF_DEBUG* environment variable to 1. With TLF_DEBUG set to 1, the Tealeaf data is logged to the Xcode console in addition to being sent to the Tealeaf server.



Name	Value
✓ TLF_DEBUG	1

Expand Variables Based On: tlf506Tlf506appIphone

Build settings like $(ARCHS) will be expanded relative to this target.

Figure 3. Debug

Application skins

An application skin is a set of web resources that govern the appearance and behavior of the application. Skins are used to adjust the application to different devices of the same family. You can package multiple skins in your application and decide at run time, on application startup, which skin to apply to the application.

Note: Only the following environments support application skins: Android, iPhone, iPad, BlackBerry 6, 7 and 10.

When you define a skin in the IBM Worklight Studio, the studio generates a folder for the skin resources and adds a <skin> element in the application descriptor. The <skin> element includes the name of the skin and a list of resource folders. When the studio builds the application, it applies the optimization rules on the resource folders in the order they occur within the <skin> element.

In the following example, two skins are packaged with the Android application: the default skin and another skin called `android.tablet`. Resources for the `android.tablet` skin are in the `android.tablet` folder.

Experimental IBM This docum refer to In case of issu

```
<android>
<skins>
<skin name="default">
<folder name="common" />
<folder name="android" />
</skin>
<skin name="android.tablet">
<folder name="common" />
<folder name="android" />
<folder name="android.tablet" />
</skin>
</skins>
</android>
```

You can also create custom skin hierarchies, by creating resource folders under the application folder and manually defining the skin hierarchy in the application descriptor. For example, you can define a `phone` folder to include resources that are related to rendering the app on a phone, and a `tablet` folder to include resources for rendering the app on a tablet. Then you can create four skins by using these resources in the following way:

- android.phone: common > android > phone
- android.tablet: common > android > tablet
- ios.phone: common > iphone > phone
- ios.tablet: common > iphone > tablet

Applying skins at run time

To set which skin to apply at run time, implement the function `getSkinName()` in the file `skinLoader.js`. This file is located under the `/common/js` folder for the app.

Deleting a skin

To delete a skin, remove the element that defines the skin from the app descriptor.

Web and native code in iPhone, iPad, and Android

Using the IBM Worklight platform, you can include in your applications pages that are developed in the native operating system language.

The natively developed pages can be invoked from your web-based pages and can then return control to the web view. You can pass data from the web page to the native page, and return data in the opposite direction. You can also animate the transition between the pages in both directions.

Switching the display from the web view to a native page

You can include in your applications pages developed in the native operating system language and can switch between them and the web view.

About this task

In iPhone, iPad, and Android applications, natively developed pages can be invoked from your web-based pages and can then return control to the web view. You can pass data from the web page to the native page, and return data in the opposite direction. You can also animate the transition between the pages in both directions.

Experimental IBM Worklight offline user documentation
This document is provided "as is"
In case of issues, refer to the online user documentation.

Procedure

To switch the display from the web view to a native page, use the WL.NativePage.show method.

Receiving data from the web view in an Objective-C page

To receive data from the calling web view, follow these instructions.

Before you begin

The native page must be implemented as an Objective-C class that inherits from UIViewController. This UIViewController class must be able to initialize through the `init` method alone. The initWithNibName:bundle: method is never called on this class instance.

Procedure

Write a UIViewController class that implements the method setDataFromWebView:.

```
-(void) setDataFromWebView:(NSDictionary *)data{
    NSString  = (NSString *) [data valueForKey:@"key"];
}
```

Related information:

http://developer.apple.com/library/ios/#documentation/UIKit/Reference/UIViewController_Class/Reference/Reference.html%23//apple_ref/occ/cl/UIViewController

Returning control to the web view from an Objective-C page

To switch back to the web view, follow these instructions.

Before you begin

The native page must be implemented as an Objective-C class that inherits from UIViewController. This UIViewController class must be able to initialize through the `init` method alone. The initWithNibName:bundle: method is never called on this class instance.

Procedure

In the native page, call the [NativePage showWebView:] method and pass it an NSDictionary object (the object can be empty). This NSDictionary can be structured with any hierarchy. The IBM Worklight runtime framework encodes it in JSON format, and then sends it as the first argument to the JavaScript callback function.

```
// The NSDictionary object will be sent as a JSON object to the JavaScript layer in the webview
[NativePage showWebView:[NSDictionary dictionaryWithObject:@"value" forKey:@"key"]]
```

Related information:

http://developer.apple.com/library/ios/#documentation/UIKit/Reference/UIViewController_Class/Reference/Reference.html%23//apple_ref/occ/cl/UIViewController

Animating the transition from an Objective-C page to a web view

To implement a transition animation when switching the display from the native page to the web view, follow these instructions.

Experimental IBM Worklight offline user documentation
This document is provided "as-is".
In case of issues, refer to the online user documentation.

Procedure

Within your animation code, call the [NativePage showWebView] method.

```
-(IBAction)returnClicked:(id)sender{
NSString *phone = [phoneNumber text];
NSDictionary *returnedData = [NSDictionary dictionaryWithObject:phone forKey:@"phoneNumber"];

// Animate transition with a flip effect
CDVAppDelegate *cordovaAppDelegate = (CDVAppDelegate *)[[UIApplication sharedApplication] delegate];

[UIView beginAnimations:nil context:NULL];
[UIView setAnimationDuration:0.5];
[UIView setAnimationTransition:UIViewAnimationTransitionFlipFromRight
forView:[cordovaAppDelegate window] cache:YES];

[UIView commitAnimations];

// Return to WebView
 [NativePage showWebView:returnedData];
}
```

Animating the transition from a web view to an Objective-C page

To implement a transition animation when switching the display from the web view to the native page, follow these instructions.

Procedure

Implement the methods: onBeforeShow and onAfterShow. These methods are called before the display switches from the web view to the native page, and after the transition.

```
-(void)onBeforeShow{
CDVAppDelegate *cordovaAppDelegate = (CDVAppDelegate *)[[UIApplication sharedApplication] delegate];
[UIView beginAnimations:nil context:NULL];
[UIView setAnimationDuration:0.5];
[UIView setAnimationTransition:UIViewAnimationTransitionFlipFromRight forView:[cordovaAppDelegate wi
}
-(void)onAfterShow{
[UIView commitAnimations];
}
```

Receiving data from the web view in a Java page

To receive data from the calling web view, follow these instructions.

Before you begin

The page must be implemented as an `Activity` object or extend an `Activity`. As with any other activity, you must declare this activity in the `AndroidManifest.xml` file.

Procedure

To receive data from the calling web view, use the `Intent` object defined on the native `Activity`. The IBM Worklight client framework makes the data available to the `Activity` in a `Bundle`.

Example

Sending data from web view to the native Activity:

```
WL.NativePage.show('com.example.android.tictactoe.library.GameActivity', this.callback, {"gameLevel"
```

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Receiving the data in the native Activity:

```
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);

//Read int value, default = 0
Integer gameLevel = getIntent().getIntExtra("gameLevel", 0);

//Read String value
String playerName = getIntent().getStringExtra("playerName");

//Read boolean value, default = false
Boolean isKeyboardEnable = getIntent().getBooleanExtra("isKeyboardEnable", false);
}
```

Related information:

http://developer.android.com/reference/android/content/Intent.html

http://developer.android.com/reference/android/app/Activity.html

http://developer.android.com/reference/android/os/Bundle.html

Returning control to the web view from a Java page

To switch back to the web view, follow these instructions

Before you begin

The page must be implemented as an `Activity` object or extend an `Activity`. As with any other activity, you must declare this activity in the `AndroidManifest.xml` file.

Procedure

In the native page, call the `finish()` function of the `Activity`. You can pass data back to the web view by creating an `Intent` object.

Example

Passing data and control to the web view:

```
Intent gameInfo = new Intent ();
gameInfo.putExtra("winnerScore", winnerScore);
gameInfo.putExtra("winnerName", winnerName);
setResult(RESULT_OK, gameInfo);
finish();
```

Receiving the data in the web view:

```
this.callback = function(data){$('resultDiv').update('The winner is: ' + data.winnerName + " with
```

Related information:

http://developer.android.com/reference/android/app/Activity.html

Animating the transitions from and to a Java page

To animate the transitions between a web view and a native page, follow these instructions.

Experimental IBM Worklight offline user documentation.
This document is provided "as is".
In case of issue, refer to the online user documentation.

Procedure

To add transition animation, use the `Activity` function `OverridePendingTransition(int, int)`.

Example

```
// Transition animation from the web view to the native page
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
overridePendingTransition(android.R.anim.fade_in, android.R.anim.fade_out);
}

// Transition animation from the native page to the web view
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
//your code goes here....
finish();
overridePendingTransition(android.R.anim.fade_in, android.R.anim.fade_out);
}
```

Related information:

http://developer.android.com/reference/android/app/Activity.html#overridePendingTransition(int, int)

Creating an IBM Worklight BlackBerry 10 environment

Follow these instructions to create an IBM Worklight BlackBerry 10 environment.

About this task

The BlackBerry 10 environment uses the latest version of Cordova, version 2.3. However, not all of the Cordova application programming interface (API) is supported yet, for example the Cordova **contacts object**. Some code can work across platforms if written in Cordova, but some must be written by using the WebWorks API. Use either Ripple or Cordova Ant scripts, and to ensure that your program runs correctly, follow these steps.

Procedure

1. Follow all instructions to install WebWorks SDK, described at HTML5 WebWorks.
2. If you are using Ant scripts, manually modify the `project.properties` file. Provide values for the following variables in `project.properties`. This is not relevant if you are using Ripple.

```
# BB10 Code Signing Password
 qnx.sigtool.password=


 For simulator:
 # QNX Simulator IP
 #
 #   If you leave this field blank, then
 #   you cannot deploy to simulator
 #
 qnx.sim.ip=

 # QNX Simulator Password
 #
 #   If you leave this field blank, then
 #   you cannot deploy to simulator
```

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to online user documentation.

```
#
qnx.sim.password=

for device:

We give the initial device ip of 169.254.0.1 which is usually the one, when connected via USB
# QNX Device IP
#
#   If you leave this field blank, then
#   you cannot deploy to device
#
qnx.device.ip=169.254.0.1

You also must change
# QNX Device Password
#
#   If you leave this field blank, then
#   you cannot deploy to device
#
qnx.device.password=

# QNX Device PIN
#
#   Fill this value in to use debug tokens when debuging on the device
qnx.device.pin=
```

3. Do **not** delete or change the following elements in `config.xml`:

   ```
   <!-- start_worklight_host_server do not change this line-->
     <access subdomains="true" uri="http://9.148.225.82" />
     <!-- end_worklight_host_server do not change this line-->
   ```

   The correct server TCP/IP address is automatically put in the `<access>` element on each Worklight build. If this element is deleted or changed, the TCP/IP address cannot be automatically updated.
4.
   a. BlackBerry 10 supports Ripple. If you intend to use Ripple, specify `{project name}/apps/{app name}/blackberry10/native/www` as the root folder in Ripple.
   b. BlackBerry 10 is based on QNX. To run the app on the phone using Cordova Ant scripts, you use the **`ant qnx`** command. Look in `native/qnx.xml` for a list of available commands. For example, **`ant qnx`** `debug-device` builds, deploys, and runs the app on the device.

Specifying the icon for an Android application

Put the icon in your application's `/android/nativeResources/res` folder. It is copied from there at build time.

About this task

You want to use a particular icon for your application in the Android environment.

Procedure

1. Place the icon that you want to use in the *`project`*`/apps/`*`application`*`/android/nativeResources/res` folder.
2. Build and deploy your application.

Results

The icon is copied to the *`project`*`/apps/`*`application`*`/android/native/res` folder.

Experimental IBM Worklight online user documentation. In case of issues, refer to the online user documentation. This document is provided "as is".

Though you can place the icon directly into the *project*/apps/*application*/android/native/res folder, you risk losing the icon if that folder is deleted for any reason.

Specifying the icon for an iPhone application

Put the icon in your application's /iphone/nativeResources/Resources folder. It is copied from there at build time.

About this task

You want to use a particular icon for your application in the iPhone environment.

Procedure

1. Place the icon that you want to use in the *project*/apps/*application*/iphone/nativeResources/Resources folder.
2. Build and deploy your application.

Results

The icon is copied to the *project*/apps/*application*/iphone/native/Resources folder.

Though you can place the icon directly into the *project*/apps/*application*/iphone/native/Resources folder, you risk losing the icon if that folder is deleted for any reason.

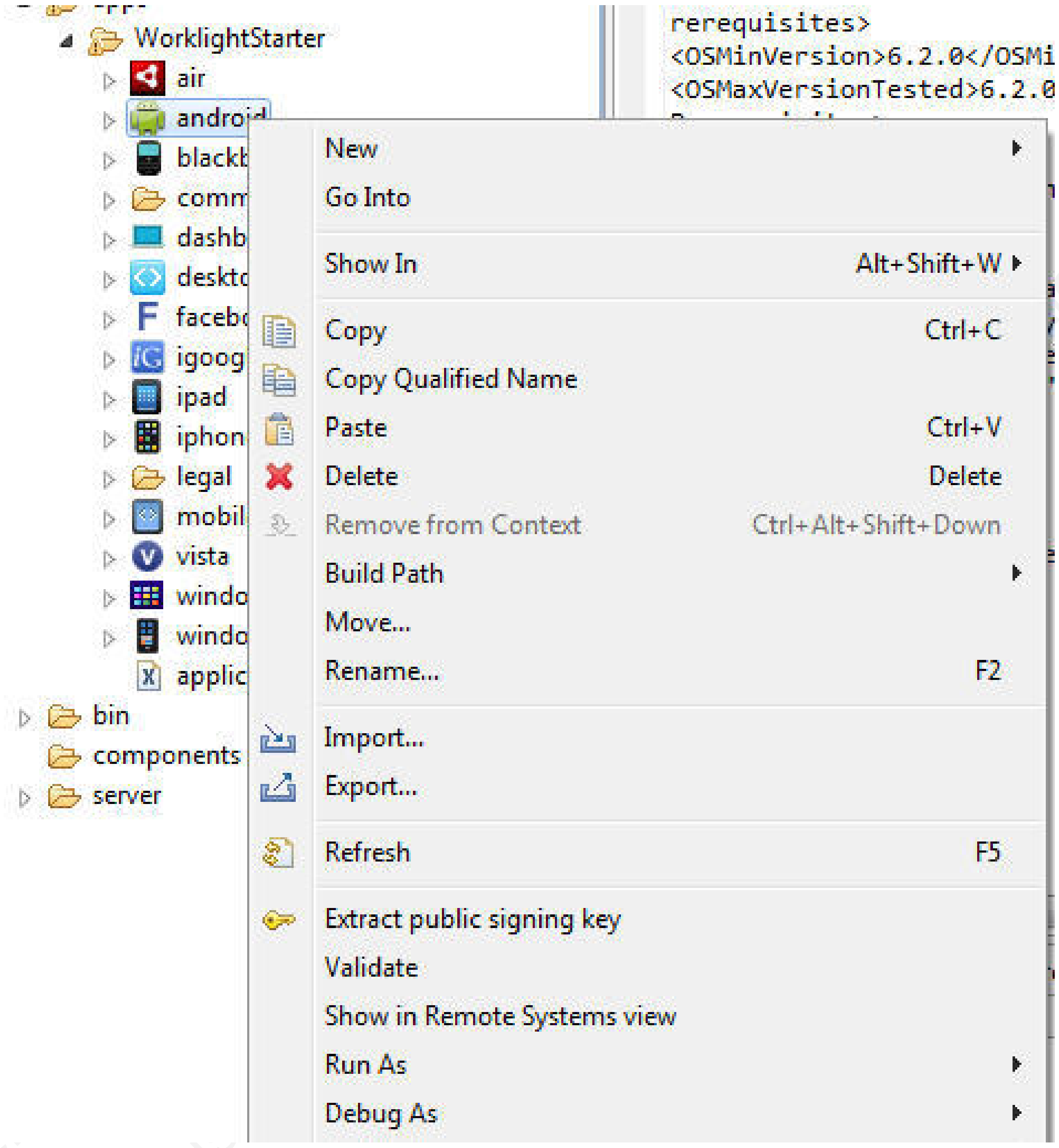
Extracting a public signing key

Copy the public signing key from the keystore to the application descriptor.

Procedure

1. In the Eclipse project explorer, in the android folder for the application, click the **Extract public signing key** menu item.

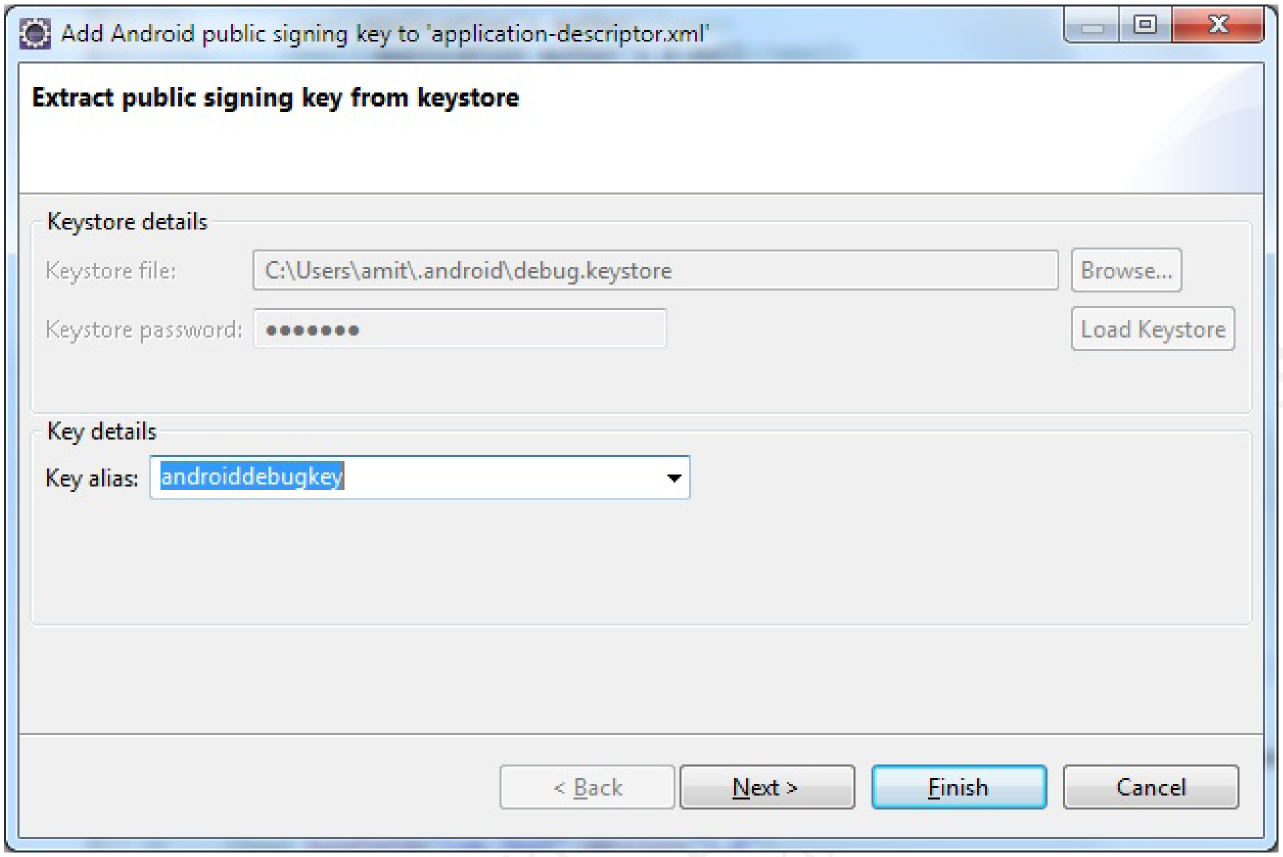
Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.



```
rerequisites>
<OSMinVersion>6.2.0</OSMi
<OSMaxVersionTested>6.2.0
```

Figure 4. Extracting the public signing key

A wizard window opens.


Add Android public signing key to 'application-descriptor.xml'

Extract public signing key from keystore

Keystore details

Keystore file: C:\Users\amit\.android\debug.keystore

Keystore password: ●●●●●●●

Key details

Key alias: androiddebugkey

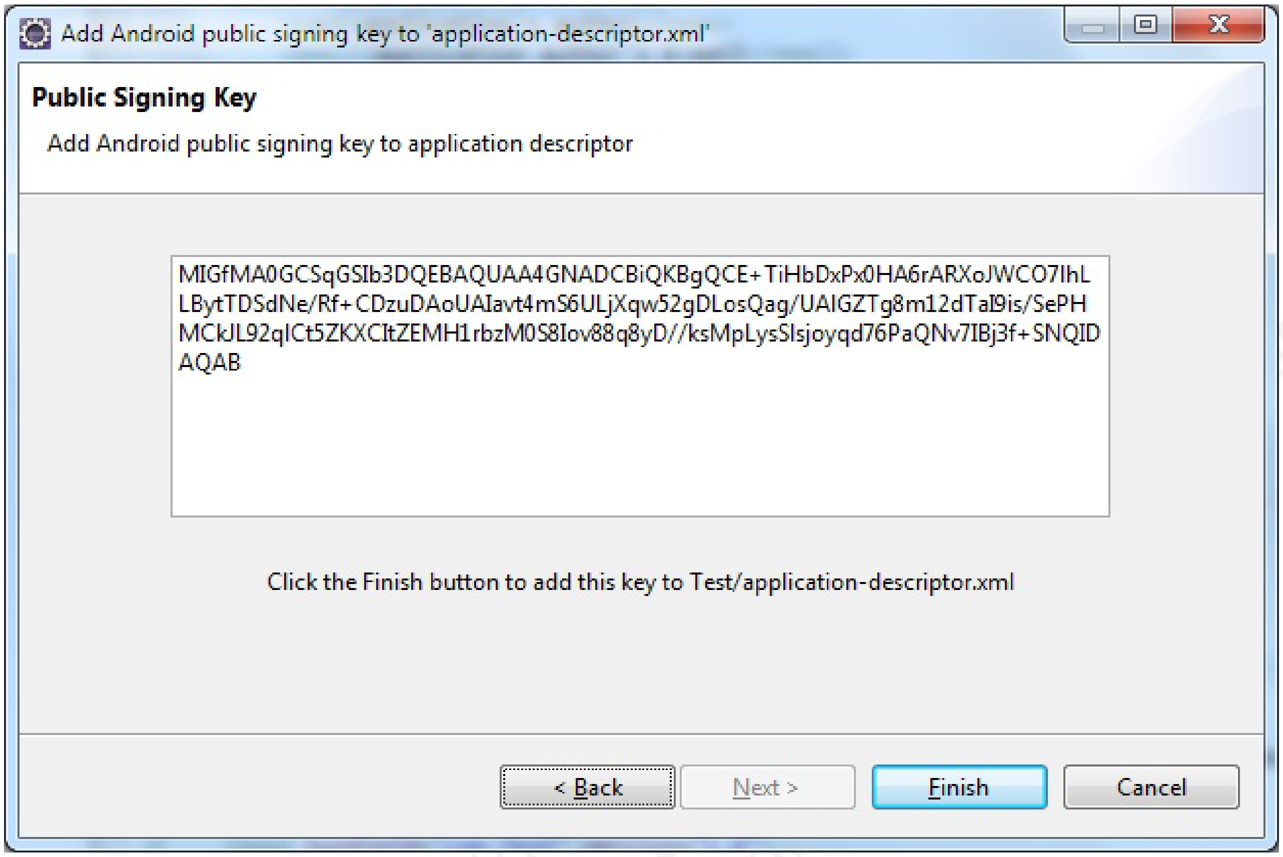
Figure 5. Adding the Android public signing key

2. In this window, enter the path to your keystore. The keystore is usually in one of the following directories, according to operating system:

Option	**Description**
Windows 7 and Windows Vista	`C:\Users\`*`user_name`*`\.android\`
Windows XP	`C:\Documents and Settings\`*`user_name`*`\.android\`
OS X and Linux	`~/.android/`

3. Enter the password to your keystore and click **Load Keystore**.
4. When the keystore is loaded, select an alias from the **Key alias** menu and click **Next**. For more information about the Android keystore, see http://developer.android.com/guide/publishing/app-signing.html.
5. In the window, click **Finish** to copy the public signing key directly into the application descriptor.

Experimental IBM Worklight. This document is for information only. In case of issues, refer to the online documentation.


Add Android public signing key to 'application-descriptor.xml'
Public Signing Key
Add Android public signing key to application descriptor

```
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCE+TiHbDxPx0HA6rARXoJWCO7IhL
LBytTDSdNe/Rf+CDzuDAoUAIavt4mS6ULjXqw52gDLosQag/UAIGZTg8m12dTal9is/SePH
MCkJL92qICt5ZKXCItZEMH1rbzM0S8Iov88q8yD//ksMpLysSIsjoyqd76PaQNv7IBj3f+SNQID
AQAB
```

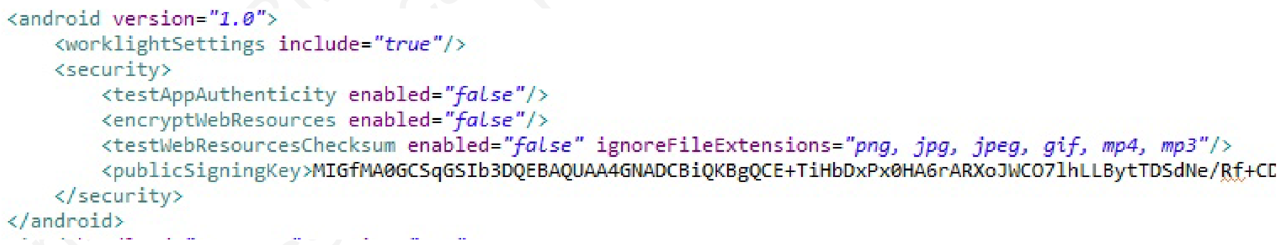
Click the Finish button to add this key to Test/application-descriptor.xml

< Back
Next >
Finish
Cancel

Figure 6. Android public signing key

Results

The public key is copied to the application descriptor. See the following figure:



```
<android version="1.0">
    <worklightSettings include="true"/>
    <security>
        <testAppAuthenticity enabled="false"/>
        <encryptWebResources enabled="false"/>
        <testWebResourcesChecksum enabled="false" ignoreFileExtensions="png, jpg, jpeg, gif, mp4, mp3"/>
        <publicSigningKey>MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCE+TiHbDxPx0HA6rARXoJWCO7lhLLBytTDSdNe/Rf+CD
    </security>
</android>
```

Figure 7. Android public signing key as shown in the application descriptor

Connecting to Worklight Server

By default, an application starts in offline mode. You can make it start in online mode, or can connect to Worklight Server later. You are responsible for maintaining the offline or online state within your application, and ensuring that your application can recover from failed attempts to connect to the server. For example, before the application logs in a new user or accesses the server under a new user, the application must ensure that a successful logout was received by the server.

About this task

By default, an application is started in offline mode. It is likely that you will want your application to connect to the Worklight Server, either when it starts or at some appropriate point in its processing. Methods for connecting are detailed in the following steps.

Procedure

- To make your application begin communicating with Worklight Server as soon as it starts, change the connectOnStartup property in the `initOptions.js` file to true. The Worklight framework automatically attempts to connect to Worklight Server as part of application startup. This approach might increase the time it takes for the application to start.
- To make your application communicate with the server at a later stage, call the method WL.Client.connect. Call this method only once, before any other WL.Client methods that communicate with the server. Remember to implement onSuccess and onFailure callback functions, for example:

```
WL.Client.connect({
    onSuccess: onConnectSuccess,
    onFailure: onConnectFailure
});
```

Related reference:

"WL.Client.connect" on page 182
This method establishes a connection to the Worklight Server.

Adding custom code to an Android app

Adding custom code to your Android app in the onCreate method is deprecated. To add custom code to your Android app, use the onWLInitCompleted method.

Since IBM Worklight V5.0.6, add custom code to the onWLInitCompleted method. The onWLInitCompleted method is invoked when the IBM Worklight initialization process is complete and the client is ready.

In IBM Worklight V5.0.5 and earlier, custom code was added to the onCreate method. However, since IBM Worklight V5.0.6, adding custom code to the onCreate method is deprecated. Support might be removed in any future version.

If you migrate an existing Android app to IBM Worklight V5.0.6, any custom code in the onCreate method is automatically moved to the onWLInitCompleted method during the migration process. A comment is also added to indicate that the code was moved.

The following code snippet is an example of a new application:

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

```
public class b extends WLDroidGap {

 @Override
 public void onCreate(Bundle savedInstanceState){
  super.onCreate(savedInstanceState);
 }

 /**
     * onWLInitCompleted is called when the Worklight runtime framework initialization is complete.
     */
 @Override
 public void onWLInitCompleted(Bundle savedInstanceState){
  super.loadUrl(getWebMainFilePath());
  // Add custom initialization code after this line
 }
}
```

Figure 8. Custom code in a new application

The following code snippet is an example of a migrated application:

```
@Override
 public void onWLInitCompleted(Bundle savedInstanceState) {
    //Additional initialization code from onCreate() was moved here
     super.loadUrl(getWebMainFilePath());
  }

  @Override
  public void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     //Additional initialization code was moved to onWLInitCompleted().
  }
```

Figure 9. Custom code in a migrated application

Development guidelines for desktop and web environments

This collection of topics gives instructions for implementing various functions in desktop and web applications.

Specifying the application taskbar for Adobe AIR applications

How to display or suppress a taskbar button for a widget.

About this task

Unlike Windows desktop gadgets and Apple Dashboard widgets, Adobe AIR applications can be displayed on the system taskbar. Widgets that are opened for a short time (for example, to perform a specific task) and are then closed should normally have a taskbar button. Conversely, widgets that remain constantly open on the desktop should not have a taskbar button, to save the space required by the button. Instead, such widgets have a tray icon that allows access to the widget.

If the taskbar button is not displayed, IBM Worklight adds a tray icon for the widget. You can use the tray icon to minimize the application, restore it, and close it.

Procedure

- To control whether your desktop widget is displayed on the taskbar, specify the `<air>` element in the application descriptor. If the `<air>` element is not specified, the taskbar button is displayed.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues refer to the online user documentation.

- To display a taskbar button for the widget, specify: `<air showInTaskbar="always" />`.
- To avoid displaying a taskbar button for the widget, specify: `<air showInTaskbar="never" />`

Configuring the authentication for web widgets

Add a realm to the `authenticationConfig.xml` file.

About this task

The `authenticationConfig.xml` file, in the *Worklight Project Name*`/server/conf` folder, is used to configure how widgets and web applications authenticate users.

For more information about configuring realms, see "Authentication configuration" on page 137.

Procedure

In the `authenticationConfig.xml` file, add a realm that uses the login forms, as follows:

```
<realm name="realm-name" loginModule="login-module-name">
<className>com.worklight.core.auth.ext.FormBasedAuthenticator</className>
<parameter name="login-page" value="/apps/services/login-file-name" />
</realm>
```

Writing login form files for web widgets

Write two files, in HTML or JSP, with the ability to carry out a security check.

Procedure

1. Create two files, one displaying the login form and another one displaying the form after a login error occurred. The files can be HTML or JSP. Both login page and login error page must be able to submit a form with the action `j_security_check` and have **`j_username`** and **`j_password`** parameters. This technique is shown in the following code example:

   ```
   <form method="post" action="j_security_check">
   <input type="text" name="j_username"/>
   <input type="password" name="j_password"/>
   </form>
   ```
2. Save both files in the *Worklight_Project_Name*`/server/webapps/gadgets-serving` folder.

Setting the size of the login screen for web widgets

If your login page is displayed in a separate browser window, configure its height and width.

Procedure

If your login page is displayed in a separate browser window, configure its height and width in the application descriptor, by using the `<loginPopupHeight>` and `<loginPopupWidth>` elements.

Deploying applications on iGoogle

To deploy an iGoogle widget on iGoogle, follow these instructions.

Experimental IBM Worklight online user documentation
In case of issues, refer to the online user documentation.
This document is provided "as is".

About this task

The use of iGoogle widgets is deprecated in Worklight version 5.0.5. Support might be removed in any future version. Use standard desktop web applications.

Procedure

1. In the Catalog screen of the IBM Worklight Console, locate the application, and then click **Download Descriptor**. The iGoogle descriptor file is downloaded to your computer.
2. Upload the descriptor file to a publicly visible place in your website.
3. Use the iGoogle services to create *Add-To* links to your descriptor.

Deploying applications on Facebook

Deprecated. To create a Facebook version of an application, follow these instructions.

Before you begin

If your Worklight Studio internal application server does not run on the default port, 8080, make sure that you set this port as the value of the configuration **`publicWorkLightPort`** in the *`Worklight_Project_Name`*`/server/conf/worklight.properties` file. Otherwise, the action **Get URL for Facebook** does not provide you with the correct URL.

About this task

The creation of Facebook versions of applications is deprecated in Worklight version 5.0.5. Support might be removed in any future version. Use standard desktop web applications.

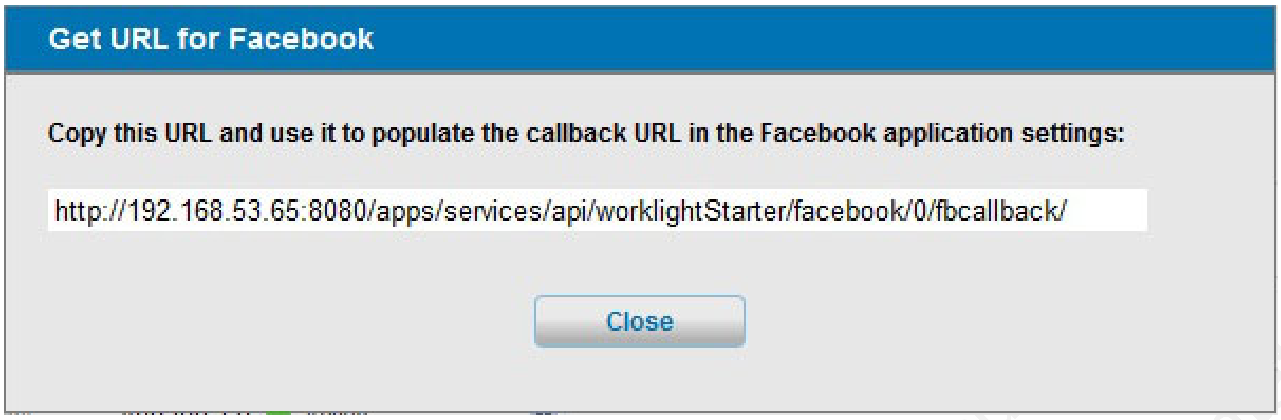
IBM Worklight applications run in Facebook as Facebook applications. Unlike personalized home page widgets or desktop widgets, Facebook applications must be declared manually in Facebook, in order to be displayed on the Facebook canvas.

Procedure

To create a Facebook version of an application:

1. In the Catalog screen of the IBM Worklight Console, locate the application, and then click **Get App URL**. A window is displayed, containing the Canvas Callback URL, as shown in the following screen image.

Experimental IBM Worklight online user documentation.
This document is provided "as is".
In case of issues, refer to the online user documentation.



Copy this URL and use it to populate the callback URL in the Facebook application settings:

http://192.168.53.65:8080/apps/services/api/worklightStarter/facebook/0/fbcallback/

Figure 10. Canvas Callback URL

2. In Facebook, create a Facebook application. In the **Canvas URL** field, enter the URL you obtained from the IBM Worklight Catalog.
3. Set the HTTPS-based URL for the app, as exposed to the Facebook servers, based on the configuration of your web environment.
4. Restart the IBM Worklight Server.

Signing Adobe AIR applications

Worklight provides a default certificate for development and test purposes. For production, obtain a certificate from a certificate authority and install it.

About this task

Adobe AIR applications must be digitally signed in order for users to install them. IBM Worklight provides a default certificate for signing AIR applications that can be used for development and test purposes.

To sign an AIR application for production distribution, using your own certificate, follow these instructions:

Procedure

1. Obtain a PKCS12 certificate from a certificate authority, and export it as a PFX file.
2. Place this certificate on your hard disk.
3. Set the `<certificate>` element under the `<air>` element in the application descriptor. The structure of the `<certificate>` element is:

   ```
   <certificate password="password" PFXFilePath="path-to-pfx"/>
   ```

   where *password* is the password for the PFX certificate, and *path-to-pfx* can either be relative to the root of the application, or an absolute path.

Signing Windows 8 apps

Worklight provides a default certificate for development and test purposes. For production, obtain a certificate from a certificate authority and install it.

Experimental IBM Worklight online user documentation.
In case of issues, refer to the online user documentation.
This document is provided "as is".

About this task

Windows 8 apps should be digitally signed before users install them. IBM Worklight provides a default certificate for signing Windows 8 apps that can be used for development and test purposes.

To sign a Windows 8 app for production distribution, using your own certificate, follow these instructions:

Note: : You can sign Windows 8 apps only on Windows systems.

Procedure

1. See http://msdn.microsoft.com/en-us/library/windows/apps/br230260.aspx for details on obtaining a PKCS12 certificate.
2. Export the PKCS12 certificate as a PFX file.
3. Place this certificate on your hard disk.
4. Set the `<certificate>` subelement under the `<windows8>` element in the application descriptor. The structure of the `<certificate>` subelement is:`<certificate PFXFilePath="Path to certificate file" password="certificate password"/>`, where *Path to certificate file* can either be relative to the root of the application, or an absolute path, and *password* is the password for the PFX certificate.

Signing Windows 7 and Vista gadgets

Worklight provides a default certificate for development and test purposes. For production, obtain a certificate from a certificate authority and install it.

About this task

The use of Windows 7 and Vista gadgets is deprecated in Worklight version 5.0.5. Support might be removed in any future version.

Windows 7 and Vista gadgets should be digitally signed before users install them. IBM Worklight provides a default certificate for signing Windows gadgets that can be used for development and test purposes.

To sign a Windows gadget for production distribution, using your own certificate, follow these instructions:

Note: You can sign Windows 7 and Vista gadgets only on Windows systems.

Procedure

1. Install the following software (in this order):
   a. Microsoft .NET Framework 4.

      You can obtain the .NET Framework at http://www.microsoft.com/download/en/details.aspx?id=17851.
   b. The .NET Development Tools Component from the Microsoft Windows SDK v7.1.

      You can obtain this component at http://www.microsoft.com/download/en/details.aspx?id=8279
2. Obtain a PKCS12 certificate from a certificate authority, and export it as a PFX file.
3. Place this certificate on your hard disk.

Experimental IBM Worklight offline user documentation.
This document is provided "as is".
In case of issues, refer to the online user documentation.

4. Set the `<certificate>` element under the `<vista>` element in the application descriptor. The structure of the `<certificate>` element is:`<certificate password="`*password*`" PFXFilePath="`*path-to-pfx*`"/>`, where *password* is the password for the PFX certificate, and *path-to-pfx* can either be relative to the root of the application, or an absolute path.

Embedding widgets in predefined web pages

Follow these instructions to incorporate widgets into web pages.

Before you begin

If your Worklight Studio internal application server does not run on the default port 8080, make sure that you also set this port as the value of the configuration `publicWorkLightPort` in the *`Worklight_Project_Name`*`/server/conf/worklight.properties`file. Otherwise, the action **Embed in Web Page** does not provide you with the correct URL.

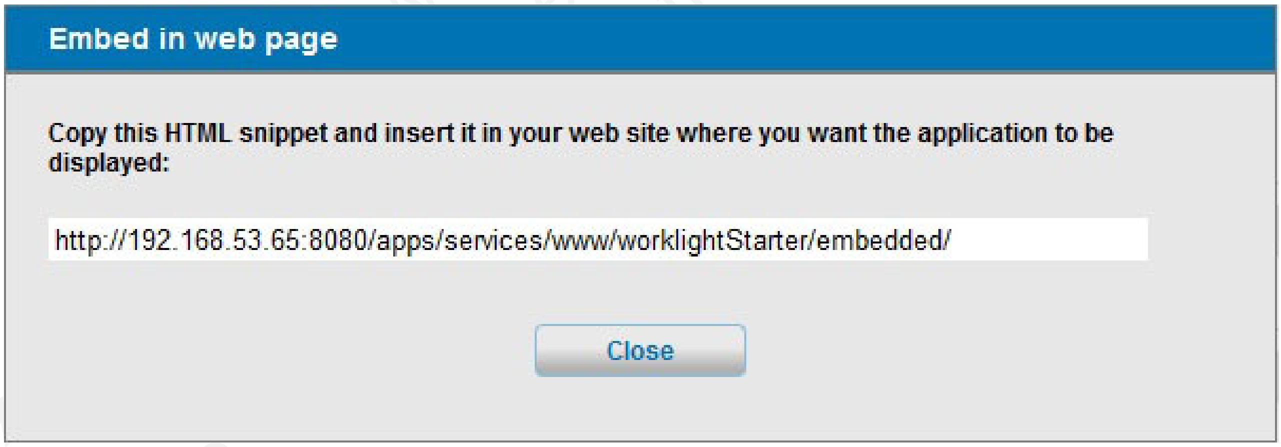
About this task

IBM Worklight widgets can be embedded in predefined web pages, such as corporate websites or intranet portals.

Procedure

To embed a widget in a predefined web page:

1. In the IBM Worklight Catalog in the IBM Worklight Administration Console, locate the widget, and then click **Embed in web page**. A window is displayed, which contains the URL of the application to which you point in your website to embed the widget. The following figure shows the window:


Embed in web page

Copy this HTML snippet and insert it in your web site where you want the application to be displayed:

http://192.168.53.65:8080/apps/services/www/worklightStarter/embedded/

Figure 11. Embedding a widget in a predefined web page

2. Paste the URL in an HTML snippet in the web page where you want to embed the widget.

```
<iframe src="URL_to_embed" width="widget_width" height="widget_height" style="border:none;"> </if
```

Development guidelines for using native API

This collection of topics gives instructions for developing native mobile applications by using the IBM Worklight native API.

Similar to other types of mobile applications with IBM Worklight, you start the development of your native app in Worklight Studio by creating a Worklight application. To develop a native app, you must create an Worklight application of type *Native API*. Your native application requires the content of such a *Native API* application. This content depends on the selected mobile environment, and your native application requires it to use the corresponding IBM Worklight native API:

- The IBM Worklight Objective-C client-side API, if your Native API application is for the iOS environment
- The IBM Worklight Java client-side API, if your Native API application is for the Android environment
- The IBM Worklight Java client-side API, if your Native API application is for the Java Platform, Micro Edition (Java ME)

To create a Native API application, you have several options:

- If you already have a Worklight project, you can create and add your Native API application in it:
  1. Click **New** > **Worklight Native API**.
  2. Select the existing project.
  3. Set the application name.
  4. Specify the environment that you need: **Android**, **iOS**, or **Java ME**.
  5. Click **Finish**.

     You created a Native API application in your Worklight project in Worklight Studio.
- If you do not have a Worklight project, you can create a Worklight project of type Native API, and request to create a Native API application as its first application in it:
  1. Click **New** > **Worklight Project**, and then select the **Native API** template.
  2. Set the application name.
  3. Specify the environment that you need: **Android**, **iOS**, or **Java ME**.
  4. Click **Finish**.

     You created a Worklight project in Worklight Studio, with a first Native API application in it.

In both cases, you created the required Native API application in Worklight Studio. This application contains:

- The application descriptor file: This file is the `application-descriptor.xml` file that is in the application root directory.
- The IBM Worklight native library and the client property file: The name and the format of this content depend on the environment.
  – for iOS:
    - The `WorklightAPI` folder defines the IBM Worklight native library.
    - The `worklight.plist` file is the client property file.
  – for Android:
    - The `worklight-android.jar` file defines the IBM Worklight native library.
    - The `wlclient.properties` file is the client property file.
  – for Java ME:
    - The `worklight-javame.jar` file and the `json4javame.jar` file together define the IBM Worklight native library.
    - The `wlclient.properties` file is the client property file.

Experimental IBM Worklight offline user documentation.
This document is provided "as is".
In case of issues refer to the online user documentation.

As a difference from a hybrid app, which you can develop entirely within Worklight Studio, you also generally need another project to develop your native app. For example:

- A project in the Xcode IDE, to develop a native application with Objective-C for iOS environment
- A project in the Eclipse IDE, to develop a native application with Java, for Android environment or for Java ME

After the Native API application is created, you must:

1. Define the various aspects of your application by setting the appropriate values in the application descriptor file.
2. Update the client property file, as needed.
3. Copy the client property file and the native library into the appropriate location of your native project. You must also create references from your native app project to this content to use the IBM Worklight native API.

- For iOS:
  1. To update the application descriptor file, see "Application Descriptor of Native API applications for iOS."
  2. To update the client property file, see "Client property file for iOS" on page 88.
  3. To copy the client property file and the native library into the appropriate location of your native project, and create appropriate references, see "Copying files of Native API applications for iOS" on page 88.
- For Android:
  1. To update the application descriptor file, see "Application Descriptor of Native API application for Android" on page 89.
  2. To update the client property file, see "Client property file for Android" on page 91.
  3. To copy the client property file and the native library into the appropriate location of your native project, and create appropriate references, see "Copying files of Native API applications for Android" on page 92.
- For Java ME:
  1. To update the application descriptor file, see "Application Descriptor of Native API application for Java Platform, Micro Edition (Java ME)" on page 92.
  2. To update the client property file, see "Client property file for Java Platform, Micro Edition (Java ME)" on page 93.
  3. To copy the client property file and the native library into the appropriate location of your native project, and create appropriate references, see "Copying files of Native API applications for Java Platform, Micro Edition (Java ME)" on page 94.

You build and deploy Native API applications by following the same procedure as for hybrid applications, by creating the `.wlapp` file and uploading it to the Worklight Console. For more information about deployment, see "Deploying content: applications and adapters" on page 350.

Experimental IBM Worklight offline user documentation.
This document is provided "as is".
In case of issues, refer to the online user documentation.

Application Descriptor of Native API applications for iOS

The application descriptor is a metadata file that is used to define various aspects of the Native API application for iOS.

The application descriptor is a metadata file that is used to define various aspects of the application. It is in the application root directory, and its name is `application-descriptor.xml`.

The following example shows the format of the application descriptor file of Native API applications for iOS:

```
<?xml version="1.0" encoding="UTF-8"?>
<nativeIOSApp
    id="ios"
    platformVersion="5.0.5"
    version="1.0"
    securityTest="security test name"
    bundleId="com.ios"
    xmlns="http://www.worklight.com/native-ios-descriptor">
    <displayName>application display name</displayName>
    <description>application description</description>
    <pushSender password="${push.apns.senderpassword}"/>
</nativeiOSApp>
```

The content of the application descriptor file is as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<nativeIOSApp
    id="ios"
    platformVersion="5.0.5"
    version="1.0"
    securityTest="security test name"
    bundleId="com.ios"
    xmlns="http://www.worklight.com/native-ios-descriptor">
```

The `<nativeIOSApp>` element is the root element of the descriptor. It has three mandatory attributes and two optional attributes:

`id`
This attribute specifies the ID of the application. The ID must be identical to the application folder name. It must be an alphanumeric string that starts with a letter. It can contain underscore ("_") characters. It must not be a reserved word in JavaScript.

`platformVersion`
Contains the version of the IBM Worklight Platform on which the app was developed.

`version`
This attribute specifies the version of the application. This version is a string of the form x.y, where x and y are numbers. It is visible to users who download the app from the app store or market.

`securityTest`
This attribute specifies a security configuration that is defined in the `authenticationConfig.xml` file. When a client attempts to access a protected resource, IBM Worklight checks whether the client is already authenticated according to the security test. If the client is not yet authenticated, IBM Worklight starts the process to obtain the client credentials and to verify them.

This attribute is **optional**.

`bundleId`
This attribute specifies the bundle ID of the application.

This attribute is **optional**.

```
    <displayName>application display name</displayName>
```

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

<displayName>
This element contains the application name. This name is visible in the IBM Worklight Console and is copied to the descriptor files of various web and desktop environments.

```
<description>application description</description>
```

<description>
This element contains the application description. This description is visible in the IBM Worklight Console and is copied to the descriptor files of various web and desktop environments.

```
<pushSender password="${push.apns.senderpassword}"/>
```

<pushSender>
This element defines the password to the SSL certificate that encrypts the communication link with the Apple Push Notification Service (APNS).

```
</nativeiOSApp>
```

</nativeIOSApp>
This tag closes the content of the application descriptor file.

Client property file for iOS

This file defines the properties that your native app for iOS requires to use the IBM Worklight native API for iOS.

The `worklight.plist` client property file contains the necessary information for initializing `WLClient`.

You must define the properties of this client property file before using it in your native app for iOS.

The following table lists the properties of the `worklight.plist` file, their descriptions, and possible examples for their values.

Table 5. Properties of the `worklight.plist` file

Property	Description
protocol	The communication protocol with the Worklight Server: `http` or `https`.
host	The host name of the Worklight Server.
port	The port of the IBM Worklight Server. If this value is left blank, the default port is used. If the **protocol** property value is `https`, you must leave this value blank.
context	The server URL context.
application id	The application ID, as defined in the `application-descriptor.xml` file.
application version	The application version, as defined in the `application-descriptor.xml` file.
environment	This property defines the IBM Worklight environment. The value of this property must be `iosnative`. You must not modify the value of this property value.

Copying files of Native API applications for iOS

To copy the files in the Native API application for iOS into the project that defines the native app for iOS

Experimental IBM Worklight offline user documentation
This document is provided "as-is".
In case of issues, refer to the online user documentation.

About this task

To use the IBM Worklight Native API for iOS in your native app, you must copy the library and the client property file of your Native API application into your native app for iOS project.

Procedure

In Worklight Studio:

1. Select the `WorklightAPI` folder and the `worklight.plist` file of your Native API application, and copy them in a location that you can access from your native iOS project

In your project for the native app for iOS (for example, in Xcode IDE):

2. Add the `WorklightAPI` folder and the `worklight.plist` file of your Native API application to your project.
   a. In the **Choose options for adding these files**window, select the **Copy items into destination group's folder (if needed)** check box and the **Create groups for any added folders** option.
3. In the **Build Phases** tab, link the following frameworks and libraries to your project:
   - `CFNetwork`
   - `SystemConfiguration`
   - `MobileCoreServices`
   - `CoreData`
   - `Security`
   - `zlib`
4. Select the project name and the target for your application.
5. Click the **Build Phases** tab.
6. In the **Build Phases** tab:
   a. Open the list in the **Link Binary with Libraries** section, and make sure that `libWorklightStaticLibProjectNative.a` is visible in the list.
   b. Open the list in the **Copy Bundle Resources** section, and make sure that the files from your resources folder are added to that section.
7. Click the **Build Settings** tab.
8. In the **Build Settings** tab:
   a. Add the following entry: `$(SRCROOT)/WorklightSDK/include for HEADER_SEARCH_PATH`
   b. In the **Other Linker Flags** field, enter the following value: `–ObjC`

Application Descriptor of Native API application for Android

The application descriptor is a metadata file that is used to define various aspects of the Native API application for Android.

The application descriptor is a metadata file that is used to define various aspects of the application. It is in the application root directory, and its name is `application-descriptor.xml`.

The following example shows the format of the application descriptor file of Native API applications for Android:

Experimental IBM Worklight offline user documentation
This document is provided "as-is".
In case of issues, refer to the online user documentation.

```
<?xml version="1.0" encoding="UTF-8"?>
<nativeAndroidApp
    id="android"
    platformVersion="5.0.5"
    securityTest="security test name"
    version="1.0"
    xmlns="http://www.worklight.com/native-android-descriptor">
    <displayName>application display name</displayName>
    <description>application description</description>
    <pushSender key="gcm api key" senderId="gcm project number"/>
    <publicSigningKey>application public signing key</publicSigningKey>
</nativeAndroidApp>
```

The content of the application descriptor file is as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<nativeAndroidApp
    id="android"
    platformVersion="5.0.5"
    securityTest="security test name"
    version="1.0"
    xmlns="http://www.worklight.com/native-android-descriptor">
```

The `<nativeAndroidApp>` element is the root element of the descriptor. It has three mandatory attributes and one optional attribute:

`id`
This attribute specifies the ID of the application. The ID must be identical to the application folder name. It must be an alphanumeric string that starts with a letter. It can contain underscore ("_") characters. It must not be a reserved word in JavaScript.

`platformVersion`
Contains the version of the IBM Worklight Platform on which the app was developed.

`version`
This attribute specifies the version of the application. This version is a string of the form x.y, where x and y are numbers. It is visible to users who download the app from the app store or market.

`securityTest`
This attribute specifies a security configuration that is defined in the `authenticationConfig.xml` file. When a client attempts to access a protected resource, IBM Worklight checks whether the client is already authenticated according to the security test. If the client is not yet authenticated, IBM Worklight starts the process to obtain the client credentials and to verify them.

This attribute is **optional**.

```
<displayName>application display name</displayName>
```

`<displayName>`
This element contains the application name. This name is visible in the IBM Worklight Console and is copied to the descriptor files of various web and desktop environments.

```
<description>application description</description>
```

`<description>`
This element contains the application description. This description is visible in the IBM Worklight Console and is copied to the descriptor files of various web and desktop environments.

```
<pushSender key="gcm api key" senderId="gcm project number"/>
```

Experimental IBM Worklight online user documentation.
This document is provided "as is".
In case of issues, refer to the online user documentation.

<pushSender>

This element contains the connectivity details to Google GCM (Android push notification service). The `key` is the GCM API key, and the `senderId` is the GCM Project Number.

```
<publicSigningKey>application public signing key</publicSigningKey>
```

<publicSigningKey>

This element contains the public key of the developer certificate that is used to sign the Android app. For instructions on how to extract this value, see "Extracting a public signing key" on page 74.

```
</nativeAndroidApp>
```

</nativeAndroidApp>

This tag closes the content of the application descriptor file.

Client property file for Android

This file defines the properties that your native app for Android requires to use the IBM Worklight native API for Android.

The `wlclient.properties` client property file contains the necessary data to use the IBM Worklight API for Android.

You must define the properties of this client property file before you use it in your native app for Android.

The following table lists the properties of the `wlclient.properties` file, their descriptions, and possible examples for their values.

Table 6. Properties and values of the `wlclient.properties` file

Property	Description	Example values
wlServerProtocol	The communication protocol with the Worklight Server.	`http` or `https`
wlServerHost	The host name of the Worklight Server.	`localhost`
wlServerPort	The port of the IBM Worklight Server. If you leave this value blank, the default port is used. If the **wlServerProtocol** property value is `https`, you must leave this value blank.	`8080`
wlServerContext	The server context, which is automatically generated.	`/`
wlAppId	The application id, as defined in the `application-descriptor.xml` file.	`myApp`
wlAppVersion	The application version, as defined in the `application-descriptor.xml` file.	`1.0`
wlEnvironment	This property defines the IBM Worklight environment. You must not modify the value of this property value.	`Androidnative`
GCMSenderID	This property defines the GCM Sender ID that you must use for push notifications. By default, this property is commented.	

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

Copying files of Native API applications for Android

To copy the files in the Native API application for Android into the project that defines the native app for Android

About this task

To use the IBM Worklight Native API for Android in your native app, you must copy the library and the client property file of your Native API application into your native app for Android project.

Procedure

In your project for the native app for Android:

1. Copy the `worklight-android.jar` file from the Native API application, and paste it into the `libs` folder of your native app for Android.
2. Copy the `wlclient.properties` client property file from the Native API application into the `assets` folder of your native app for Android.
3. If the push notification support is required:
   a. Copy the `gcm.jar` file from the Native API application.
   b. Paste the `gcm.jar` into the `libs` folder of your native app for Android.
   c. Copy the `push.png` file from the Native API application.
   d. In the `res` folder of your native app for Android, identify the folders with a name that starts with `drawable` (such as `res/drawable` or `res/drawable-ldpi`), and then paste the `push.png` file into each of these folders.
4. Add the following lines to the `AndroidManifest.xml` file of your native app for Android:
   a. `<activity android:name="com.worklight.wlclient.ui.UIActivity"/>` With this line, a designated IBM Worklight UI activity can run in the user application.
   b. `<uses-permission android:name="android.permission.INTERNET"/>` This line adds Internet access permissions to the user application.

Application Descriptor of Native API application for Java Platform, Micro Edition (Java ME)

The application descriptor is a metadata file that is used to define various aspects of the Native API application for Java ME.

The application descriptor is a metadata file that is used to define various aspects of the application. It is in the application root directory, and its name is `application-descriptor.xml`.

The following example shows the format of the application descriptor file of Native API applications for Java ME:

```
<?xml version="1.0" encoding="UTF-8"?>
<nativeJavaMEApp
    id="JavaME"
    platformVersion="5.0.5"
    version="1.0"
    securityTest="security test name"
```

Experimental IBM Worklight offline user documentation
In case of issues, refer to the online user documentation.
This document is provided "as-is".

```
    xmlns="http://www.worklight.com/native-javame-descriptor">
    <displayName>application display name</displayName>
    <description>application description</description>
</nativeJavaMEApp>
```

The content of the application descriptor file is as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<nativeJavaMEApp MEApp
    id="JavaME"
    platformVersion="5.0.5"
    version="1.0"
    securityTest="security test name"
    xmlns="http://www.worklight.com/native-javame-descriptor">
```

The `<nativeJavaMEApp>` element is the root element of the descriptor. It has three mandatory attributes and one optional attribute:

id
This attribute specifies the ID of the application. The ID must be identical to the application folder name. It must be an alphanumeric string that starts with a letter. It can contain underscore ("_") characters. It must not be a reserved word in JavaScript.

platformVersion
Contains the version of the IBM Worklight Platform on which the app was developed.

version
This attribute specifies the version of the application. This version is a string of the form x.y, where x and y are numbers. It is visible to users who download the app from the app store or market.

securityTest
This attribute specifies a security configuration that is defined in the `authenticationConfig.xml` file. When a client attempts to access a protected resource, IBM Worklight checks whether the client is already authenticated according to the security test. If the client is not yet authenticated, IBM Worklight starts the process to obtain the client credentials and to verify them.

This attribute is **optional**.

```
    <displayName>application display name</displayName>
```

<displayName>
This element contains the application name. This name is visible in the IBM Worklight Console and is copied to the descriptor files of various web and desktop environments.

```
    <description>application description</description>
```

<description>
This element contains the application description. This description is visible in the IBM Worklight Console and is copied to the descriptor files of various web and desktop environments.

```
</nativeJavaMEApp>
```

</nativeJavaMEApp>
This tag closes the content of the application descriptor file.

Client property file for Java Platform, Micro Edition (Java ME)

This file defines the properties that your native app for Java Platform, Micro Edition (Java ME) requires to use the IBM Worklight native API for Java ME.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issue, refer to the online user documentation.

The `wlclient.properties` client property file contains the necessary data to use the IBM Worklight API for Java ME.

You must define the properties of this client property file before using it in your native app for Java ME.

The following table lists the properties of the `wlclient.properties` file, their descriptions, and possible examples for their values.

Table 7. Properties and values of the `wlclient.properties` file

Property	Description	Example values
`wlServerProtocol`	The communication protocol with the Worklight Server.	`http` or `https`
`wlServerHost`	The host name of the Worklight Server.	`localhost`
`wlServerPort`	The port of the IBM Worklight Server.	`8080`
`wlServerContext`	The server context, which is automatically generated.	`/`
`wlAppId`	The application ID, as defined in the `application-descriptor.xml` file.	`myApp`
`wlAppVersion`	The application version, as defined in the `application-descriptor.xml` file.	`1.0`
`wlEnvironment`	This property defines the IBM Worklight environment. You must not modify the value of this property value.	`JavaMEnative`

Copying files of Native API applications for Java Platform, Micro Edition (Java ME)

To copy the files in the Native API application for Java ME into the project that defines the app for Java ME.

About this task

To use the IBM Worklight Native API for Java ME in your native app, you must copy the library and the client property file of your Native API application into your native app for Java ME project.

Procedure

1. Create a `lib` folder in your native Java ME application.

   Note: You can name this folder differently. If you select a folder name other than `lib`, ensure that you use this folder name instead of `lib` in the following steps.
2. Make sure that the build path of your native Java ME application includes this `lib` folder.
3. Copy the `worklight-javame.jar` file of your Native API application into this `lib` folder of your native Java ME application.
4. Copy the `json4javame.jar` file of your Native API application into this `lib` folder of your native Java ME application.
5. Copy the `wlclient.properties` file of your Native API application into the `res` folder of your native Java ME application.

Experimental IBM Worklight online user documentation.
This document is provided "as is".
In case of issues, refer to the online user documentation.

Developing the server side of an IBM Worklight application

This collection of topics relates to various aspects of developing the server-side components of a Worklight application.

Overview of IBM Worklight adapters

Adapters run on the server and connect to mobile apps.

Adapters are the server-side code of applications that are deployed on and serviced by the IBM Worklight Mobile Application Platform. Adapters connect to enterprise applications (otherwise referred to as *back-end systems*), deliver data to and from mobile applications, and perform any necessary server-side logic on this data.

Benefits of IBM Worklight adapters

Adapters provide various benefits, as follows:

- **Fast Development**: Adapters are developed in JavaScript and XSL. Developers employ flexible and powerful server-side JavaScript to produce succinct and readable code for integrating with back-end applications and processing data. Developers can also use XSL to transform hierarchical back-end data to JSON.
- **Read-only and Transactional Capabilities**: IBM Worklight adapters support read-only and transactional access modes to back-end systems.
- **Security**: IBM Worklight adapters use flexible authentication facilities to create connections with back-end systems. Adapters offer control over the identity of the user with whom the connection is made. The user can be a *system* user, or a user on whose behalf the transaction is made.
- **Transparency**: Data retrieved from back-end applications is exposed in a uniform manner, so that application developers can access data uniformly, regardless of its source, format, and protocol.

The adapter framework

The adapter framework mediates between the mobile apps and the back-end services. A typical flow is depicted in the following diagram. The app, the back-end application, and the JavaScript code and XSLT components in the Worklight Server are supplied by the adapter or app developer. The procedure and auto-conversions are part of the IBM Worklight Platform.

Experimental IBM Worklight user documentation
This documentation is provided "as is".
In case of issues, refer to the official user documentation.

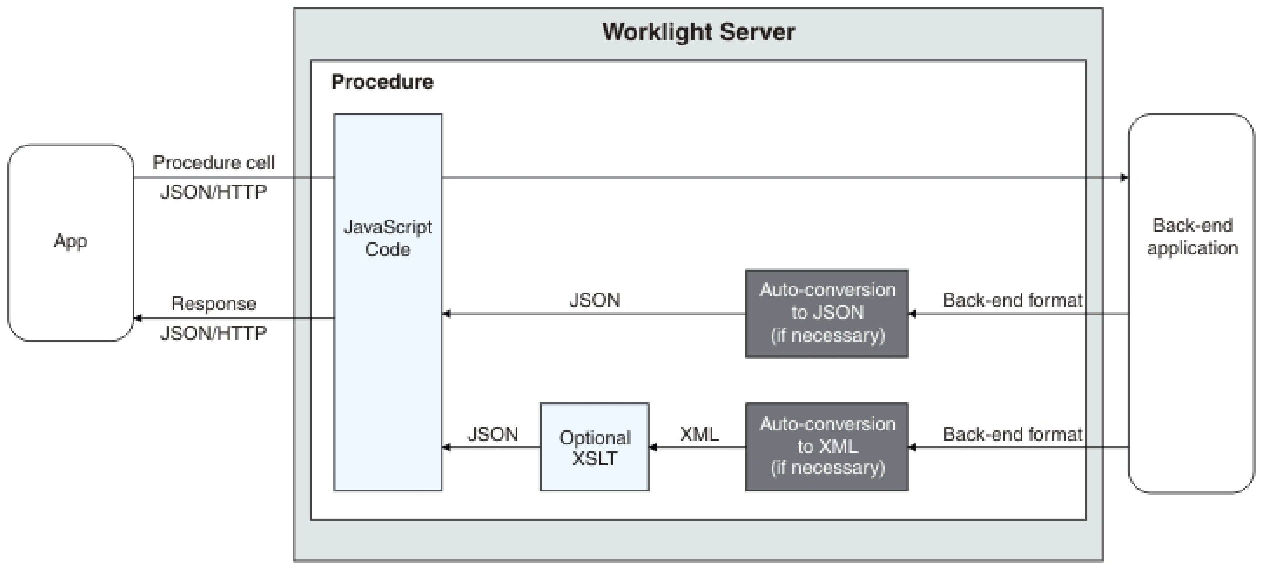


Figure 12. The adapter framework

1. An adapter exposes a set of services, called procedures. Mobile apps invoke procedures by issuing Ajax requests.
2. The procedure retrieves information from the back-end application.
3. The back-end application then returns data in some format.
   - If this format is JSON, the IBM Worklight Server keeps the data intact.
   - If this format is not JSON, the IBM Worklight Server automatically converts it to JSON. Alternatively, the developer can provide an XSL transformation to convert the data to JSON. In such a case, the IBM Worklight Server first converts the data to XML (if it is not in XML already) that serves as input for the XSL transformation.
4. The JavaScript implementation of the procedure receives the JSON data, performs any additional processing, and returns it to the calling app.

HTTP POST requests are used for client-server communications between the Worklight application and the Worklight server. Parameters must be supplied in a plain text or numeric format. To transfer images (or any other type of file data), they must be converted to base64 first.

Anatomy of adapters

IBM Worklight adapters are developed by using XML, JavaScript, and XSL. Each adapter must have the following elements:

- Exactly one XML file, describing the connectivity to the back-end system to which the adapter connects, and listing the procedures that are exposed by the adapter to other adapters and to applications.
- Exactly one JavaScript file, containing the implementation of the procedures declared in the XML file.
- Zero or more XSL files, each containing a transformation from the raw XML data retrieved by the adapter to JSON returned by adapter procedures.

The files are packaged in a compressed file with a `.adapter` suffix (such as `myadapter.adapter`).

The root element of the XML configuration files is <adapter>. The main subelements of the <adapter> element are as follows:

- <connectivity>: Defines the connection properties and load constraints of the back-end system. When the back-end requires user authentication, this element defines how the credentials are obtained from the user.
- <procedure>: Declares a procedure that is exposed by the adapter.

The structure of the <adapter> element is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<wl:adapter>
<description>
<connectivity>
<connectionPolicy>
<loadConstraints>
</connectivity>

<procedure />  <!-- One or more such elements -->
</wl:adapter>
```

The HTTP adapter

The IBM Worklight HTTP adapter can be used to invoke RESTful services and SOAP-based services. It can also be used to perform HTML scraping.

You can use the HTTP adapter to send GET, POST, PUT, and DELETE HTTP requests and retrieve data from the response body. Data in the response can arrive in XML, HTML, or JSON formats.

You can use SSL in an HTTP adapter with simple and mutual authentication to connect to back-end services. Configure the IBM Worklight Server to use SSL in an HTTP adapter by implementing the following steps:

- Set the URL protocol of the HTTP adapter to https.
- Store SSL certificates in a keystore that is defined in the worklight.properties file. The keystore setup process is described in "SSL certificate keystore setup" on page 409.
- If you use SSL with mutual authentication, the following extra steps must also be implemented:
  - Generate your own private key for the HTTP adapter or use one provided by a trusted authority.
  - If you generated your own private key, export the public certificate of the generated private key and import it into the back-end truststore.
  - Save the private key of the keystore that is defined in the worklight.properties file.
  - Define an alias and password for the private key in the <connectionPolicy> element of the HTTP adapter XML file, *adaptername*.xml. The <sslCertificateAlias> and <sslCertificatePassword> subelements are described in "The <connectionPolicy> element of the HTTP adapter" on page 103.

Note however that SSL represents transport level security, which is independent of basic authentication. It is possible to do basic authentication either over HTTP or HTTPS.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

The SQL adapter

You can use the IBM Worklight SQL adapter to execute parameterized SQL queries and stored procedures that retrieve or update data in the database.

The Cast Iron adapter

The IBM Worklight Cast Iron adapter initiates orchestrations in Cast Iron to retrieve and return data to mobile clients.

Cast Iron accesses various enterprise data sources, such as databases, web services, and JMS, and provides validation, aggregation, and formatting capabilities.

The Cast Iron adapter supports two patterns of connectivity:

Outbound pattern.
The invocation of Cast Iron orchestrations from Worklight.

Inbound pattern.
Cast Iron sends notifications to devices through Worklight.

The Cast Iron adapter supports the invocation of a Cast Iron orchestration over HTTP only. Cast Iron Template Integration Projects (TIPs) are provided in Cast Iron as examples of this technique, and for you to use as a basis for your own orchestrations. For more information, see the Cast Iron documentation.

Cast Iron uses the standard IBM Worklight notification adapter and event sources to publish notification messages to be delivered to devices by using one of the many notification providers.

For information about defining event sources, see "Method WL.Server.createEventSource" on page 286.

Cast Iron Template Integration Projects (TIPs) are provided in Cast Iron as examples of this technique, and for you to use as a basis for your own notification scenarios. For more information, see the Cast Iron documentation.

To protect the notification adapter, use basic authentication.

The JMS adapter

The IBM Worklight JMS adapter can be used to send and receive messages from a JMS-enabled messaging provider. It can be used to send and receive the headers and body of the messages.

Troubleshooting a Cast Iron adapter – connectivity issues

Symptom: The IBM Worklight adapter cannot communicate with the Cast Iron server.

Causes:

- Cast Iron provides two network interfaces, one for administration and one for data. Ensure that you are using the correct host name or IP address of the Cast Iron data interface. You can find this information under the Network menu item in the Cast Iron administrative interface. This information is stored in the *adapter-name*`.xml` file for your adapter.

Experimental IBM Worklight online user documentation
This document is provided "as is".
In case of issues refer to the online documentation.

- The invocation fails with a message `Failed to parse the payload from backend`. This failure is typically caused by a mismatch between the data returned by the Cast Iron orchestration and the `returnedContentType` parameter in the *`adapter-name`*`.js` implementation. For example, the Cast Iron orchestration returns JSON but the adapter is configured to expect XML.

The adapter XML File

The adapter XML file is used to configure connectivity to the back-end system and to declare the procedures exposed by the adapters to applications and to other adapters.

The root element of the document is `<adapter>`.

- For elements whose content is the same for all types of back-end application, this section contains complete details of the tag content.
- For elements whose content is different for different types of back-end applications, this section contains a general description of the content of the elements. Full details of the content can be found in the topic that describes the specific adapter.

<adapter> element of the adapter XML file

The <adapter> element is the root element and has various attributes and subelements.

The `<adapter>` element is the root element of the adapter configuration file. It has the following structure:

```
<wl:adapter
name="adapter-name"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wl="http://www.worklight.com/integration"
xmlns:http="http://www.worklight.com/integration/http"
xmlns:sql="http://www.worklight.com/integration/sql"
xsi:schemaLocation="
http://www.worklight.com/integration integration.xsd
http://www.worklight.com/integration/http http.xsd
http://www.worklight.com/integration/sql sql.xsd
>
```

IBM Worklight provides two schemas that are used by all adapters, and in addition, provides a specific schema for each type of adapter. Each schema must be associated with a different namespace. Namespaces are declared using the `xmlns` attribute, and are linked to their schemas by using the `xsi:schemaLocation` attribute.

The mandatory schemas are `http://www.w3.org/2001/XMLSchema-instance`, which is associated with the `xsi` namespace, and `http://www.worklight.com/integration`, which is associated with the `wl` namespace.

Because each adapter connects to a single back-end application and uses a single integration technology, each adapter only requires one back-end-related namespace. For example, for an HTTP adapter you must declare the `xmlns:http` namespace and associate it with the `http.xsd` schema.

The `<adapter>` element has the following attributes:

Experimental IBM offline user documentation
This document is provided "as is".
In case of issue, refer to the online user documentation.

Table 8. <adapter> element attributes

Attribute	Description
name	Mandatory. The name of the adapter. This name must be unique within the Worklight Server. It can contain alphanumeric characters and underscores, and must start with a letter. **Note:** After an adapter has been defined and deployed, its name cannot be modified.
xmlns:*namespace*	Mandatory. Defines schema namespaces. This attribute must appear three times, as follows: `xmlns:xsi` – Defines the namespace associated with the `http://www.w3.org/2001/XMLSchema-instance` schema. `xmlns:wl` – Defines the namespace associated with the `http://www.worklight.com/integration` schema. `xmlns:`*`namespace`* – Defines the namespace associated with the schema related to the back-end application, for example, `xmlns:sap` or `xmlns:sql`.
xsi:schemaLocation	Optional. Identifies the schema locations, in the following format: `xsi:schemaLocation="http://www.worklight.com/integration` If the attribute is missing, auto-complete for XML elements and attributes defined in the schema will not be available in the IBM Worklight Studio. at run time, this attribute has no effect.

The `<adapter>` element has the following subelements:

Table 9. <adapter> element subelements

Subelement	Description
<displayName>	**Note**: This element is deprecated. Optional. The name of the adapter to be displayed in the IBM Worklight Console. If the `<displayName>` element is not specified, the value of the `name` attribute is used instead in the IBM Worklight Console.
<description>	Optional. Additional information about the adapter, which is displayed in the IBM Worklight Console.
<connectivity>	Mandatory. Defines the connection properties and load constraints of the back-end system. For more information, see "<connectivity> element of the adapter XML file" on page 101.

Experimental IBM Worklight offline user documentation.
This document is provided as is.
In case of issues, refer to the online documentation.

Table 9. <adapter> element subelements (continued)

Subelement	Description
<procedure>	Mandatory. Defines a process for accessing a service exposed by a back-end application. Occurs once for each procedure exposed by the adapter. For more information, see "<procedure> element of the adapter XML file" on page 102.

<connectivity> element of the adapter XML file

The `<connectivity>` element defines the mechanism by which the adapter connects to the back-end application.

It has the following subelements:

Table 10. <connectivity> element subelements

Subelement	Description
`<connectionPolicy>`	Mandatory. Defines back-end-specific connection properties.
`<loadConstraints>`	Mandatory. Defines the number of concurrent connections which the IBM Worklight Server can open to the back end.

<connectionPolicy> element of the adapter XML file

The `<connectionPolicy>` element defines connection properties.

The structure of the `<connectionPolicy>` element depends on the integration technology of the back-end application. For more information, see the related links.

Related reference:

"The <connectionPolicy> element of the HTTP adapter" on page 103
The structure of the `<ConnectionPolicy>` element.

"The <connectionPolicy> element of the SQL adapter" on page 107
The connection policy of an SQL adapter is configured in two places: the `worklight.properties` file and the adapter XML configuration file.

"The <connectionPolicy> element of the JMS adapter" on page 109
The structure of the `<connectionPolicy>` element.

<loadConstraints> element of the adapter XML file

The `<loadConstraints>` element defines the maximum load that is exerted on a back-end application by setting the maximum number of concurrent requests that can be performed on the system.

IBM Worklight queues incoming service requests from IBM Worklight applications. While the number of concurrent requests is below the maximum, IBM Worklight forwards the requests to the back-end application according to their order in the queue. If the number of concurrent requests is above the maximum, IBM Worklight waits until an already handled request is finished, before it services the next one in the queue. If a request waits in the queue for longer than the timeout configured in the procedure, IBM Worklight removes it from the queue, and returns a `Request Timed Out` exception to the caller.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online documentation.

The `<loadConstraints>` element has the following attributes:

Attribute	Description
maxConcurrentConnectionsPerNode	Mandatory. The maximum number of concurrent requests that can be performed per server node of the back-end application.

Consider a case where the back-end application must serve about 100 transactions per second, and where each transaction takes an average response time of 2 seconds. The back-end application defines four server nodes to manage these requests. Each node must thus be able to manage an average of 50 transactions per second (100 x 2 / 4). To properly communicate with this back-end application, you must then set the value of the **maxConcurrentConnectionsPerNode** attribute to at least 50.

```
<loadConstraints maxConcurrentConnectionsPerNode="50" />
```

Note: If you increase the value of this attribute, the back-end application needs more memory. Do not set this value too high to avoid memory issues. Instead, estimate the average and peak number of transactions per second, and evaluate their average durations. Then, calculate the number of required concurrent connections as indicated in this example, and add a 5-10 margin to define the value of this attribute. Then, monitor your server, and adjust this value as required, to ensure that you back-end application can process all incoming requests.

For more information about how to size your back-end application, see:
- Scalability and Hardware Sizing (PDF)
- Hardware Calculator (XLS)

<procedure> element of the adapter XML file

The `<procedure>` element defines a process for accessing a service exposed by a back-end application.

The service can retrieve data from the back end or perform a transaction at the back end.

The `<procedure>` element has the following structure:

```
<procedure
name="unique-name"
connectAs="value"
requestTimeoutInSeconds="value"
audit="value"
securityTest="value"
/>
```

The `<procedure>` element has the following attributes:

Table 11. <procedure> element attributes

Attribute	Description
name	Mandatory. The name of the procedure. This name must be unique within the adapter. It can contain alphanumeric characters and underscores, and must start with a letter.

Experimental IBM Worklight offline user documentation. This document is provided "as-is". In case of issues, refer to the online user documentation.

Table 11. <procedure> element attributes (continued)

Attribute	Description
connectAs	Optional. Defines how to create a connection to the back end for invoking the retrieve procedure. Valid values are as follows: `server`: Default. The connection to the back end is created according to the connection policy defined for the adapter. For information about configuring connection policies, see the related links. `endUser`: The connection to the back end is created with the user's identity. Only valid if a user realm has been identified in the security tests for this procedure.
requestTimeoutInSeconds	Optional. The timeout in seconds for waiting until receiving a response from the back end, including the time for opening the connection. The default is 30 seconds.
audit	Optional. Defines whether calls to the procedure are logged in the audit log. The log file is *`Worklight Project Name`*`/server/log/audit/audit.log`. Valid values are as follows: `true`: Calls to the procedure are logged in the audit log. `false`: Default. Calls to the procedure are not logged in the audit log.
securityTest	Optional. The name of the security test that you want to use to protect the adapter procedure, as defined in the `authenticationConfig.xml` file.

The root element of the HTTP adapter XML file

The structure of the root element.

The root element of the HTTP adapter has the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<wl:adapter>
name="adapter-name"
authenticationRealm="realm"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wl="http://www.worklight.com/integration"
xmlns:http="http://www.worklight.com/integration/http"
xsi:schemaLocation=
"http://www.worklight.com/integration integration.xsd
http://www.worklight.com/integration/http http.xsd">
...
</wl:adapter>
```

The <connectionPolicy> element of the HTTP adapter

The structure of the `<ConnectionPolicy>` element.

The `<ConnectionPolicy>` element has the following structure:

```
<connectionPolicy xsi:type="http:HTTPConnectionPolicyType"
cookiePolicy="cookie-policy" maxRedirects="int">
<protocol>protocol</protocol>
```

Experimental IBM Worklight offline user documentation
This document is provided as is. In case of issues, refer to the online user documentation.

```
<domain>host-name</domain>
<port>host-port</port>
<sslCertificateAlias>ssl-certificate-alias</sslCertificateAlias>
<sslCertificatePassword>ssl-certificate-password</sslCertificatePassword>
<authentication> ... </authentication>
<proxy> ... </proxy>
</connectionPolicy>
```

The `<ConnectionPolicy>` element has the following attributes:

Table 12. <ConnectionPolicy> element attributes

Attribute	Description
xsi:type	Mandatory. The value of this attribute must be set to `http:HTTPConnectionPolicyType`.
cookiePolicy	Optional. This attribute sets how the HTTP adapter handles cookies that arrive from the back-end application. Valid values are as follows: • RFC_2109 (The default) • RFC_2965 • NETSCAPE • IGNORE_COOKIES
maxRedirects	Optional. The maximum number of redirects that the HTTP adapter can follow. This attribute is useful when the back-end application sends circular redirects as a result of some error, such as authentication failures. The default value is 20.

The `<ConnectionPolicy>` element has the following subelements:

Table 13. `<ConnectionPolicy>` element subelements

Subelement	Description
protocol	Optional. The URL protocol to use. Possible values are `http` (default) and `https`.
domain	Mandatory. The host address.
port	Optional. The port address. The default value is 80.
sslCertificateAlias	The alias of the adapter private SSL key, which is used by the HTTP adapter key manager to access the correct SSL certificate in the keystore. Optional for regular HTTP authentication and simple SSL authentication. Mandatory for mutual SSL authentication. The keystore setup process is described in "SSL certificate keystore setup" on page 409

Experimental IBM Worklight offline user documentation
This document is provided as is.
In case of issues, refer to the online user documentation.

Table 13. `<ConnectionPolicy>` element subelements (continued)

Subelement	Description
sslCertificatePassword	The password of the adapter private SSL key, which is used by the HTTP adapter key manager to access the correct SSL certificate in the keystore. Optional for regular HTTP authentication and simple SSL authentication. Mandatory for mutual SSL authentication. The keystore setup process is described in "SSL certificate keystore setup" on page 409
authentication	Optional. Authentication configuration of the HTTP adapter. See "The <authentication> element of the HTTP adapter."
proxy	Optional. Used when the back-end application must be accessed through a proxy. See "The <proxy> element of the HTTP adapter" on page 106.

The <authentication> element of the HTTP adapter

The HTTP adapter can use one of four protocols, and can also contain a server identity.

You can configure the HTTP adapter to use one of four authentication protocols by defining the `<authentication>` element. You can define this element only within the `<connectionPolicy>` element. Depending on the authentication protocol that the HTTP adapter uses, among the following ones, define the `<authentication>` element as follows:

- Basic Authentication

```
<authentication>
  <basic/>
</authentication>
```

- Digest Authentication

```
<authentication>
  <digest/>
</authentication>
```

- NTLM Authentication

```
<authentication>
  <ntlm hostname="value"/>
</authentication>
```

  The `hostname` attribute is optional, and denotes the name of the computer on which the IBM Worklight Server runs. Its default value is `${local.hostname}`.

- SPNEGO/Kerberos Authentication

```
<authentication>
  <spnego stripPortOffServiceName="true"/>
</authentication>
```

  The attribute `stripPortOffServiceName` is optional, and specifies whether the Kerberos client uses the service name without the port number. The default value is `false`.

  When you use this option, you must also place the `krb5.conf` file under *`Worklight Project Name`*`/server/conf`. The file must contain Kerberos

Experimental IBM Worklight online documentation
This document is provided as a courtesy.
In case of issues, refer to the online user documentation.

configuration such as the location of the Kerberos server, and domain names. Its structure is described in the Kerberos V5 System Administrator's Guide in the mit.edu website.

Specifying the Server Identity

If the adapter exposes procedures with the attribute `connectAs="server"`, the connection policy can contain a `<serverIdentity>` element. This feature applies to all authentication schemes, for example:

```
<authentication>
  <basic/>
  <serverIdentity>
    <username> ${user} </username>
    <password> ${password} </password>
  </serverIdentity>
</authentication>
```

The <proxy> element of the HTTP adapter

Use a `<proxy>` element if you access an application through a proxy.

If the back-end application must be accessed through a proxy, add a `<proxy>` element inside the `<connectionPolicy>` element. If the proxy requires authentication, add a nested `<authentication>` element inside `<proxy>`. This element has the same structure as the one used to describe the authentication protocol of the adapter, described in "The <authentication> element of the HTTP adapter" on page 105.

The following example shows a proxy that requires basic authentication and uses a server identity:

```
<connectionPolicy xsi:type="http:HTTPConnectionPolicyType">
<protocol>http</protocol>
<domain>www.bbc.co.uk</domain>
<proxy>
<protocol>http</protocol>
<domain>wl-proxy</domain>
<port>8167</port>
<authentication>
<basic/>
<serverIdentity>
<username>${proxy.user}</username>
<password>${proxy.password}</password>
</serverIdentity>
</authentication>
</proxy>
</connectionPolicy>
```

The root element of the SQL adapter XML file

The structure of the root element.

The root element of the SQL adapter has the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<wl:adapter>
name="adapter-name"
authenticationRealm="realm"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wl="http://www.worklight.com/integration"
xmlns:http="http://www.worklight.com/integration/sql"
xsi:schemaLocation=
```

Experimental use of Worklight or released documentation. In case of issues, this document is provided "as is". Refer to the online documentation.

```
"http://www.worklight.com/integration integration.xsd
http://www.worklight.com/integration/sql sql.xsd">
...
</wl:adapter>
```

The <connectionPolicy> element of the SQL adapter

The connection policy of an SQL adapter is configured in two places: the `worklight.properties` file and the adapter XML configuration file.

Configuring the policy in the worklight.properties File

In the `worklight.properties` file, configure the properties described in the following table. In all properties, *n* can be 1 - 10, such as `custom-db.1.relative-jndi-name`.

Table 14. Properties in the `worklight.properties` file

Property	Description
custom-db.*n*.relative-jndi-name	The relative JNDI name of your database. This is the part of the JNDI name that is prefixed by `java:comp/env/`. For example, if the full JNDI name is `java:/comp/env/mydatabase`, the relative JNDI name is `mydatabase`.
custom-db-*n*.driver	A JDBC driver to your database. The JDBC driver of MySQL, for example, is `com.mysql.jdbc.Driver`. The driver must be placed in the *`Worklight Project Name`*`/server/lib` folder.
custom-db.*n*.url	The JDBC connection string to your database. For example: `jdbc:mysql://localhost:3306/mydatabase`.
custom-db.*n*.username	The user name used to connect to the database.
custom-db.*n*.password	The user password used to connect to the database.

Configuring the adapter XML file

In the adapter XML configuration file, configure the `<ConnectionPolicy>` element. The `<ConnectionPolicy>` element has the following structure:

```
<connectionPolicy xsi:type="sql:SQLConnectionPolicy">
<dataSourceJNDIName> java:comp/env/relative-jndi-name
</dataSourceJNDIName>
</connectionPolicy>
```

The `<ConnectionPolicy>` element has the following attribute:

Table 15. `<ConnectionPolicy>` element attribute

Attribute	Description
xsi:type	Mandatory. The value of this attribute must be set to `sql:SQLConnectionPolicy`.

The `<ConnectionPolicy>` element has the following subelement:

Experimental IBM Worklight offline user documentation. This document is provided "as-is". In case of issues, refer to the online user documentation.

Table 16. `<ConnectionPolicy>` element subelement

Subelement	Description
dataSourceJNDIName	Mandatory. The JNDI name (the data source name), as defined in the `worklight.properties` file.

In the reference example, the `<ConnectionPolicy>` element is defined as follows:

```
<connectionPolicy xsi:type="sql:SQLConnectionPolicy">
<dataSourceJNDIName>java:/comp/env/bankDS</dataSourceJNDIName>
</connectionPolicy>
```

The root element of the Cast Iron adapter XML file

Structure of the root element

The root element of the SQL adapter has the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<wl:adapter>
name="adapter-name"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wl="http://www.worklight.com/integration"
xmlns:http="http://www.worklight.com/integration/http"
xmlns:http="http://www.worklight.com/integration/ci"

</wl:adapter>
```

The <connectionPolicy> element of the Cast Iron adapter

Structure of the `<connectionPolicy>` element

The `<ConnectionPolicy>` element has the following structure:

```
<connectionPolicy xsi:type="http:HTTPConnectionPolicyType"
<protocol> protocol </protocol>
<domain> host-name </domain>
<port> host-port </port>
</connectionPolicy>
```

The `<ConnectionPolicy>` element has the following attributes:

Table 17. `<ConnectionPolicy>` element attributes

Attribute	Description
xsi:type	Mandatory. The value of this attribute must be set to `http:HTTPConnectionPolicyType`.

The `<ConnectionPolicy>` element has the following subelements:

Table 18. `<ConnectionPolicy>` element subelements

Subelement	Description
protocol	Optional. The URL protocol to use. Possible values are `http` (default) and `https`.
domain	Mandatory. The host address.
port	Optional. The port address. The default value is 80.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

The root element of the JMS adapter XML file

The structure of the root element of the JMS adapter.

The root element of the JMS adapter has the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<wl:adapter>
name="adapter-name"
authenticationRealm="realm"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wl="http://www.worklight.com/integration"
xmlns:jms="http://www.worklight.com/integration/jms"
xsi:schemaLocation=
"http://www.worklight.com/integration integration.xsd
http://www.worklight.com/integration/jms jms.xsd">
...
</wl:adapter>
```

The <connectionPolicy> element of the JMS adapter

The structure of the `<connectionPolicy>` element.

The `<connectionPolicy>` element has the following structure:

```
<connectionPolicy xsi:type="jms:JMSConnectionPolicyType">

<!-- optional jndi repository connection details -->
<namingConnection
url="jndi repository url"
initialContextFactory="JMS provider initial context factory class name"
user="optional jndi repository connection user name"
password="optional jndi repository connection password">
<!-- end of optional jndi repository connection details -->

<jmsConnection
connectionFactory="jndi connection factory name"
user="messaging service connection user name"
password="messaging service connection password">
</connectionPolicy>
```

The `<connectionPolicy>` element has the following attributes:

Table 19. <connectionPolicy> element attributes

Attribute	Description
xsi:type	Mandatory. The value of this attribute must be set to `jms:JMSConnectionPolicyType`.

The `<connectionPolicy>` element has the following subelements:

Table 20. `<connectionPolicy>` element subelements

Subelement	Description
namingConnection	Optional. Describes how to connect to an external JNDI repository. Only used if the JNDI objects are not stored in the JEE server that the adapter is deployed in. See "The <namingConnection> element of the JMS adapter" on page 110.

Experimental IBM Worklight offline user documentation. This document is provided as is. In case of issues, refer to the online user documentation.

Table 20. *<connectionPolicy> element subelements (continued)*

Subelement	Description
jmsConnection	Mandatory. Describes the connection factory and optional security details used to connect to the messaging system. See "The <jmsConnection> element of the JMS adapter."

The <namingConnection> element of the JMS adapter

Use the `<namingConnection>` element to identify how the IBM Worklight server connects to an external repository.

The JMS Adapter uses administered objects that must be predefined in a JNDI repository. The repository can either be defined in the JEE server context or an external JNDI repository. If you use an external repository, specify the `<namingConnection>` element to identify how the Worklight server connects to the repository.

The `<namingConnection>` element has the following attributes:

Attribute	Description
url	Mandatory. The url of the external JNDI repository. For example: *iiop://localhost*. The url syntax is dependent on the JNDI provider.
initialContextFactory	Mandatory. The initialContextFactory class name of the JNDI provider. For example: *com.ibm.Websphere.naming.WsnInitialContextFactory*. The driver, and any associated files, must be placed in the `/server/lib` directory. If you develop in the Eclipse environment, the driver and associated files must be placed in the `/lib` directory. **Note:** If you develop for WebSphere Application Server with WebSphere MQ, do not add the WebSphere MQ Java archive (JAR) files to the `/lib` directory. If the WebSphere MQ JAR files are added, classloading problems will occur because the files already exist in the WebSphere Application Server environment.
user	Optional. User name of a user with authority to connect to the JNDI repository. If `user` is not specified, the default user name is *guest*.
password	Optional. Password for the user specified in the `user` attribute. If `user` is not specified, the default password is *guest*.

Experimental IBM Worklight offline user documentation. This document is provided as is. In case of issues, refer to the online user documentation.

The <jmsConnection> element of the JMS adapter

Use the `<jmsConnection>` element to identify how the IBM Worklight server connects to a messaging system.

The `<jmsConnection>` element has the following attributes:

Attribute	Description
connectionFactory	Mandatory. The name of the connection factory used when connecting to the messaging system. This is the name of the administered object in the JNDI repository. **Note:** If you are deploying in WebSphere Application Server, the connection factory must be a global JNDI object. The object must be addressed without the `java:comp/env` context. For example: `jms/MyConnFactory` and not `java:comp/env/jms/MyConnFactory`. However, if you are deploying in Tomcat, the connection factory must be addressed including the `java:/comp/env` context. For example: `java:comp/env/jms/MyConnFactory`.
user	Optional. User name of a user with authority to connect to the messaging system.
password	Optional. Password for the user specified in the `user` attribute.

Creating an IBM Worklight adapter

Follow these instructions to create an IBM Worklight Project and configure a new IBM Worklight adapter.

About this task

On initial creation of a new adapter, the IBM Worklight Studio automatically generates the default skeleton for the adapter with all the required properties, based on the type (HTTP, SQL, or JMS). You need only to modify the default skeleton to configure an adapter.

Procedure

1. Optional: Perform this step only if you have not already created a Worklight project. If you set up IBM Worklight shortcuts, right-click the Project Explorer perspective panel in Eclipse and click **New** > **Worklight Project**. Otherwise, click **New** > **Other**, then select **Worklight** > **Worklight Project** from the list of wizards and click **Next**.

Experimental IBM Worklight online user documentation. This document is provided as is. In case of issues, refer to the online user documentation.

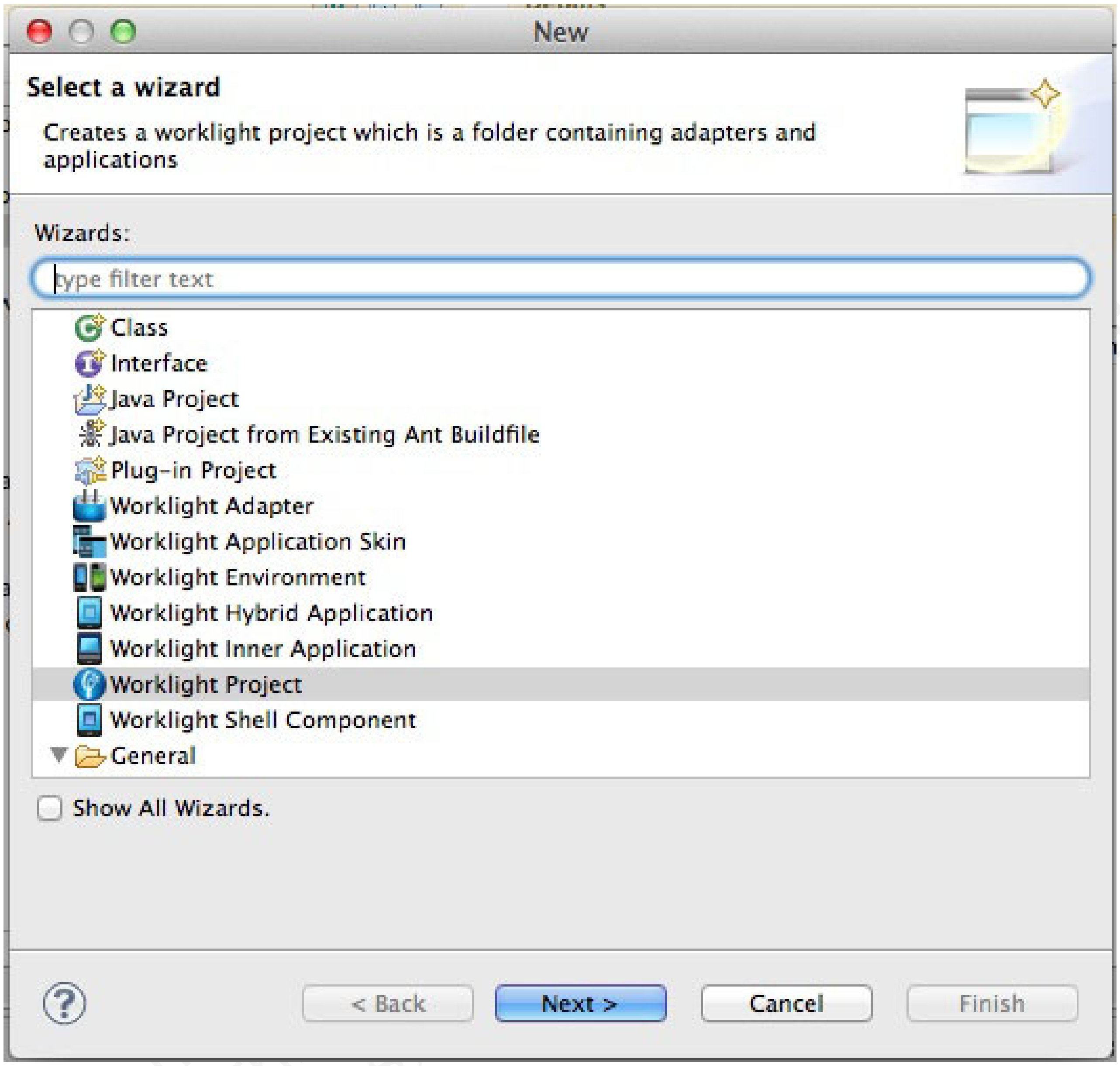
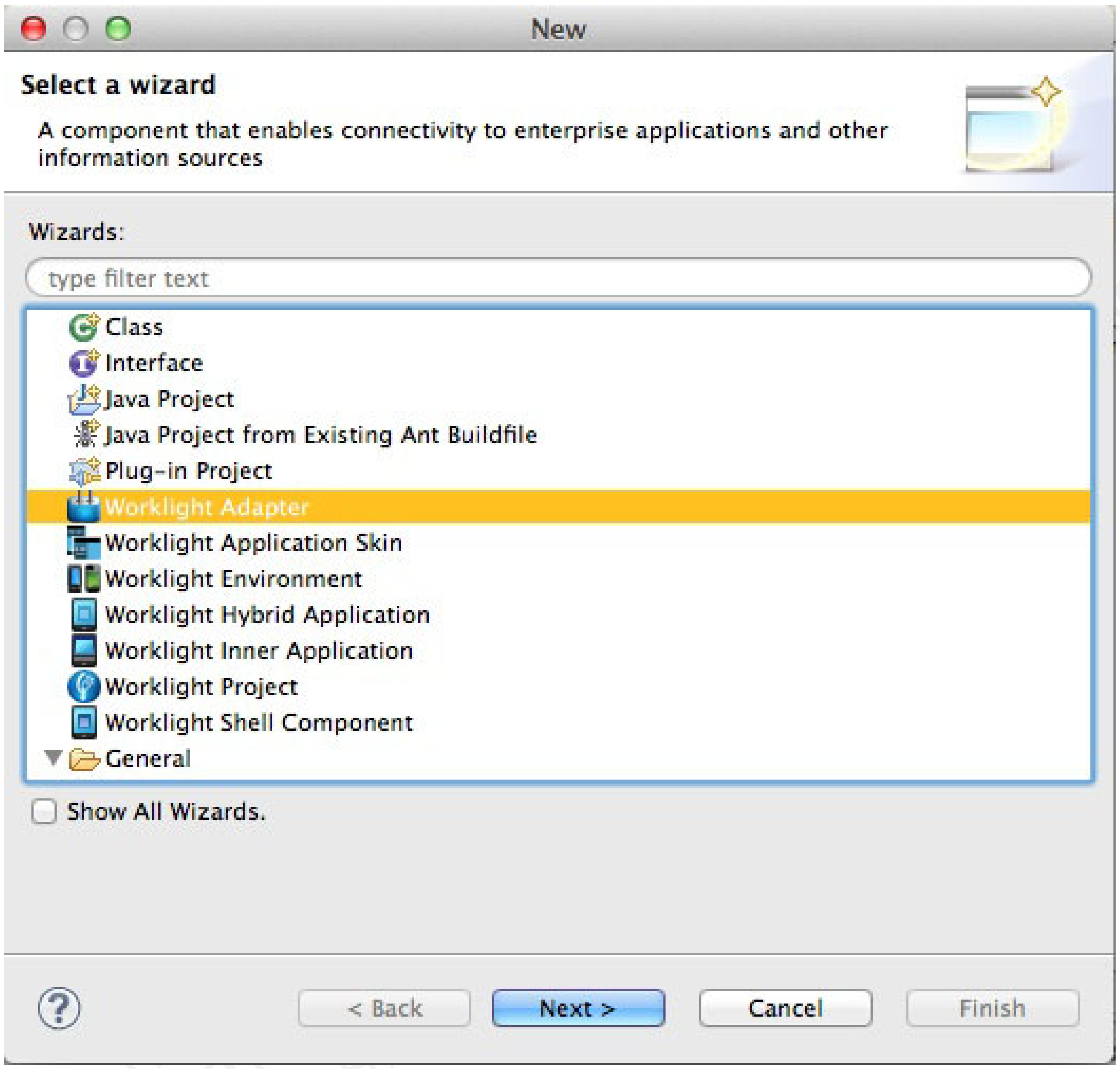

New
Select a wizard
Creates a worklight project which is a folder containing adapters and applications
Wizards:
type filter text
Class
Interface
Java Project
Java Project from Existing Ant Buildfile
Plug-in Project
Worklight Adapter
Worklight Application Skin
Worklight Environment
Worklight Hybrid Application
Worklight Inner Application
Worklight Project
Worklight Shell Component
General
Show All Wizards.
< Back
Next >
Cancel
Finish

Figure 13. Creating an IBM Worklight project from the wizard.

2. In the New IBM Worklight Project window, specify a name for the project and click **Finish**.
3. If you set up IBM Worklight shortcuts, right-click the IBM Worklight Project to which you want to add the adapter, and select **New** > **Adapter**. Otherwise, select **New** > **Other**, then select **Worklight** > **Adapter** from the list of wizards and click **Next**.


New

Select a wizard

A component that enables connectivity to enterprise applications and other information sources

Wizards:

type filter text

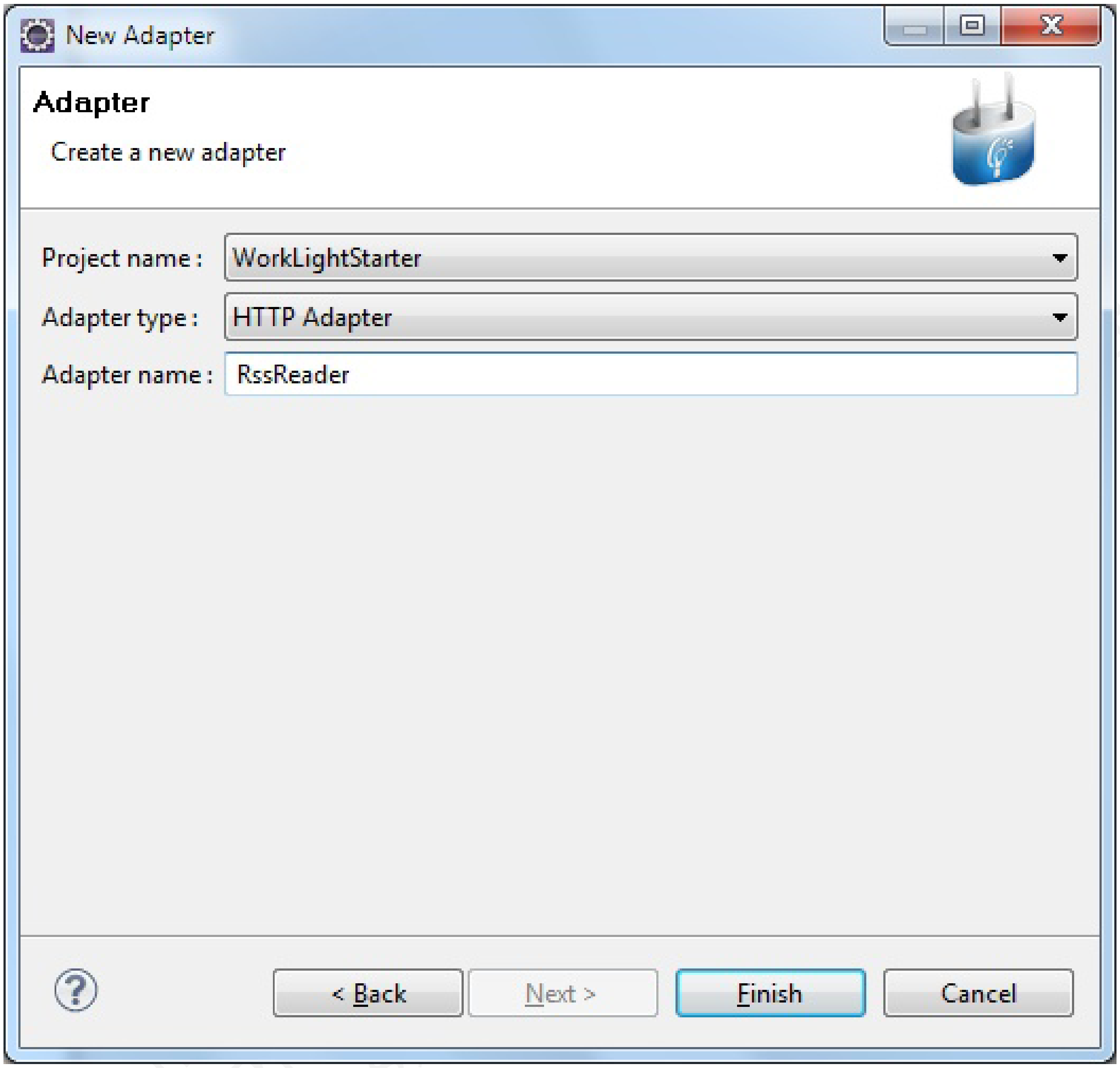
- Class
- Interface
- Java Project
- Java Project from Existing Ant Buildfile
- Plug-in Project
- Worklight Adapter
- Worklight Application Skin
- Worklight Environment
- Worklight Hybrid Application
- Worklight Inner Application
- Worklight Project
- Worklight Shell Component
- General

☐ Show All Wizards.

Figure 14. Configuring a new IBM Worklight adapter.

The New Adapter window is displayed.

4. Select the required adapter type from the **Adapter type** list and enter a name for the adapter in the **Adapter name** field. Click **Finish**.



Project name : WorkLightStarter

Adapter type : HTTP Adapter

Adapter name : RssReader

Figure 15. Selecting an adapter type.

Adapter invocation service

Adapter procedures can be invoked by issuing an HTTP request to the IBM Worklight invocation service: `http(s)://<server>:<port>/<Context>/invoke`.

The following parameters are required:

Table 21. Parameters for adapter invocation

Property	Description
`adapter`	The name of the adapter
`procedure`	The name of the procedure
`parameters`	An array of parameter values

The request can be either `GET` or `POST`.

Note: The invocation service uses the same authentication framework as described in the "Authentication configuration" on page 137 section.

The default security test for adapter procedures contains Anti-XSRF protection, but this configuration can be overridden by either:

- Implementing your own authentication realm (see "Authenticators and Login Modules" on page 140 for more details).
- Disabling the authentication requirement for a specific procedure. You can do so by adding the **`securityTest`**`="`*`wl_unprotected`*`"` property to the `<procedure>` element in the adapter XML file.

  Note: Disabling authentication requirement on a procedure means that this procedure becomes completely unprotected and anyone who knows the adapter and the procedure name can access it. Therefore, consider protecting sensitive adapter procedures.

Implementing adapter procedures

Implement a procedure in the adapter XML file, using an appropriate signature and any return value.

Before you begin

You have declared a procedure in the adapter XML file, using a `<procedure>` tag.

Procedure

Implement the procedure in the adapter JavaScript file. The signature of the JavaScript function that implements the procedure has the following format:

```
function funcName (param1, param2, ...),
```

Where:

- *funcName* is the name of function which the procedure implements. This name must be the same as the value specified in the `name` attribute of the`<procedure>` element in the adapter XML file.
- *param1* and *param2* are the function parameters. The parameters can be scalars (strings, integers, and so on) or objects.

In your JavaScript code, you can use the Worklight server-side JavaScript API to access back-end applications, invoke other procedures, access user properties, and write log and debug lines.
You can return any value from your function, scalar or object.

The Rhino container

IBM Worklight uses Rhino as the engine for running the JavaScript script used to implement adapter procedures.

Rhino is an open source JavaScript container developed by Mozilla. In addition to being part of Java 6, Rhino has two other advantages:

- It compiles the JavaScript code into byte code, which runs faster than interpreted code.
- It provides access to Java code directly from JavaScript. For example:

Experimental IBM Worklight offline user documentation.
This document is provided "as is".
In case of issues refer to the online user documentation.

```
var date = new java.util.Date();
var millisec = date.getTime()
```

Encoding a SOAP XML envelope

Follow these instructions to encode a SOAP XML envelope within a request body

About this task

When you need to invoke a SOAP-based service in an HTTP adapter, encode the SOAP XML envelope within the request body.

Procedure

Encode XML within JavaScript by using E4X E4X is officially part of JavaScript 1.6. This technology can be used to encode any XML document, not necessarily SOAP envelopes. For more information about E4X, see the related link.

Example

```
var request =
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Body>
<requestMessageObject xmlns="http://acme.com/ws/">
<messageHeader>
<version>1.0</version>
<originatingDevice>{originatingDevice}</originatingDevice>
<originatingIP>
{WL.Server.configuration["local.IPAddress"]}
</originatingIP>
<requestTimestamp>
{new Date().toLocaleString()}
</requestTimestamp>
</messageHeader>
<messageData>
<context>
<userkey>{userKey}</userkey>
<sessionid>{sessionid}</sessionid>
</context>
</messageData>
</requestMessageObject>
</S:Body>
</S:Envelope>;
```

You can use the WL.Server.signSoapMessage() method only inside a procedure declared within an HTTP adapter. It signs a fragment of the specified envelope with ID `wsId`, using the key in the specified **`keystoreAlias`**, inserting the digital signature into the input document.

To use WL.Server.signSoapMessage() API when running IBM Worklight on IBM WebSphere Application Server you might need to add a JVM argument that instructs WebSphere to use a specific **`SOAPMessageFactory`** implementation instead of a default one. To do this, you must go to **Application servers *{server_name}*** > **Process definition** > **Java Virtual Machine** and provide the following argument under `Generic JVM arguments`, typing in the code phrase exactly as it is presented here:

```
Djavax.xml.soap.MessageFactory=com.sun.xml.internal.messaging.saaj.soap.ver1_1.SOAPMessage
```

You must then restart the JVM.

Experimental IBM Worklight offline user documentation
In case of issues, refer to the online user documentation.
This document is provided as is.

Important: This workaround is only for IBM WebSphere.

Related information:

http://www.w3schools.com/xml/xml_e4x.asp

Calling Java code from a JavaScript adapter

Follow these instructions to instantiate Java objects and call their methods from JavaScript code in your adapter.

Before you begin

Attention: The name of any Java package to which you refer from within an adapter must start with the domains `com`, `org`, or `net`.

Procedure

1. Instantiate a Java object by using the `new` keyword and apply the method on the newly instantiated object.
2. Optional: Assign a JavaScript variable to be used as a reference to the newly instantiated object.
3. Include the Java classes that are called from the JavaScript adapter in your IBM Worklight project under *`Worklight Project Folder`*`/server/java`. The IBM Worklight Studio automatically builds them and deploys them to the IBM Worklight Server, also placing the result of the build under *`Worklight Project Folder`*`/bin`

Example

```
var x = new MyJavaClass();
var y = x.myMethod(1, "a");
```

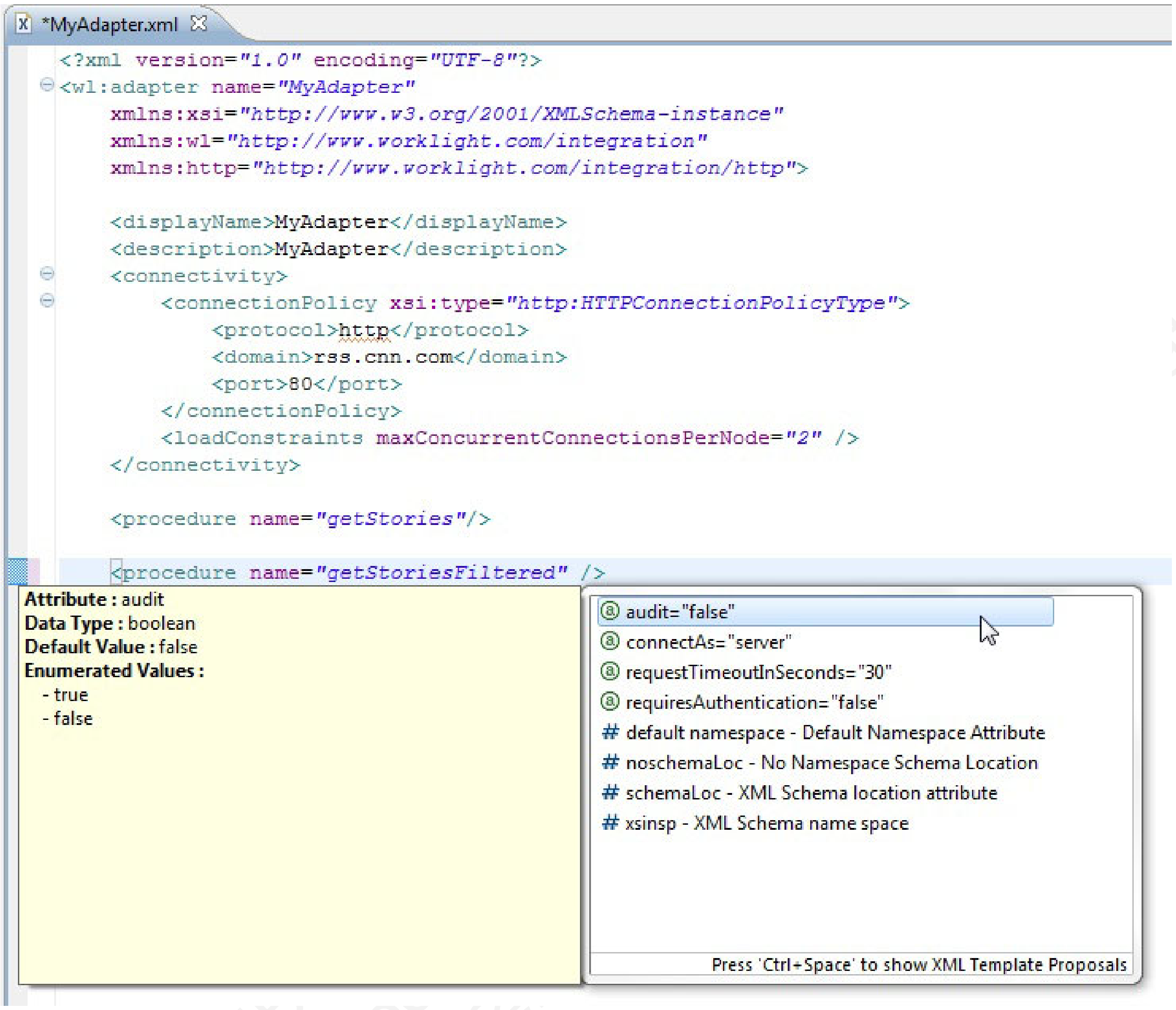
Features of the IBM Worklight Studio

The Worklight Studio provides the facilities to automatically complete attribute values, validate adapters on three levels, and to fix errors in adapter configuration.

Auto-complete

The auto-complete feature offers a list of possible values for attribute values. In the example below, the JavaScript Editor displays a list of values for the possible field types of a record field. In this way, the auto-complete feature helps correct configuration of an adapter.

Experimental IBM Worklight online user documentation
This document is provided "as is".
In case of issues refer to the online documentation.



*MyAdapter.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<wl:adapter name="MyAdapter"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:wl="http://www.worklight.com/integration"
    xmlns:http="http://www.worklight.com/integration/http">

    <displayName>MyAdapter</displayName>
    <description>MyAdapter</description>
    <connectivity>
        <connectionPolicy xsi:type="http:HTTPConnectionPolicyType">
            <protocol>http</protocol>
            <domain>rss.cnn.com</domain>
            <port>80</port>
        </connectionPolicy>
        <loadConstraints maxConcurrentConnectionsPerNode="2" />
    </connectivity>

    <procedure name="getStories"/>

    <procedure name="getStoriesFiltered" />
```

Attribute : audit
Data Type : boolean
Default Value : false
Enumerated Values :
- true
- false

```
audit="false"
connectAs="server"
requestTimeoutInSeconds="30"
requiresAuthentication="false"
# default namespace - Default Namespace Attribute
# noschemaLoc - No Namespace Schema Location
# schemaLoc - XML Schema location attribute
# xsinsp - XML Schema name space
```

Press 'Ctrl+Space' to show XML Template Proposals

Figure 16. Adapter configuration through the auto-complete feature.

Adapter validation

IBM Worklight Studio provides adapter validation on three levels:

Schema validation
The XML Editor validates the XML file as well-formed and conforming to the rules defined in the validating schema.

Logical validation of the XML
IBM Worklight Studio provides logical validation of the XML, based on IBM Worklight adapter constraints. For example, if a procedure is a JavaScript procedure, then field mapping is not permitted.

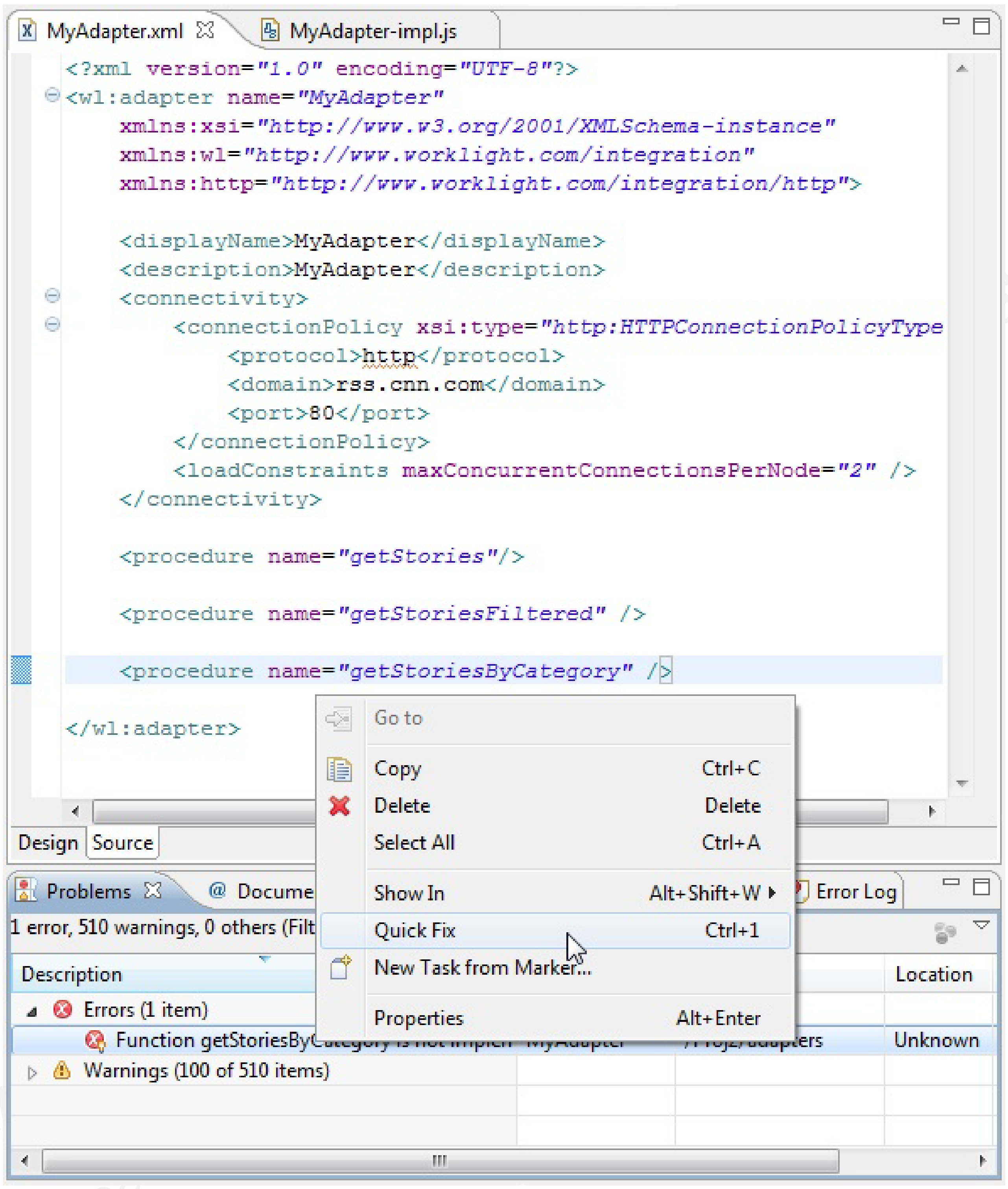
XML/JavaScript validation
IBM Worklight Studio provides validation between XML and JavaScript. It verifies that each declared JavaScript procedure has a corresponding procedure in the adapter JavaScript file with the appropriate signature (that is, input parameters and return values).

Quick fix

The IBM Worklight Studio provides Quick Fix options for adapter configuration errors.

Whenever an error is detected, the error console displays the offending code. To activate the Quick Fix window, right-click the error in the console and select **Quick Fix**. Alternatively, press Ctrl+1.

Experimental IBM Worklight offline user documentation.
This document is provided "as is".
In case of issues, refer to the online user documentation.



```
<?xml version="1.0" encoding="UTF-8"?>
<wl:adapter name="MyAdapter"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:wl="http://www.worklight.com/integration"
    xmlns:http="http://www.worklight.com/integration/http">

    <displayName>MyAdapter</displayName>
    <description>MyAdapter</description>
    <connectivity>
        <connectionPolicy xsi:type="http:HTTPConnectionPolicyType
            <protocol>http</protocol>
            <domain>rss.cnn.com</domain>
            <port>80</port>
        </connectionPolicy>
        <loadConstraints maxConcurrentConnectionsPerNode="2" />
    </connectivity>

    <procedure name="getStories"/>

    <procedure name="getStoriesFiltered" />

    <procedure name="getStoriesByCategory" />

</wl:adapter>
```

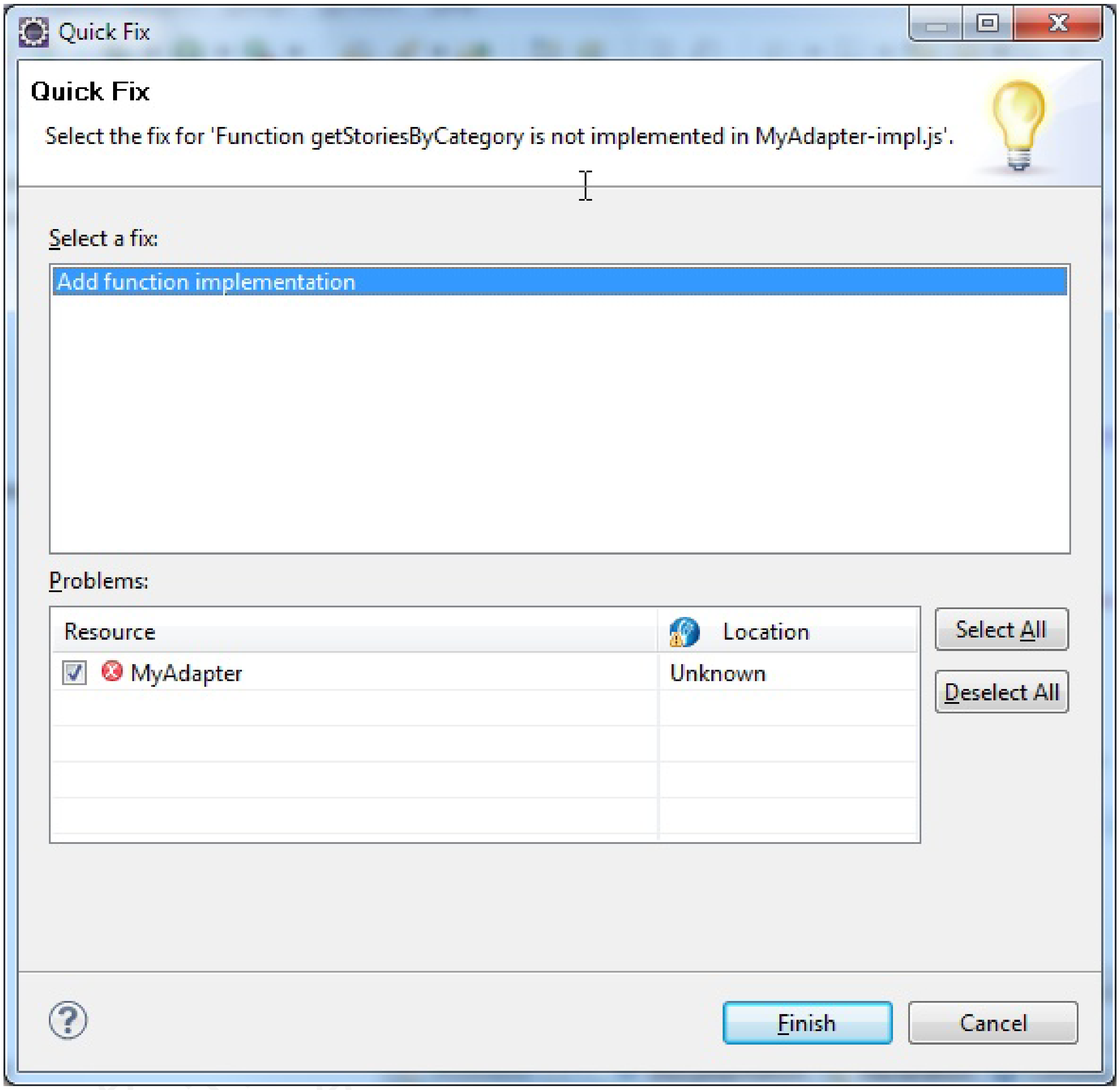
1 error, 510 warnings, 0 others (Filt

Errors (1 item)

Function getStoriesByC... MyAdapter /Proj2/adapters Unknown

Warnings (100 of 510 items)

Figure 17. Quick Fix options for adapter configuration problems.



Quick Fix

Select the fix for 'Function getStoriesByCategory is not implemented in MyAdapter-impl.js'.

Select a fix:

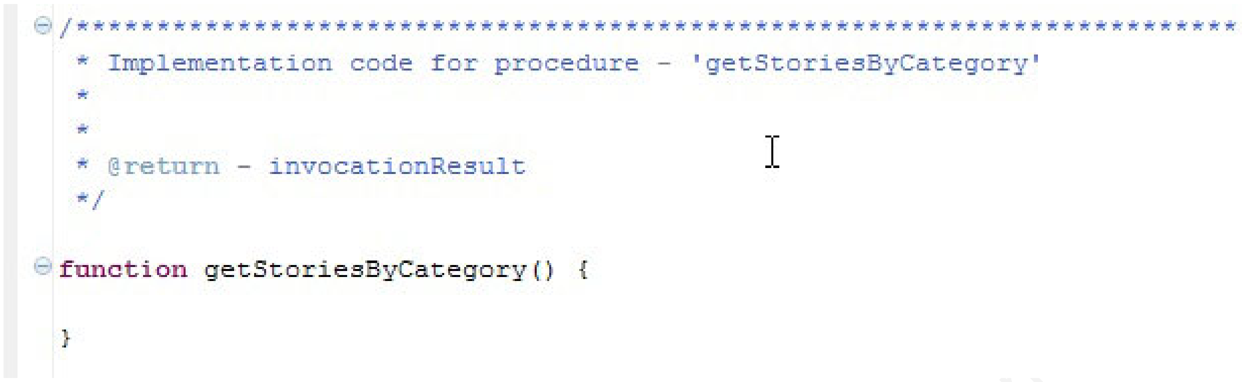
Add function implementation

Problems:

Resource	Location
✓ MyAdapter	Unknown

Figure 18. Quick Fix option for missing JavaScript functions.

Specifically, the IBM Worklight Studio provides a Quick Fix option for missing JavaScript functions. The Quick Fix creates the missing function in the corresponding JavaScript file (also creating the file if one does not exist).



```
/*********************************************************************
 * Implementation code for procedure - 'getStoriesByCategory'
 *
 *
 * @return - invocationResult
 */

function getStoriesByCategory() {

}
```

Procedure invocation

You can test a procedure by running it within the Worklight Studio.

Note: This feature is available only when you are running IBM Worklight Studio. It is not available when you run an adapter on a stand-alone server based on WebSphere Application Server or Tomcat.

In IBM Worklight Studio, you can select a procedure, enter a set of parameters, and invoke the procedure on the IBM Worklight Server. Only procedure invocations are supported, with results displayed in a browser window. For each invoked procedure, the IBM Worklight Studio remembers the most recent parameter values, so you can reinvoke the procedure without re-entering parameter values.

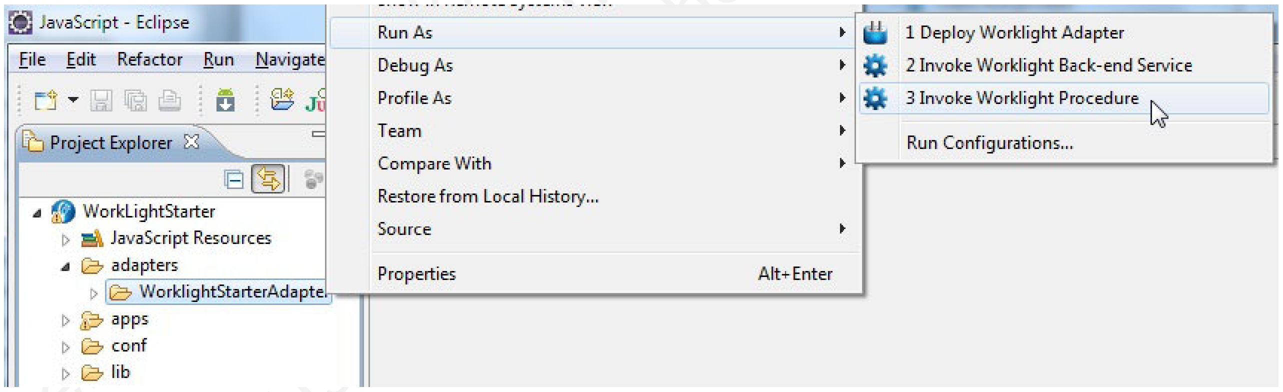
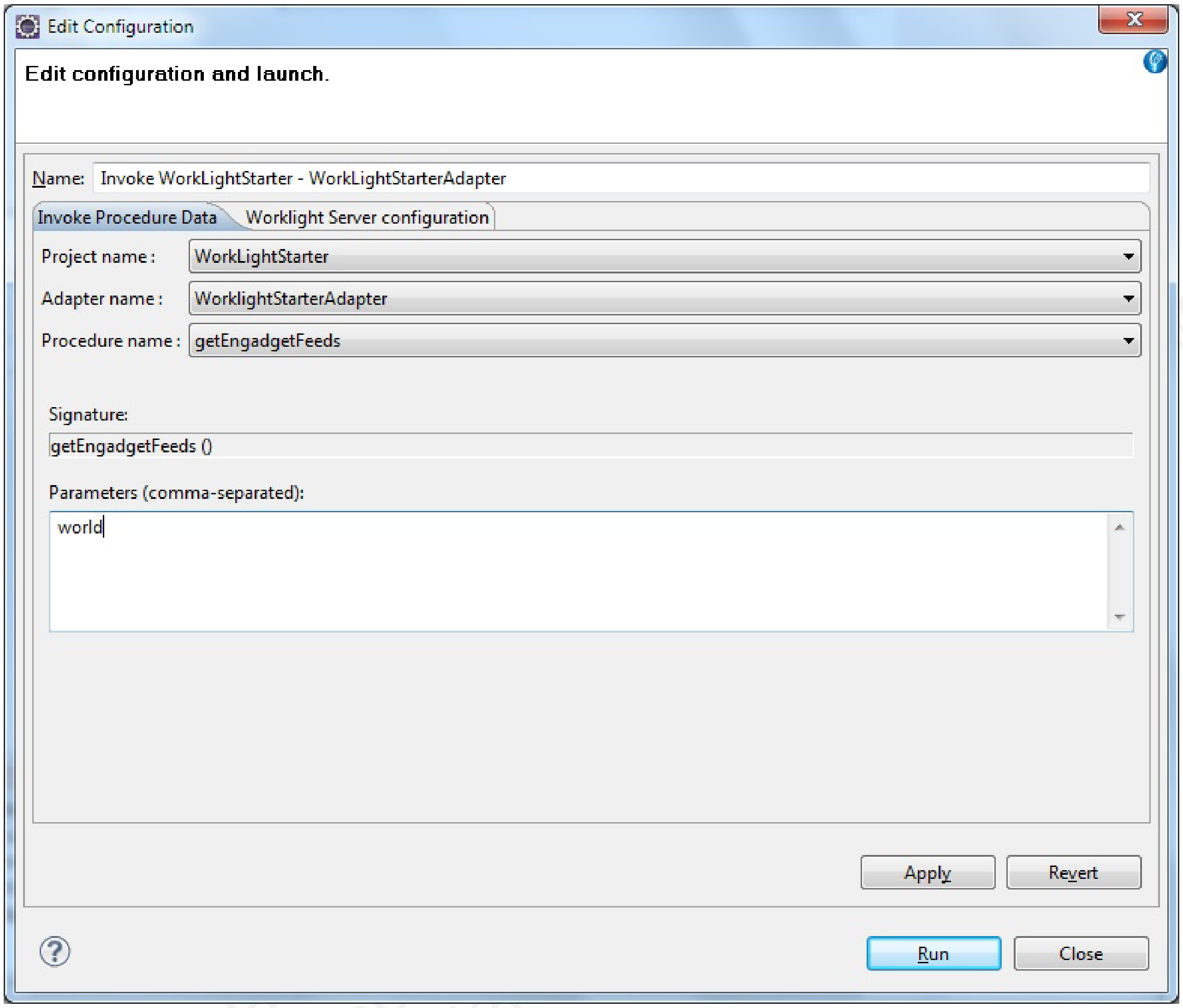


Figure 19. Invoking IBM Worklight procedures.

In the dialog box, provide a comma-separated list of procedure parameters.



Edit configuration and launch.

Name: Invoke WorkLightStarter - WorkLightStarterAdapter

Project name : WorkLightStarter

Adapter name : WorklightStarterAdapter

Procedure name : getEngadgetFeeds

Signature:

getEngadgetFeeds ()

Parameters (comma-separated):

world

Figure 20. Launching the procedure.

Invoking a back-end service

You can invoke a back-end service and receive the data retrieved by the service, in the Worklight studio.

About this task

Note: This feature is only available when running within the IBM Worklight Studio. It is not available when running an adapter on a stand-alone server based on WebSphere Application Server or Tomcat.

In IBM Worklight Studio, you can invoke a back-end service and immediately receive the data retrieved by the service in XML and JSON formats. You can also define and test a custom XSL transformation that converts the resulting XML into JSON.

Experimental IB
This do
In case of issues, refe
on.

Procedure

To run a back-end service:

1. Right-click an adapter file, and select **Run As** > **Invoke Worklight Back-end Service**.

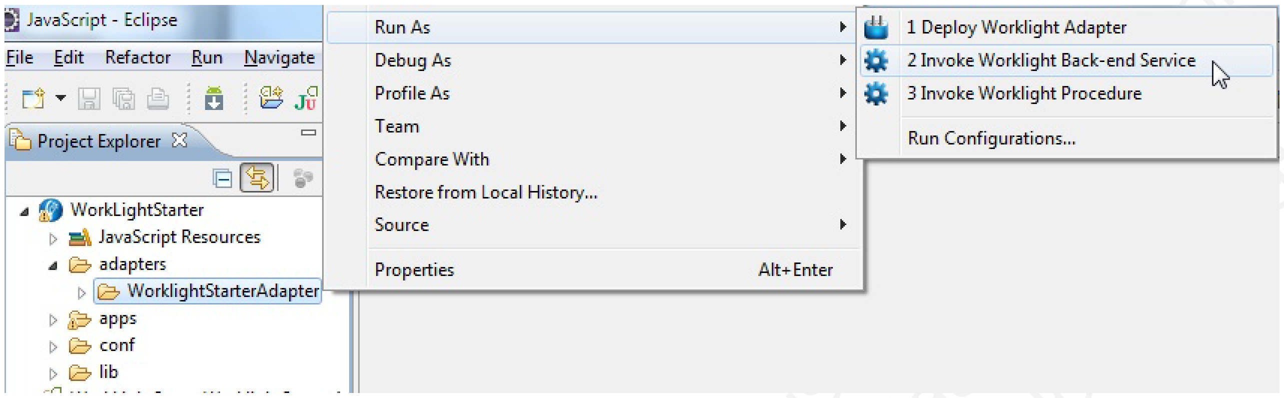
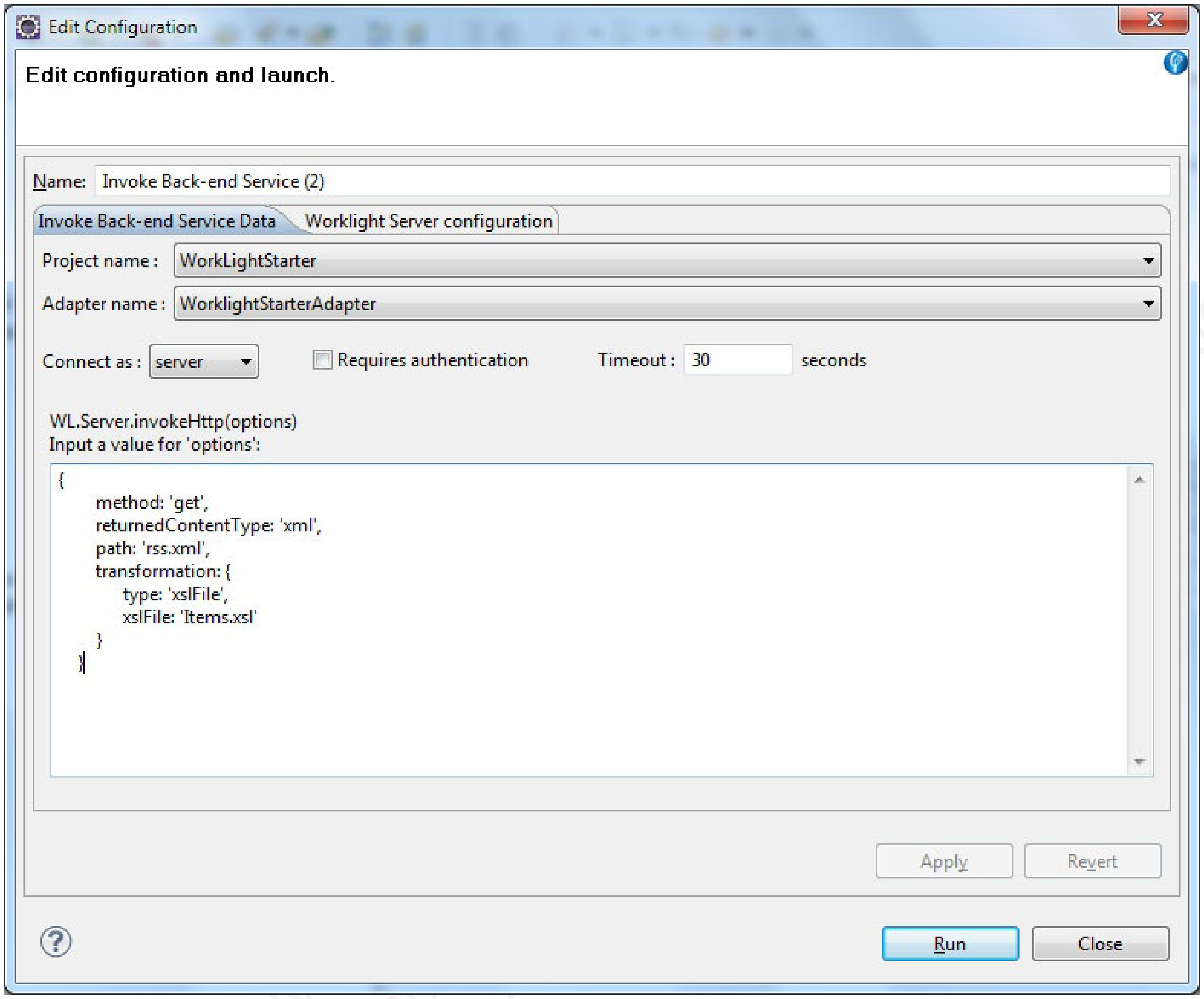


Figure 21. Invoking an IBM Worklight back-end service

2. In the dialog box, provide the invocation service parameters. You can copy them from your code and paste them directly into the dialog box.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.



Edit Configuration

Edit configuration and launch.

Name: Invoke Back-end Service (2)

Project name : WorkLightStarter

Adapter name : WorklightStarterAdapter

Connect as : server

Requires authentication

Timeout : 30 seconds

WL.Server.invokeHttp(options)
Input a value for 'options':

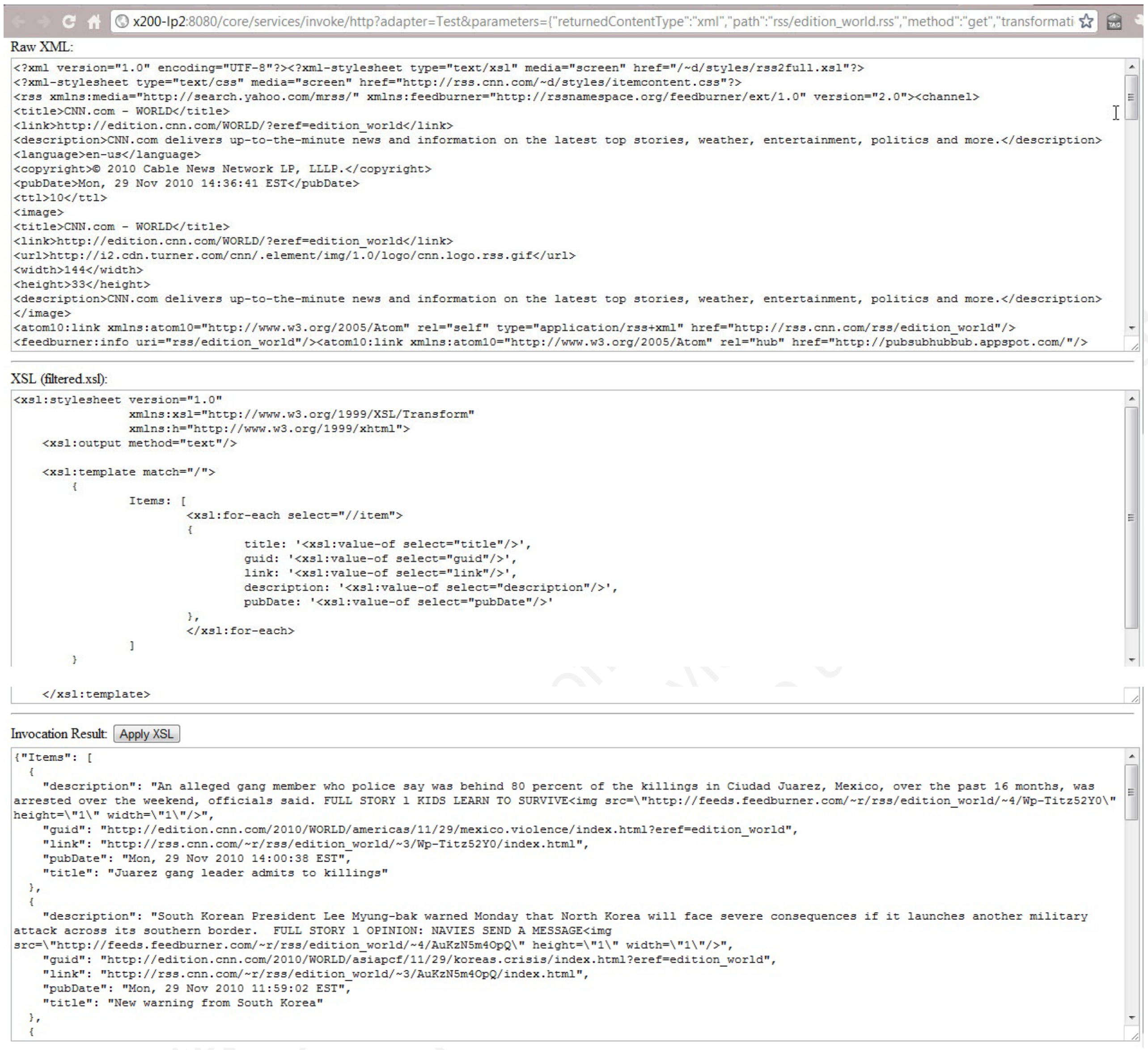
```
{
    method: 'get',
    returnedContentType: 'xml',
    path: 'rss.xml',
    transformation: {
        type: 'xslFile',
        xslFile: 'Items.xsl'
    }
}
```

Figure 22. Invocation parameters.

A browser window opens, displaying the retrieved data in XML and JSON format, and the XSL transformation (if defined) that was used to convert the XML to JSON.

3. Optional: Change the XSL transformation by editing it in the edit box, then click **Apply XSL** to regenerate the JSON format.

Experimental IBM
This document
In case of issues refer
ion.



x200-lp2:8080/core/services/invoke/http?adapter=Test&parameters={"returnedContentType":"xml","path":"rss/edition_world.rss","method":"get","transformati

Raw XML:

```
<?xml version="1.0" encoding="UTF-8"?><?xml-stylesheet type="text/xsl" media="screen" href="/~d/styles/rss2full.xsl"?>
<?xml-stylesheet type="text/css" media="screen" href="http://rss.cnn.com/~d/styles/itemcontent.css"?>
<rss xmlns:media="http://search.yahoo.com/mrss/" xmlns:feedburner="http://rssnamespace.org/feedburner/ext/1.0" version="2.0"><channel>
<title>CNN.com - WORLD</title>
<link>http://edition.cnn.com/WORLD/?eref=edition_world</link>
<description>CNN.com delivers up-to-the-minute news and information on the latest top stories, weather, entertainment, politics and more.</description>
<language>en-us</language>
<copyright>© 2010 Cable News Network LP, LLLP.</copyright>
<pubDate>Mon, 29 Nov 2010 14:36:41 EST</pubDate>
<ttl>10</ttl>
<image>
<title>CNN.com - WORLD</title>
<link>http://edition.cnn.com/WORLD/?eref=edition_world</link>
<url>http://i2.cdn.turner.com/cnn/.element/img/1.0/logo/cnn.logo.rss.gif</url>
<width>144</width>
<height>33</height>
<description>CNN.com delivers up-to-the-minute news and information on the latest top stories, weather, entertainment, politics and more.</description>
</image>
<atom10:link xmlns:atom10="http://www.w3.org/2005/Atom" rel="self" type="application/rss+xml" href="http://rss.cnn.com/rss/edition_world"/>
<feedburner:info uri="rss/edition_world"/><atom10:link xmlns:atom10="http://www.w3.org/2005/Atom" rel="hub" href="http://pubsubhubbub.appspot.com/"/>
```

XSL (filtered.xsl):

```
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:h="http://www.w3.org/1999/xhtml">
    <xsl:output method="text"/>

    <xsl:template match="/">
        {
                Items: [
                        <xsl:for-each select="//item">
                        {
                                title: '<xsl:value-of select="title"/>',
                                guid: '<xsl:value-of select="guid"/>',
                                link: '<xsl:value-of select="link"/>',
                                description: '<xsl:value-of select="description"/>',
                                pubDate: '<xsl:value-of select="pubDate"/>'
                        },
                        </xsl:for-each>
                ]
        }

    </xsl:template>
```

Invocation Result: Apply XSL

```
{"Items": [
  {
    "description": "An alleged gang member who police say was behind 80 percent of the killings in Ciudad Juarez, Mexico, over the past 16 months, was
arrested over the weekend, officials said. FULL STORY l KIDS LEARN TO SURVIVE<img src=\"http://feeds.feedburner.com/~r/rss/edition_world/~4/Wp-Titz52Y0\"
height=\"1\" width=\"1\"/>",
    "guid": "http://edition.cnn.com/2010/WORLD/americas/11/29/mexico.violence/index.html?eref=edition_world",
    "link": "http://rss.cnn.com/~r/rss/edition_world/~3/Wp-Titz52Y0/index.html",
    "pubDate": "Mon, 29 Nov 2010 14:00:38 EST",
    "title": "Juarez gang leader admits to killings"
  },
  {
    "description": "South Korean President Lee Myung-bak warned Monday that North Korea will face severe consequences if it launches another military
attack across its southern border.  FULL STORY l OPINION: NAVIES SEND A MESSAGE<img
src=\"http://feeds.feedburner.com/~r/rss/edition_world/~4/AuKzN5m4OpQ\" height=\"1\" width=\"1\"/>",
    "guid": "http://edition.cnn.com/2010/WORLD/asiapcf/11/29/koreas.crisis/index.html?eref=edition_world",
    "link": "http://rss.cnn.com/~r/rss/edition_world/~3/AuKzN5m4OpQ/index.html",
    "pubDate": "Mon, 29 Nov 2010 11:59:02 EST",
    "title": "New warning from South Korea"
  },
  {
```

Figure 23. Browser window, showing retrieved data in XML and JSON format.

Deploying an adapter

In IBM Worklight Studio, you can automatically deploy a new or modified adapter to the IBM Worklight Server.

Procedure

Right-click the adapter folder and select **Run As** > **Deploy adapter on Worklight Server**.

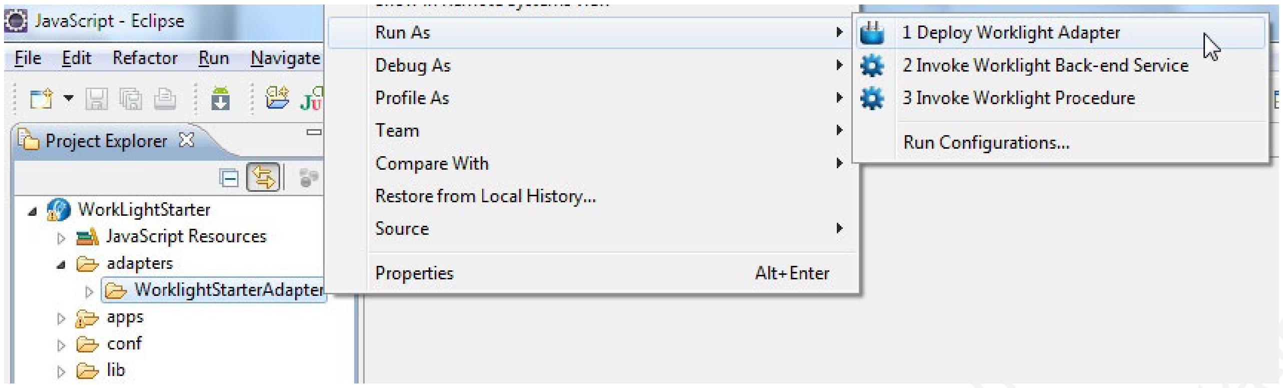


Figure 24. Deploying a Worklight adapter.

Transporting Worklight applications to test and production environments

When you have developed an application, transport it to a separate test and production project.

About this task

When you finish a development cycle of your application, you usually transport it to a testing environment, and then to a production environment.

Consider maintaining a separate IBM Worklight project for each such environment. Set up each project to contain only the configuration and custom server-side code related to that environment, and not the adapters and applications. In this way, you can maintain project customizations in a source control system. It also decouples the source code of adapters and applications from the environment they are running in. This decoupling makes their source code simpler and minimizes the need to change them when moving across environments.

The tools that you can use to transport apps and adapters across development, QA, and production environments are described in the following topics.

Transporting an application from development to another environment

To transport an application from the development environment to another environment, first update the application descriptor to match the new environment, then deploy one or more `.wlapp` and adapter files.

Before you begin

You have built one or more applications in the IBM Worklight Studio and a set of `.wlapp` files is created in the `bin` folder of your IBM Worklight Project. You now want to deploy the applications to a test or production environment.

- If you build an entire app, a file called *`app-name`*`.wlapp` is created, containing the code and resources of all environments that are supported by your app. For example: `myApp.wlapp`.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

- If you build an app only for specific environments, a file called *app-name-env-version*.wlapp is created per environment. For example: myApp-android-1.0.wlapp.

About this task

First, you prepare the application or applications for deployment and then you deploy them. You can deploy many apps within the same project. The following instructions lead you through this process.

Procedure

1. For each application in the project, change the settings in the application-descriptor.xml file to match your production environment.

   The following settings might need changing, depending on the functions of the app.
   - Context root
   - Settings screen
   - IBM Worklight Server root URL. Set worklightServerRootURL to the URL of the remote server that the application will be using.
   - Push notification certificates
   - Device provisioning
   - Application authenticity
   - User authentication
   - The Android shared user ID

   For more information, see "The application descriptor" on page 55
2. Change the settings in the worklight.properties file to match your production environment.

   Specify the database by setting wl.db.type to MYSQL, DB2, DERBY or ORACLE depending on your database. Set wl.db.jndi.name to the JNDI name for the database as follows:
   - On WebSphere Application Server, Full Profile or Liberty Profile:

     ```
     wl.db.jndi.name=jdbc/WorklightDS
     ```
   - On Apache Tomcat:

     ```
     wl.db.jndi.name=java:comp/env/jdbc/WorklightDS
     ```

   Specify the settings for the report database by setting wl.reports.db.type to MYSQL, DB2, DERBY or ORACLE depending on your database. Set wl.reports.db.jndi.name to the JNDI name for the database as follows:
   - On WebSphere Application Server, Full Profile or Liberty Profile:

     ```
     wl.reports.db.jndi.name=jdbc/WorklightReportsDS
     ```
   - On Apache Tomcat:

     ```
     wl.db.jndi.name=java:comp/env/jdbc/WorklightReportsDS
     ```

   Specify the properties that describe public Worklight Server access by setting publicWorkLightProtocol, publicWorkLightPort, and publicWorkLightContext. You might want to look at the worklight.properties file that was created during installation. You can locate the worklight.war file in the *installation_directory*/WorklightServer directory. Extract the .war file, and the worklight.properties file is located under the WEB-INF/classes/conf/ directory in the .war file.

   For more information, see "IBM Worklight properties" on page 406

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

3. Build each application in either of two ways:
   - Right-click the application and clink **Run As** > **Build All and Deploy**.
   - Use the Ant script tool that is described in "Building an application" on page 135

   This creates a *projectName*.war file in the \bin folder. This file contains the project configuration that was done in steps 1 and 2 and any classes built from Java code in the server/java folder.
4. Rename the .war file to *context_root*.war, where *context_root* is as set in the application-descriptor.xml file.
5. Deploy the .war file to the remote server, as described in "Deploying a customization .war file to an application server"
6. Open the IBM Worklight Console of the target environment. The address is of the format http://*your-remote-server*:*server-port*/*context_root*/console
7. From the IBM Worklight Console, deploy the relevant .wlapp files from the bin folder of your IBM Worklight project.
   - For more information about how to deploy an app by using IBM Worklight Console, see "Deploying apps" on page 133.
   - You can also deploy the app to the target environment by using an Ant task that is provided by IBM Worklight. For more information about how to deploy an app by using the provided Ant task, see "Deploying an application" on page 135.
8. Obtain the adapters from the development environment.
   a. Navigate to the bin folder in your project.
   b. Copy the .adapter file or files.
9. From the IBM Worklight Console, deploy the .adapter files from the bin folder of your project.
   - For more information about how to deploy an adapter by using IBM Worklight Console, see "Deploying adapters" on page 134.
   - You can also deploy the adapter to the target environment by using an Ant task that is provided with IBM Worklight. For more information about how to deploy an adapter by using the provided Ant task, see "Deploying an adapter" on page 136.

Deploying a customization .war file to an application server

After you build a project, you must deploy it to an application server. The method depends on the server platform.

About this task

When you build your Worklight project, the customization .war file is created in the bin directory of the project. You must deploy it to the application server that you chose when you installed Worklight Server. The following topics contain instructions to deploy the .war file on different application server platforms. You can deploy only one .war file to a single application server instance. The .war file contains the console that you upload applications to. The .war file you choose to deploy is not specific to the application from which the .war file was generated. You can upload all the IBM Worklight apps that are generated from the same Worklight Studio version to the same .war file. The instructions assume that the .war file is named projectName.war. This is the default name of the .war file, but you might have renamed it to a different name as described in previous topics.

Experimental IBM Worklight user documentation. This document is provided 'as is'. In case of issues, refer to the online user documentation.

Procedure

- To deploy the customization `.war` file to Apache Tomcat, see "Deploying a customization .war file to Apache Tomcat."
- To deploy the customization `.war` file to WebSphere Application Server Liberty Profile, see "Deploying a customization .war file to WebSphere Application Server Liberty Profile."
- To deploy the customization `.war` file to WebSphere Application Server Full Profile, see "Deploying a customization .war file to WebSphere Application Server Full Profile" on page 131.

Deploying a customization .war file to Apache Tomcat

Copy the `.war` file to the `webapps` directory and start Tomcat.

Before you begin

You have built a project, and the customization `.war` file is in the `bin` directory of the project.

Procedure

1. Copy the Worklight Customization `.war` file to Tomcat by issuing one of the following commands in the `bin` directory:
   - On UNIX and Linux systems: `cp projectName.war` *`TOMCAT_HOME`*`/webapps`
   - On Windows systems: `copy projectName.war` *`TOMCAT_HOME`*`/webapps`

   where *TOMCAT_HOME* is the directory where Tomcat is installed.
2. Start Tomcat.

Results

You can now access IBM Worklight Console at `http://`*`server`*`:`*`port`*`/projectName/console`, where *server* is the host name of your server and *port* is the port number (default 9080).

Deploying a customization .war file to WebSphere Application Server Liberty Profile

Copy the `.war` file to the `apps` directory, declare the app, and restart the server.

Before you begin

You have built a project, and the customization `.war` file is in the `bin` directory of the project.

Procedure

1. Copy the Worklight Customization `.war` file to the server by issuing one of the following commands in the `bin` directory:
   - On UNIX and Linux systems: `cp projectName.war` *`install_dir`*`/server/wlp/usr/servers/worklightServer/apps`, where *install_dir* is the directory in which WebSphere Application Server Liberty Profile is installed.
   - On Windows 7 systems: `copy projectName.war C:\ProgramData\IBM\Worklight\WAS85liberty-server\wlp\usr\servers\worklightServer\apps`
   - On Windows XP systems: `copy projectName.war C:\Documents and Settings\All Users\Application Data\IBM\Worklight\WAS85liberty-server\wlp\usr\servers\worklightServer\apps`
2. Declare the application in the `server.xml` file:

Experimental IBM Worklight online user documentation
This document is provided "as is"
In case of issues, refer to the online user documentation.

a. Open the `server.xml` file.

   On UNIX and Linux systems, the file is in the `*install_dir*/server/wlp/usr/servers/worklightServer` directory.

   On Windows 7 systems, the file is in the `C:\ProgramData\IBM\Worklight\WAS85liberty-server\wlp\usr\servers\worklightServer` directory.

   On Windows XP systems, the file is in the `C:\Documents and Settings\All Users\Application Data\IBM\Worklight\WAS85liberty-server\wlp\usr\servers\worklightServer` directory.

b. Add the following information to the file:

```
<application id="projectName" name="projectName" location="projectName.war" type="war">
       <classloader delegation="parentLast">
           <commonLibrary>
               <fileset dir="${shared.resource.dir}/lib" includes="worklight-jee-library.j
           </commonLibrary>
       </classloader>
</application>
```

3. Restart the server

Results

You can now access IBM Worklight Console at `http://*server*:*port*/projectName/console`, where *server* is the host name of your server and *port* is the port number (default 9080).

Deploying a customization .war file to WebSphere Application Server Full Profile

Install the `.war` file into WebSphere Application Server, configure the class loader policies, and start the application.

Before you begin

You have built a project, and the customization `.war` file is in the `bin` directory of the project. You have created a stand-alone profile with an application server named Worklight and the server is using the default ports.

Procedure

1. Log on to the WebSphere Application Server administration console for your IBM Worklight server. The address is of the form `http://*server*.com:9060/ibm/console`, where *server* is the name of the server.
2. Install the IBM Worklight customization `.war` file:
   a. Click **Applications** > **New** > **New Enterprise Application** or **Applications** > **New Application** > **New Enterprise Application**, depending on your version of WebSphere Application Server.
   b. Navigate to your Worklight project `bin` directory to locate the `projectName.war` file
   c. Select `projectName.war` and click **Next**.
   d. On the "How do you want to install the application?" page, select **Detailed** and click **Next**.
   e. On the "Application Security Warnings" page, click **Continue**.
   f. Click **Continue** repeatedly until you reach Step 4 of the wizard, Map Shared Libraries.
   g. Select the **Select** check box for `projectName_war` and click **Reference shared libraries**.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

h. From the **Available** list, select **Worklight Platform Library** and click the **>** button.
i. Click **OK**.
j. Click **Next** until you reach the "Map context roots for web modules" page.
k. In the **Context Root field**, type `/projectName`.
l. Click **Next**.
m. Click **Finish**.

3. Optional: As an alternative to step 2, you can map the shared libraries as follows:
   a. Click **Applications** > **Application Types** > **WebSphere enterprise applications** > **projectName_war**.
   b. In the References section, click **Shared library references**.
   c. Select the **Select** check box for `projectName_war` and click **Reference shared libraries**.
   d. From the Available list, select **Worklight Platform Library** and click the **>** button.
   e. Click **OK** twice to return to the `projectName_war` configuration page.
   f. Click the **Save** link
4. Configure the class loader policies and then start the application:
   a. Click the **Manage Applications** link, or click **Applications** > **WebSphere Enterprise Applications**.
   b. From the list of applications, click **projectName_war**.
   c. In the "Detail Properties" section, click the **Class loading and update detection** link.
   d. In the "Class loader order" pane, click **Classes loaded with local class loader first (parent last)**.
   e. Click **OK**.
   f. In the Modules section, click **Manage Modules**.
   g. From the list of modules, select the Worklight module.
   h. In the "Class loader order" pane, click **Classes loaded with local class loader first (parent last)**.
   i. Click **OK** twice.
   j. Click **Save**
   k. Select the **Select** check box for projectName_war and click **Start**.

Results

You can now access IBM Worklight Console at `http://`*`server`*`:`*`port`*`/projectName/console`, where *server* is the host name of your server and *port* is the port number (default 9080).

Administering adapters and apps in the IBM Worklight Console

Open the console before performing administrative tasks.

About this task

Before performing any of the other tasks in this collection of topics, open the IBM Worklight Console:

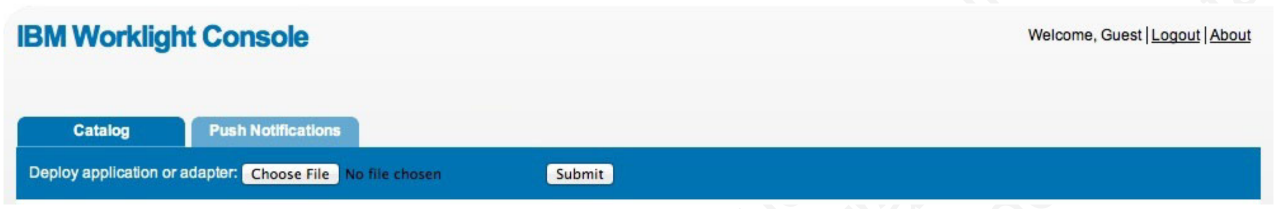
Experimental IBM Worklight online user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Procedure

1. Open a browser and enter the following URL: `http://`*`Worklight Server`*`:8080/console` 8080 is the default port. You might be using a different value.
2. If your IBM Worklight Server is configured to require login, and you are not currently logged in, log in when prompted to do so.

Results

The IBM Worklight Catalog page opens, and you can start performing adapter administration tasks.


IBM Worklight Console
Welcome, Guest | Logout | About
Catalog
Push Notifications
Deploy application or adapter: Choose File No file chosen
Submit

Deploying apps

Deploy an app by submitting it.

Procedure

To deploy an app:

1. Click **Browse**, then navigate to your `.wlapp` file and select it.
2. Click **Submit**.

Results

A message is displayed, indicating whether the deployment action succeeded or failed.

Deleting apps

Delete an app by clicking **Delete**.

Procedure

To delete an app:

Click **Delete** to the right of the app name.

Exporting adapter configuration files

Export the configuration files for the adapter by copying them from the source folder.

Procedure

To export a deployed adapter:

Obtain the adapter from the development environment.

1. Navigate to the `/bin` folder in your project
2. Copy the `.adapter` file or files.

Experimental IBM Worklight offline user documentation.
This document is provided "as is".
In case of issues, refer to the online user documentation.

Deploying adapters

Deploy an adapter from the console.

Procedure

To deploy an adapter:

1. Click **Browse**, then navigate to your `.adapter` file and select it.
2. Click **Submit**.

   A message is displayed indicating whether the deployment action succeeded or failed. If it succeeded, the details of the deployed adapter are added to the catalog.
3. Click **Show details** to view the connectivity details for the adapter and the list of procedures it exposes.

Modifying adapters

To modify an adapter, replace it with a new one.

Procedure

To modify an adapter:

Deploy the modified adapter file, as described in “Deploying adapters.”

Results

The new adapter replaces the original one.

Deleting adapters

Delete an adapter by clicking **Delete**.

Procedure

To delete an adapter:

Click **Delete** to the right of the adapter name.

Ant tasks for building and deploying

A set of Ant tasks is supplied with the Enterprise and Consumer editions.

IBM Worklight provides a set of Ant tasks that help you build and deploy adapters and apps to your IBM Worklight Server, and project customizations to your application server. A typical use of these Ant tasks is to integrate them with a central build service that is invoked manually or periodically on a central build server.

The Ant tasks are available with the IBM Worklight Enterprise Edition and the IBM Worklight Consumer Edition. They are not available with the IBM Worklight Developer Edition.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Building an application

The Ant task for building an application has the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<project base="." default="target-name">
<target name="target-name">
<taskdef resource="com/worklight/ant/defaults.properties">
<classpath>
<pathelement location="path_to_worklight-ant-platform.version.jar" />
</classpath>
</taskdef>
<app-builder
 applicationFolder="adapter-source-files-folder"
 environments="list-of-environments"
 nativeProjectPrefix="project-name"
 outputFolder="output-folder"/>
</target>
</project>
```

The `<app-builder>` element has the following attributes:

- The `applicationFolder` attribute specifies the root folder for the application, which contains the `application-descriptor.xml` file and other source files for the application.
- The `environments` attribute is a comma-separated list of environments to build. This attribute is optional. The default action is to build all environments.
- The `nativeProjectPrefix` attribute is mandatory when you build iOS applications
- The `ouptputFolder` attribute specifies the folder to which the resulting `.wlapp` file is written.

By default, running the Ant task to build an application does not handle the Dojo Toolkit, because Ant is not run with `build-dojo.xml`. You must explicitly configure the task to do so, by using the following app-builder setting in the Ant build file:

```
skinBuildExtensions=build-dojo.xml
```

If you use this setting, the Dojo Toolkit files are deployed with your application.

Deploying an application

The Ant task for deploying an application has the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<project base="." default="target-name">
<target name="target-name">
<taskdef resource="com/worklight/ant/defaults.properties">
<classpath>
<pathelement location="path_to_worklight-ant-platform.version.jar" />
</classpath>
</taskdef>
<app-deployer worklightServerHost="http://server-address:port" deployable="app.wlapp" />
</target>
</project>
```

The `<app-deployer>` element has the following attributes:

- The `worklightServerHost` attribute specifies the full URL of your Worklight server.
- The `deployable` attribute contains the `.wlapp` file to deploy.

Experimental IBM Worklight offline user documentation.
This document is provided "as is".
In case of issues, refer to the online user documentation.

If you must deploy more than one `.wlapp` file, add an <app-deployer> element for each file.

Building an adapter

The Ant task for building an adapter has the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<project base="." default="target-name">
<target name="target-name">
<taskdef resource="com/worklight/ant/defaults.properties">
<classpath>
<pathelement location="path_to_worklight-ant-platform.version.jar" />
</classpath>
</taskdef>
<adapter-builder folder="adapter-source-files-folder" destinationfolder="destination-folder" />
</target>
</project>
```

The `<adapter-builder>` element has the following attributes:

- The `folder` attribute specifies the folder that contains the source files of the adapter (its `.xml` and `.js` files).
- The `destinationfolder` attribute specifies the folder to which the resulting `.adapter` file is written.

If you must build more than one adapter file, add an <adapter-builder> element for each adapter.

Deploying an adapter

The Ant task for deploying an adapter has the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<project base="." default="target-name">
<target name="target-name">
<taskdef resource="com/worklight/ant/defaults.properties">
<classpath>
<pathelement location="path_to_worklight-ant-platform.version>.jar" />
</classpath>
</taskdef>
<adapter-deployer worklightserverhost="http://server-address:port" deployable="myAdapter.adapter" />
</target>
</project>
```

The `<adapter-deployer>` element has the following attributes:

- The `worklightserverhost` attribute specifies the full URL of your Worklight server.
- The `deployable` attribute specifies the `.adapter` file to deploy.

If you must deploy more than one `.adapter` file, add an <adapter-deployer> element for each file.

Deploying a project

The Ant task for deploying a project (building the server customization archive file) has the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="myProject" default="all">
<target name="taskdefs">
<taskdef resource="com/worklight/ant/defaults.properties">
```

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

```
<classpath>
<pathelement location="home_directory/git/worklight/worklight-build-tools/target/classes" />
<pathelement location="home_directory/git/worklightworklight-build-tools/target/worklight-build-to
</classpath>
</taskdef>
</target>
<target name="all" depends="taskdefs">
<war-builder
 projectfolder="."
 destinationfolder="bin/war"
 warfile="bin/cust.war"
 classesFolder="classes-folder"/>
</target>
</project>
```

The `<war-builder>` element has the following attributes:
- The `projectfolder` attribute specifies the path to your project.
- The `destinationfolder` attribute specifies a folder for holding temporary files.
- The `warfile` attribute specifies the destination and file name of the generated `.war` file
- The `classesFolder` attribute specifies a folder with compiled Java classes to add to the `.war` file. `.jar` files in the `projectfolder\server\lib` directory are added automatically

Authentication configuration

This collection of topics contains tasks and supporting information for configuring authentication in applications.

Protected resources

You can protect certain Worklight application resources.

Certain IBM Worklight application resources can be protected, and require client authentication before they can be accessed. The types of resources that can be protected are: adapter procedures, applications, and static IBM Worklight web applications, such as the IBM Worklight Console. Note, however, that the Application Center is not subject to the authentication model described here.

Security Tests

A security test defines a security configuration for a protected resource. Predefined tests are supplied for standard web and mobile security requirements. You can write your own custom security tests and define the sequence in which they are implemented.

A security test specifies one or more authentication realms and an authentication realm can be used by any number of security tests. A protectable resource can be protected by any number of realms.

A protected resource is protected by a security test. When a client attempts to access a protected resource, IBM Worklight checks whether the client is already authenticated according to all realms of the security test. If the client is not yet authenticated, IBM Worklight triggers the process of authentication for all unauthenticated realms.

Before you define security tests, define the authentication realms that the tests use.

Experimental IBM Worklight offline user documentation
This document is provided as is.
In case of issues, refer to the online user documentation.

Define a security test for each environment in the `application-descriptor.xml` file, by using the property **`securityTest`**`="`*`test_name`*`"`. If no security test is defined for a specific environment, only a minimal set of default platform tests is run.

You can define three types of security test:

webSecurityTest
: A test that is predefined to contain realms that are related to web security.

  Use a webSecurityTest to protect web applications.

  A webSecurityTest must contain one testUser element with a realm definition for user authentication. The identity that is obtained from this realm is considered to be a user identity.

  By default, a webSecurityTest includes protection against cross-site request forgery (XSRF) attacks.

mobileSecurityTest
: A test that is predefined to contain realms that are related to mobile security.

  Use a mobileSecurityTest to protect mobile applications.

  A mobileSecurityTest must contain one testUser element with a realm definition for user authentication. The identity that is obtained from this realm is considered to be a user identity.

  A mobileSecurityTest must contain one testDevice element with a realm definition for device authentication. The identity that is obtained from this realm is considered to be a device identity.

  By default, a mobileSecurityTest includes protection against XSRF attacks and the ability to remotely disable the mobile application from the Worklight Console.

customSecurityTest
: A custom security test. No predefined realms are added.

  Use a customSecurityTest to define your own security requirements and the sequence and grouping in which they occur.

  You can define any number of tests within a customSecurityTest. Each test specifies one realm. To define a realm as a user identity realm, add the property **`isInternalUserId`**`="true"` to the test. The **`isInternalUserID`** attribute means that this realm is used for user identification for reporting and push subscriptions. There must be exactly one such realm for every security configuration that is applied to a mobile or web resource.

  For a device auto provisioning realm, the **`isInternalDeviceID`** attribute means that this realm is used for device identification for reporting, push subscriptions, and device SSO features. There must be exactly one such realm for every security configuration that is applied to a mobile resource.

  Important: When you use device auto provisioning in customSecurityTests, an authenticity realm must also be present within the tests, otherwise provisioning cannot succeed.

  To specify the order in which a client must authenticate in the different realms, add the property **`step`**`="`*`n`*`"` to each test, where *n* indicates the sequence. If a sequence is not specified, then all tests are done in a single step.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

Note: Application authenticity and Device provisioning are not supported in Java Platform, Micro Edition (Java ME).

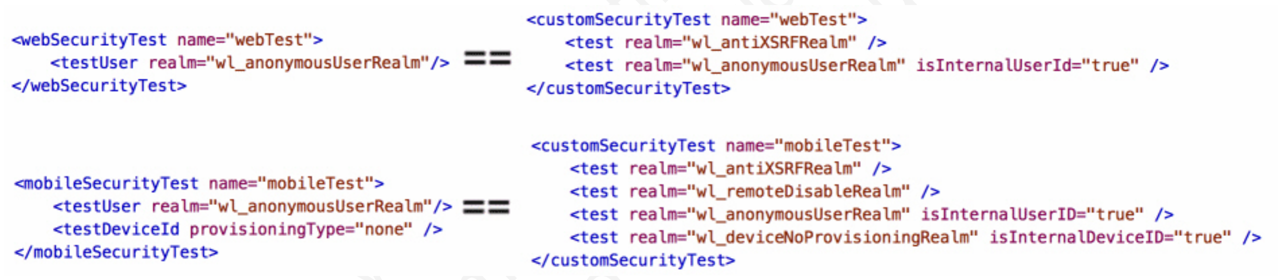
Sample security tests

The following figure shows what a webSecurityTest and a mobileSecurityTest contain. The security tests on the right are detailed equivalent of the security tests on the left.

The webSecurityTest contains:
- The following realms, enabled by default: `wl_anonymousUserRealm` and `wl_antiXSRFRealm`.
- The user realm that you must specify.

The mobileSecurityTest contains:
- The following realms, enabled by default: `wl_anonymousUserRealm`, `wl_antiXSRFRealm`, `wl_remoteDisableRealm` and `wl_deviceNoProvisioningRealm`.
- The user and device realms that you must specify.

A customSecurityTest has no realms that are enabled by default. You must define all realms that you want your customSecurityTest to contain.



```
<webSecurityTest name="webTest">
    <testUser realm="wl_anonymousUserRealm"/>
</webSecurityTest>
```

==

```
<customSecurityTest name="webTest">
    <test realm="wl_antiXSRFRealm" />
    <test realm="wl_anonymousUserRealm" isInternalUserId="true" />
</customSecurityTest>
```

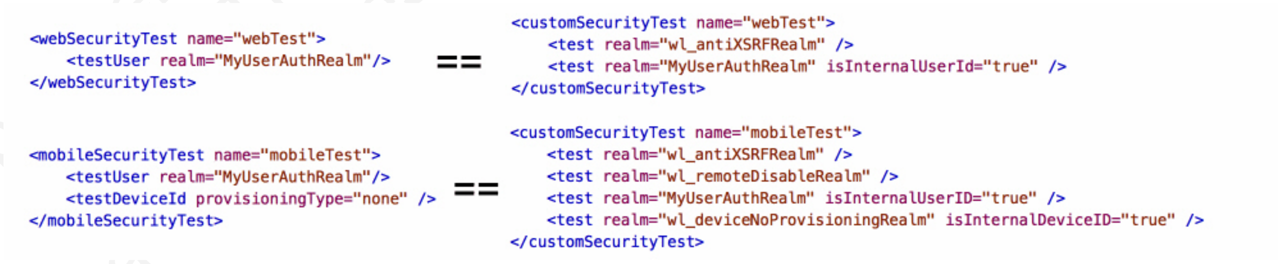
```
<mobileSecurityTest name="mobileTest">
    <testUser realm="wl_anonymousUserRealm"/>
    <testDeviceId provisioningType="none" />
</mobileSecurityTest>
```

==

```
<customSecurityTest name="mobileTest">
    <test realm="wl_antiXSRFRealm" />
    <test realm="wl_remoteDisableRealm" />
    <test realm="wl_anonymousUserRealm" isInternalUserID="true" />
    <test realm="wl_deviceNoProvisioningRealm" isInternalDeviceID="true" />
</customSecurityTest>
```

Figure 25. Examples of webSecurityTest, mobileSecurityTest, and their equivalent as a customSecurityTest

Usually, you add your own realm to your configuration to authenticate users. The following figure shows a configuration where the realm named MyUserAuthRealm is the realm that the developer added.



```
<webSecurityTest name="webTest">
    <testUser realm="MyUserAuthRealm"/>
</webSecurityTest>
```

==

```
<customSecurityTest name="webTest">
    <test realm="wl_antiXSRFRealm" />
    <test realm="MyUserAuthRealm" isInternalUserId="true" />
</customSecurityTest>
```

```
<mobileSecurityTest name="mobileTest">
    <testUser realm="MyUserAuthRealm"/>
    <testDeviceId provisioningType="none" />
</mobileSecurityTest>
```

==

```
<customSecurityTest name="mobileTest">
    <test realm="wl_antiXSRFRealm" />
    <test realm="wl_remoteDisableRealm" />
    <test realm="MyUserAuthRealm" isInternalUserID="true" />
    <test realm="wl_deviceNoProvisioningRealm" isInternalDeviceID="true" />
</customSecurityTest>
```

Figure 26. Examples with your own realm name as a realm definition for testUser

Authentication realms

Resources are protected by *authentication realms*. Authentication processes can be interactive or non-interactive.

An authentication realm defines the process to be used to authenticate users, and consists of the following steps:

1. Specification of how to collect user credentials, for example, by using a form, using basic HTTP authentication or using SSO.
2. Specification of how to verify the user credentials, for example, checking that the password matches the user name, or using an LDAP server or some other authentication server.
3. Specification of how to build the user identity, that is, how to build objects that contain all the necessary user properties.

The same realm can be used In different security tests. In this case, clients must undergo the authentication process that is defined for the realm only once

Authentication processes can be interactive or non-interactive, as demonstrated in the following authentication process examples:
- An example of interactive authentication is a login form that is displayed when a user attempts to access a protected resource. The authentication process includes verifying the user credentials.
- An example of non-interactive authentication is a user cookie that the authentication process looks for when a user attempts to access a protected resource. If there is a cookie, this cookie is used to authenticate the user. If there is no cookie, a cookie is created, and this cookie is used to authenticate the user in the future.

Authenticators and Login Modules

An authenticator collects client credentials. A login module validates them.

An authenticator is a server component which is used to collect credentials from the client. The authenticator passes the credentials to a login module, which validates them and builds a client identity object.

An authenticator can, for example, collect any type of information accessible from an HTTP request object, such as cookies or any data in headers or the body of the request.

A login module can validate the credentials that are passed to it in various ways. For example:
- Using a web service
- Looking up the client ID in a database
- Using an LTPA token

A number of predefined authenticators and login modules are supplied. If these do not meet your needs, you can write your own in Java.

The authentication configuration file

All types of authentication component are configured in the authentication configuration file.

Authentication components, security tests, realms, login modules, and authenticators are all configured in the authentication configuration file, `authenticationConfig.xml`, which is in the `/server/conf` directory of the Worklight project. A web security test or mobile security test must contain a <testUser> element that specifies realm name. The definition of a realm includes the class name of an authenticator, and a reference to a login module. Authenticators are the

Experimental IBM Worklight offline user documentation. This document is provided as is. In case of issues refer to the online user documentation.

entities which authenticate clients. Authenticators collect client information, and then use login modules to verify this information.

Table 22. Predefined realms: properties of the `<test realm>` element.

Authenticator class name	Login module reference	Description
`wl_anonymousUserRealm`	**`wl_anonymousUserLoginModule`**	Implement persistent cookie authentication
`wl_remoteDisableRealm`	**`wl_remoteDisableLoginModule`**	Implement remote disable
`wl_antiXSRFRealm`	**`wl_antiXSRFLoginModule`**	Implement anti-XSRF header check.
`wl_noDeviceProvisioningRealm`	**`wl_noDeviceProvisioningLoginModule`**	Implement device authentication without provisioning
`wl_autoDeviceProvisioningRealm`	**`wl_autoDeviceProvisioningLoginModule`**	Implement device authentication with auto-provisioning

IBM Worklight static resources (other than Application Center) such as the Worklight Console are also configured in the authentication configuration file, in the **`<resource>`** element.

The configuration file has the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:loginConfiguration xmlns:tns="http://www.worklight.com/auth/config" xmlns:xsi="http://www.w3.
<staticResources>
<resource> ... </resource>
<resource> ... </resource>
</staticResources>
<realms>
<realm> ... </realm>
<realm> ... </realm>
</realms>
<loginModules>
<loginModule> ... </loginModule>
<loginModule> ... </loginModule>
</loginModules>
</tns:loginConfiguration>
```

Configuring IBM Worklight web application authorization

Configure authentication to the Worklight Administration Console, usage reports, and Application Center.

The Worklight web applications that require authentication are the IBM Worklight Administration Console, the IBM Worklight usage reports, and the IBM Worklight Application Center console. The Worklight Administration Console and Worklight usage reports are configured by using `<resource>` elements in the `authenticationConfig.xml` file.

The IBM Worklight Application Center console is not subject to the authentication model described here. For information about setting up authentication for the Application Center console, see "Configuration of the Application Center after installation" on page 429.

Configuring Authenticators and Realms

Authenticators are defined within the realm that uses them.

Experimental IBM Worklight offline user documentation. This document is provided as is. In case of issues refer to the online user documentation.

Realms are defined in <realm> elements in the authenticationConfig.xml file. The <realms> element contains a separate <realm> subelement for each realm.

Modify realms by using the authentication configuration editor.

The <realm> element has the following attributes:

Table 23. The <realm> element attributes

Attribute	Description
name	Mandatory. The unique name by which the realm is referenced by the protected resources. **Note:** If a realm uses a Facebook authenticator, its name must start with "facebook.". The use of Facebook realms is deprecated in Worklight version 5.0.5. Support might be removed in any future version.
loginModule	Mandatory. The name of the login module used by the realm.

The <realm> element has the following subelements:

Table 24. The <realm> element subelements

Element	Description
<className>	Mandatory. The class name of the authenticator. For details of the supported authenticators, see the following topics.
<parameter>	Optional. Represents the name-value pairs which are passed to the authenticator upon instantiation. This element may appear multiple times.
<onLoginUrl>	Optional. Defines the path to which the client is forwarded upon successful login. If this element is not specified, then depending on the authenticator type, either the current request processing is continued, or a saved request is restored.

Basic authenticator

Description and syntax of the basic authenticator.

Description

The basic authenticator implements basic HTTP authentication.

Note: You can use the basic authenticator only for web applications, not for mobile applications.

Experimental IBM Worklight offline user documentation
This document is provided as is.
In case of issues, refer to the online user documentation.

Class Name

`com.worklight.core.auth.ext.BasicAuthenticator`

Parameters

The basic authenticator has the following parameters:

Parameter	**Description**
`<basic-realm-name>`	Mandatory. A string that is sent to the client as a realm name, and presented by the browser in the login dialog.

```
<realm name="realmForMyApp" loginModule="DatabaseLoginModule">
  <className> com.worklight.core.auth.ext.BasicAuthenticator </className>
  <parameter name="basic-realm-name" value="My App" />
</realm>
```

Form-based authenticator

Description and syntax of the form-based authenticator.

Description

The form-based authenticator presents a login form to the user. The login form must contain fields named `j_username` and `j_password`, and the submit action must be `j_security_check`. If the login fails, the user is redirected to an error page.

Class Name

`com.worklight.core.auth.ext.FormBasedAuthenticator`

Parameters

The form-based authenticator has the following parameters:

Parameter	Description
`login-page`	Mandatory. Path to the login page, relative to the web application context under the `conf` directory. A sample `login.html` file is provided under this directory when creating a Worklight project in the Worklight Studio. The authenticator passes to the login page error messages. To display the error message, use the placeholder `${errorMessage}` within your login page, as depicted in the example.
`auth-redirect`	If the login form is not stored locally in the web application, provide a URL to the login page in this parameter. The Worklight Server uses redirect to pass the login to the external domain. This parameter cannot be used together with the `login-page` parameter.

Experimental IBM Worklight offline user documentation.
This document is provided "as is".
In case of issues, refer to the online user documentation.

```
<realm name="AppAuthRealm" loginModule="DatabaseLoginModule">
<className> com.worklight.core.auth.ext.FormBasedAuthenticator </className>
<parameter name="login-page" value="login.html" />
</realm>
```

Header authenticator

Description and syntax of the header authenticator.

Description

The header authenticator is not interactive. The header authenticator must be used with the Header login module.

Class Name

`com.worklight.core.auth.ext.HeaderAuthenticator`

Parameters

None.

```
<realm name="RealmHeader" loginModule="HeaderLoginModule">
<className> com.worklight.core.auth.ext.HeaderAuthenticator </className>
</realm>
```

Persistent cookie authenticator

Description and syntax of the persistent cookie authenticator.

Description

The persistent cookie authenticator looks for a specific cookie in any request that is sent to it. If the request does not contain the cookie, the authenticator creates a cookie, and sends it in the response. This authenticator is not interactive, that is, it does not ask the user for credentials, and is mainly used in environment realms.

Class Name

`com.worklight.core.auth.ext.PersistentCookieAuthenticator`

Parameters

The persistent cookie authenticator class has the following parameter:

Parameter	Description
`<cookie-name>`	Optional. The name of the persistent cookie. If this parameter is not specified, the default name, `WL_PERSISTENT_COOKIE`, is used.

```
<realm name="PersistentCookie" loginModule="dummy">
<className> com.worklight.core.auth.ext.PersistentCookieAuthenticator </className>
</realm>
```

Adapter authenticator

Description and syntax of the adapter authenticator

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Description

Use the adapter authenticator to develop custom authentication logic in JavaScript within an adapter. You generally use this technique to define multi-step login processes that you cannot implement simply by configuring another type of authenticator, such as a basic authenticator.

You can use an adapter authenticator to protect adapter procedures only.

Important: The adapter authenticator code performs all the validations and the creation of the user identity. You must use it with a "Non-validating login module" on page 147.

Class Name

`com.worklight.integration.auth.AdapterAuthenticator`

Parameters

The adapter authenticator class has the following parameters:

Parameter	Description
`<login-function>`	Mandatory. The name of the JavaScript function, in the format <adapter-name.function-name>, which is invoked when the login is triggered (depending on the configuration, either when the client app explicitly invoked `WL.Client.login` or if it tried accessing a protected procedure). This function receives as an input parameter the request headers so that it can access the user agent and other information passed on the request. This function should not be exposed as a procedure by the adapter.
`<logout-function>`	Optional. The name of the JavaScript function, in the format <adapter-name.function-name>, which is invoked when the session is terminated (either when the client app invoked `WL.Client.logout` or when the Server decided to terminate the session). This function receives no parameters. It should not be exposed as a procedure by the adapter.

Example

```
<realm name="ACMERealm" loginModule="ACMELoginModule">
<className> com.worklight.integration.auth.AdapterAuthenticator </className>
<parameter name="login-function" value="ACMEAuthAdapter.triggerLogin" />
<parameter name="logout-function" value="ACMEAuthAdapter.logout" />
</realm>
```

LTPA authenticator

Description and syntax for the LTPA authenticator.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Description

Use the Lightweight Third-Party Authentication authenticator to integrate with the WebSphere Application Server LTPA mechanisms.

Note: This authenticator is supported only on WebSphere Application Server. To avoid unnecessary errors on other application servers, the authenticator is commented out in the default `authenticationConfig.xml` file that is created with an empty Worklight project. To use it, remove the comments first.

This authenticator can be used with the `WASLTPAModule` login module.

Class Name

`com.worklight.core.auth.ext.WebSphereFormBasedAuthenticator`

Parameters

The adapter authenticator class has the following parameters:

Parameter	Description
`login-page`	Mandatory. The login page URL relative to the web application context.
`error-page`	Mandatory. The error page URL relative to the web application context.
`cookie-domain`	Optional. A String such as `example.com`, which specifies the domain in which the LTPA SSO cookie applies. If this parameter is not set, no domain attribute is set on the cookie. The single sign-on is then restricted to the application server host name and does not work with other hosts in the same domain.
`httponly-cookie`	Optional. A String with a value of either `true` or `false`, which specifies whether the cookie has the HttpOnly attribute set. This attribute helps to prevent cross-site scripting attacks.
`cookie-name`	Optional. A String that specifies the name of the LTPA SSO cookie. If this parameter is not set, the default cookie name is `LtpaToken`.

Example

```
<realm name="WASLTPARealm" loginModule="WASLTPAModule">
<className>com.worklight.core.auth.ext.WebSphereFormBasedAuthenticator</className>
<parameter name="login-page" value="/login.html"/>
<parameter name="error-page" value="/loginError.html"/>
<parameter name="cookie-domain" value="example.com"/>
<parameter name="httponly-cookie" value="true"/>
<parameter name="cookie-name" value="LtpaToken2"/>
</realm>
```

Attributes of login modules

Login modules are defined in `<loginModule>` elements in the `authenticationConfig.xml` file.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

The `<loginModules>` element contains a separate `<loginModule>` subelement for each login module.

The `<loginModule>` element has the following attributes:

Attribute	Description
`name`	Mandatory. The unique name by which realms reference the login module.
`audit`	Optional. Defines whether login attempts that use the login module are logged in the audit log. The log file is *Worklight Project Name*`/server/log/audit/audit.log`. Valid values are: **`true`** Login and logout attempts are logged in the audit log. **`false`** Default. Login and logout attempts are not logged in the audit log.

The `<loginModule>` element has the following subelements:

Element	Description
`<className>`	Mandatory. The class name of the login module. For details of the supported login modules, see the following topics.
`<parameter>`	Optional. An initialization property of the login module. The supported properties and their semantics depend on the login module class. This element can occur multiple times.

Non-validating login module

The non-validating login module accepts any user name and password passed by the authenticator.

Class Name

`com.worklight.core.auth.ext.NonValidatingLoginModule`

Parameters

None

```
<loginModule name="dummy" canBeResourceLogin="false" isIdentityAssociationKey="true">
<className> com.worklight.core.auth.ext.NonValidatingLoginModule </className>
</loginModule>
```

Database login module

The database login module verifies the user name and password by executing an SQL query.

Experimental IBM Worklight offline user documentation. This document is provided as-is. In case of issues, refer to the online user documentation.

Class Name

`com.worklight.core.auth.ext.RDBMSLoginModule`

Parameters

The database login module has the following parameters:

Parameter	Description
`dsJndiName`	Mandatory. The JNDI name of the data source that defines the database.
`principalsQuery`	Mandatory. The SQL query text. The query has the following structure: `SELECT` *`UserID`*`,` *`Password`*`,` *`DisplayName`* `FROM` *`UsersTable`* `WHERE` where: *UserID*, *Password*, and *DisplayName* are the names of the columns that contain user login names, passwords, and display names *UserTable* is the database table that contains this data.

```
<loginModule name="MyDatabase" canBeResourceLogin="true" isIdentityAssociationKey="true">
<className> com.worklight.core.auth.ext.RDBMSLoginModule </className>
<parameter name="dsJndiName" value="java:/MyDS"/>
<parameter name="principalsQuery">
SELECT userid, password, concat(firstName, ' ', lastName) as display_name FROM users WHERE userid=?
</parameter>
</loginModule>
```

Single identity login module

The single identity login module is used to grant access to the Worklight Console to a single user, the identity of which is defined in the `worklight.properties` file.

Class Name

`com.worklight.core.auth.ext.SingleIdentityLoginModule`

Parameters

None

Configuration

.The `worklight.properties` file must contain the following properties:

Key	Description
`console.username`	Name of the user who can access the Console
`console.password`	Password of the user who can access the Console. The password can be encrypted as indicated in "Storing properties in encrypted format" on page 411.

```
<loginModule name="Console" canBeResourceLogin="false" isIdentityAssociationKey="false">
<className> com.worklight.core.auth.ext.SingleIdentityLoginModule </className>
</loginModule>
```

Experimental IBM Worklight online user documentation.
This document is provided "as is".
In case of issues, refer to the online user documentation.

Header login module

The Header login module is always used with the Header authenticator. It validates the request by looking for specific headers.

Class Name

com.worklight.core.auth.ext.HeaderLoginModule

Parameters

The Header login module has the following parameters:

Parameter	Description
user-name-header	Mandatory. The name of the header that contains the user name. If the request does not contain this header, the authentication fails.
display-name-header	Optional. The name of the header that contains the display name. If this parameter is not specified, the user name is used as the display name.

```
<loginModule name="HeaderLoginModule" audit="true">
  <className>com.worklight.core.auth.ext.HeaderLoginModule</className>
  <parameter name="user-name-header" value="userid"/>
  <parameter name="display-name-header" value="username"/>
</loginModule>
```

WASLTPAModule login module

The WASLTPAModule login module enables integration with WebSphere Application Server LTPA mechanisms.

Note: This login module is only supported on WebSphere Application Server. To avoid unnecessary errors when Worklight is run on other application servers, the login module is commented out in the default `authenticationConfig.xml` file that is created with an empty Worklight project. To use it, remove the comments first.

Class Name

com.worklight.core.auth.ext.WebSphereLoginModule

Parameters

None.

```
<loginModule name="WASLTPAModule" canBeResourceLogin="true" isIdentityAssociationKey="false">
<className>com.worklight.core.auth.ext.WebSphereLoginModule</className>
</loginModule>
```

LDAP login module

You can use the LDAP login module to authenticate users against LDAP servers, for example Active Directory, or OpenLDAP.

LDAP login module implements a `UserNamePasswordLoginModule` interface, so you must use it with an authenticator that implements a `UsernamePasswordAuthenticator` interface.

Experimental IBM Worklight online user documentation. This document is provided "as-is". In case of issues, refer to the online user documentation.

Class Name

`com.worklight.core.auth.ext.LdapLoginModule`

Parameters

You must set the following parameters for the LDAP login module:

Parameter	Description
`ldapProviderUrl`	Mandatory. The IP address or the URL of the LDAP server.
`ldapTimeoutMs`	Mandatory. The connection timeout to the LDAP server in milliseconds.
`ldapSecurityAuthentication`	Mandatory. The LDAP security authentication type. The value is usually `simple`. Consult your LDAP a obtain the relevant authentication type.
`validationType`	Mandatory. The type of validation. The value can be `exists`, `searchPattern`, or `custom`. See the followin more details.
`ldapSecurityPrincipalPattern`	Mandatory. Depending on the LDAP server type, this parameter might require security credentials that supply in several formats. Some LDAP servers require only the user name, for example *john*, and other user name and the domain, for example *john@server.com*. You use this property to define the pattern to user name based credentials. You can use the `{username}` placeholder.
`ldapSearchFilterPattern`	Optional. This parameter is required only if the value of the `validationType` parameter is `searchPatter` parameter to define a search filter pattern that is run when a successful LDAP binding is established. T validation is successful if the search returns one or more entries. You can use the `{username}` placehold might change depending on the LDAP server type.
`ldapSearchBase`	Optional. This parameter is required only if the `validationType` parameter is `searchPattern`. Use this p define the base of the LDAP search.

Sample LDAP login module definition:

```
<loginModule name="LDAPLoginModule">
 <className>com.worklight.core.auth.ext.LdapLoginModule</className>
 <parameter name="ldapProviderUrl" value="ldap://10.0.1.2"/>
 <parameter name="ldapTimeoutMs" value="2000"/>
 <parameter name="ldapSecurityAuthentication" value="simple"/>
 <parameter name="validationType" value="searchPattern"/>
 <parameter name="ldapSecurityPrincipalPattern" value="{username}@myserver.com"/>
 <parameter name="ldapSearchFilterPattern" value="(&(objectClass=user)(sAMAccountName={username})(mem
 <parameter name="ldapSearchBase" value="dc=myserver,dc=com"/>
</loginModule>
```

Values of the validationType parameter

Value	Description
`exists`	The login module tries to establish the LDAP binding with the supplied credentials. The credentials validation is successful if the binding is successfully established.

Experimental IBM Worklight online user documentation
This document is provided "as is".
In case of ... prior to the online user documentation.

Value	Description
`searchPattern`	The login module tries to do the `exists` validation. When the validation succeeds, the login module issues a search query to the LDAP server context, according to the `ldapSearchFilterPattern` and `ldapSearchBase` parameters. The credentials validation is successful if the search query returns one or more entries.
`custom`	With this value, you can implement custom validation logic. The login module tries to do the `exists` validation. When the validation succeeds, the login module calls a `public boolean doCustomValidation(LdapContext ldapCtx, String username)` method. To override this method, you must create a custom Java class in your Worklight project and extend from `com.worklight.core.auth.ext.UserNamePasswordLoginModule`. See the following example.

Sample custom validation implementation:

```
package mycode;
import javax.naming.ldap.LdapContext;
import com.worklight.core.auth.ext;

public class MyCustomLdapLoginModule extends LdapLoginModule {

    @Override
    public boolean doCustomValidation(LdapContext ldapCtx, String username, String password) {

        boolean success = true;

        // Do some custom validations here using ldapCtx, validationProperties and username
        // Return true in case of validation success and false otherwise

        return success;
    }

}
```

Note:

After you implement your custom extension of `LdapLoginModule`, use it as a `className` value of LoginModule in your `AuthenticationConfig.xml` file.

You can also override other public methods of `LdapLoginModule`. See “Deprecated interface WorkLightLoginModule” on page 307 for more details.

Scope of mobile device authentication

You can require mobile devices to authenticate themselves. Device identity is used in several places within the Worklight platform.

In addition to requiring users to authenticate before they access certain resources, you can also require mobile devices to authenticate before apps installed on them can access the Worklight Server.

Device and application authentication is a process that allows making claims of type "this is application A installed on device D".

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues refer to the online user documentation.

Device and application authentication is relevant only for applications that are installed on mobile devices.

Mobile device provisioning

Provisioning is the process of obtaining a security certificate. There are three modes of the provisioning process.

When a Worklight application first runs on a mobile device, it creates a pair of PKI-based keys. It then uses the keys to sign the public characteristics of the device and application, and sends them to the Worklight Server for authentication purposes.

A key pair alone is not sufficient to sign these public characteristics because any app can create a key pair. In order for a key pair to be trusted, it must be signed by an external trusted authority to create a certificate. The process of obtaining such a certificate is called *provisioning*.

After a certificate is obtained, the app can store the key pair in the device keystore, access to which is protected by the operating system.

The provisioning process has three modes:

No provisioning
In this mode, the provisioning process does not happen. This mode is suitable only during the development cycle, to temporarily disable the provisioning for the application. Technically, the client application does not trigger the provisioning process, and the server does not verify the client certificate.

Auto-provisioning
In this mode, the Worklight Server automatically issues a certificate for the device and application data that are provided by the client application. Use this option only when the Worklight application authenticity features are enabled.

Custom provisioning
In this mode, the Worklight Server is augmented with custom logic that controls the device and application provisioning process. This logic can involve integration with an external system, such as a mobile device manager (MDM). The external system can issue the client certificate based on an activation code that is obtained from the app, or can instruct the Worklight Server to do so.

Note: Auto-provisioning and custom provisioning are supported only on iOS and Android devices.

Device auto-provisioning

Device auto-provisioning has three aspects:
- Provisioning granularity: the scope of the provisioned entity.
- Pre required login: the realms that a client needs to be authenticated with before it can get permission to perform provisioning.
- CA Certificate: the parent certificate which issues device certificates for the provisioning process.

The default behavior is as follows:

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

- Provisioning granularity: a single application.
- Pre required login: a login is required to the authentication realm, if any, defined for the current security test.
- CA Certificate: a Worklight CA Certificate, which is embedded into the platform.

Whether it is obtained by an auto-provisioning or custom provisioning process, the certificate is stored by the client app on the device, and used for signing the payload sent to the Worklight Server. The Worklight Server validates the client certificate, regardless of how it is obtained.

The server sends a request for ID, which the client responds to with a certificate-signed payload. If the client does not have the certificate, then a request is sent to the Worklight server automatically to get a certificate, and after that is done, the client automatically sends the signed payload.

After the server sends the `ok` response, the original request is sent automatically.

Granularity of provisioning

The key pair that is used to sign the device and app properties can represent a single application, a group of applications, or an entire device. For example:

Single application
A company's provisioning process requires separate activation for each application that is installed on the device. In this case, the application is the provisionable entity, and each application must generate its own key pair.

Group of applications
A company develops different groups of applications to employees in different geographical regions. If the activation is required per region, the key pair would represent the group of applications that belong to that region. All applications from the same group use the same key pair for their signatures.

Entire device
In this case, the key pair represents the whole device. All the applications from the same vendor that are installed on that device use the same key pair.

Configuring and implementing device provisioning

You can change the default behavior with regard to CA certificates. You can also implement custom provisioning.

Procedure

- To use a CA certificate other than the default Worklight CA certificate, configure the following properties in the `worklight.properties` file:

  wl.ca.keystore.path
  The path to the keystore, relative to the server folder in the Worklight Project, for example: `conf/default.keystore`.

  wl.ca.keystore.type
  The type of the keystore file. Valid values are `jks` or `pkcs12`.

  wl.ca.keystore.password
  The password to the keystore file, for example: `worklight`.

Experimental IBM Worklight user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

wl.ca.key.alias
The alias of the entry where the private key and certificate are stored, in the keystore, for example: `keypair1`.

wl.ca.key.alias.password
The password to the alias in the keystore for example: `worklight`.

- If you want to change the provisioning mechanism to use different granularity (application, device or group) or a different list of pre-required realms, you can create your own customized authenticator, login module and challenge handler. For more information about custom authentication, see the module *Custom Authenticator and Login Module* under category 8, *Authenticator and security*, in "Getting started tutorials and samples" on page 29.

Device single sign-on (SSO)

Single sign-on (SSO) enables users to access multiple resources (that is, applications and adapter procedures) by authenticating only once.

When a user successfully logs in through an SSO login module, the user gains access to all resources that are using the same SSO login module, without having to authenticate again for each of them. The authenticated state remains alive as long as requests to resources protected by the login module are being issued within the timeout period, which is identical to the session timeout period.

Device authentication

The SSO feature requires the use of device authentication. This means that for a protected resource that needs to be protected with SSO, there must also be a device authentication realm in the securityTest protecting the resource in the `authenticationConfig.xml` file. Device authentication should take place before the SSO-enabled user authentication.

Supported devices

SSO is supported on Android and iOS devices.

Performance

When you use the single sign-on feature, the load on the database might increase, and you might have to adjust the database configuration.

Configuring device single sign-on

Assign a common identifier in the application descriptor file for each application that is to be accessed by single sign-on, and enable single sign-on from a mobileSecurityTest element or from a customSecurityTest element.

Procedure

- For each application (or for each adapter accessed by those applications), assign the same value to the following attributes in the application descriptor file:
  - For Android, assign the same **`sharedUserId`** value for each application. The following example assigns the value `"my.sso"`:

```
<android version="1.0" sharedUserId="my.sso">
...
</android>
```

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues refer to the online user documentation.

  - For iOS, assign the same AppID prefix for each application. The AppID prefix is defined when you create new AppID on the Apple Developer Provisioning Portal.

  For information about the application descriptor file, see "The application descriptor" on page 55.

- When configuring mobileSecurityTest elements, enable single sign-on from the securityTest element by setting the value of the **sso** attribute to `true`. Note that you can enable SSO for user realms only. For example:

```
<mobileSecurityTest name="mst">
  <testDeviceId provisioningType="none"/>
  <testUser realm="myUserRealm" sso="true"/>
</mobileSecurityTest>
```

- When configuring customSecurityTest elements, enable single sign-on by configuring an **ssoDeviceLoginModule** property on the user login module in the authentication configuration file. For example:

```
<loginModule name="MySSO" ssoDeviceLoginModule="WLDeviceNoProvisioningLoginModule">
  <className>com.worklight.core.auth.ext.NonValidatingLoginModule</className>
</loginModule>
```

  In this example, `"MySSO"` is the name of the user login module for which single sign-on is being enabled so that its login can be shared. `"WLDeviceNoProvisioningLoginModule"` is the name of the login module that handles device authentication; in this case, with no provisioning. To use auto-provisioning as the device login module, set the **ssoDeviceLoginModule** property to the value `"WLDeviceAutoProvisioningLoginModule"`. With custom provisioning, you define the name when you create the custom provisioning login module.

- When configuring customSecurityTest elements, you must configure the user realm at least one step later than the device realm. This is necessary to ensure that the SSO feature operates correctly. The following example illustrates a correct customSecurityTest configuration:

```
<customSecurityTest name="adapter">
  <test realm="wl_deviceNoProvisioningRealm" isInternalDeviceID="true" step="1"/>
  <test realm="MySSO" isInternalUserID="true" step="2"/>
</customSecurityTest>
```

- A cleanup task cleans the database of orphaned and expired single-sign-on login contexts. To configure the cleanup task interval, uncomment the **sso.cleanup.taskFrequencyInSeconds** setting in the worklight properties file and assign the required task interval value expressed in seconds. For information about the worklight properties file, see "IBM Worklight properties" on page 406.

Data synchronization with JSONStore

You can write an application that maintains a local copy of its data and, on request, pushes the local updates to a back-end service.

The local copy is a JSON data store. IBM Worklight supplies an API for working with a JSON Store through the methods of the JavaScript class WL.JSONStore.

Using the JSONStore API, you can store data locally and push changes from the client to a server. You can search the local data store and update and delete data within it. You can secure the local data store by using password-based encryption.

Most of the operations that are provided in the API for using data synchronization, operate on the local copy of the data that is stored on the device. **add**, **replace**,

Experimental IBM Worklight user documentation. This document is provided "as is". In case of issues, use the online user documentation.

`remove` and all other operations are specific to the local copy of the data that is stored by the JSON data store. The JSON data store tracks modifications to data made locally. The exceptions are `push`, `pushSelected`, and `load`, which act on the local and remote data.

You can push the local modifications to a Worklight server adapter. Push the data to the adapter by using the `push` and `pushSelected` methods. The `push` call attempts to push all the local modifications to the adapter. The `pushSelected` operates only on the set of documents you specify.

Multiple stores are supported per application, and each of those stores can hold many collections. Refer to WL.JSONStore.initCollection API for details on how to initialize different stores and collections.

To relate a collection to an adapter, you must specify the `adapter` option as part of the **collection creation** options. You do not have to associate a collection with an adapter. If an adapter is not specified for the collection, calls to `push` and `pushSelected` return an error.

For more information, see WL.JSONStore .

Developing an app that uses data synchronization

Create an app and adapter, then use a wizard to create the JavaScript for handling a local data store, and incorporate it in your app.

About this task

You can use the advanced data management and synchronization feature to store data securely on a mobile device. It provides full access to the local copy of the data while tracking modifications made locally to be synchronized with a Worklight adapter when appropriate.

A typical use of this feature is as follows:
- A mobile worker whose job includes customer visits downloads a large set of information to a mobile device when network conditions are adequate (for example, in the office, using the corporate WiFi network)
- The downloaded data is securely stored on the device.
- The worker uses the local copy of the data in an application on the mobile device, with the device either online or offline.
- The Worklight SDK tracks any changes that are made to the local copy of the data.
- At an appropriate time, perhaps back in the office at the end of the week and again connected to the corporate WiFi network, the worker synchronizes the updates that were made locally on the device with a Worklight adapter that pushes the updates into a back-end system.

Procedure

1. In Worklight Studio, create an application.
   a. In the Project Explorer tab, right-click the project name.
   b. Click **New** > **Worklight Hybrid Application**. The Hybrid Application window opens.
   c. Complete the fields in this window appropriately and click **Finish**. A standard application structure is created.

Experimental IBM Worklight online user documentation. This document is provided "as is". In case of issues refer to the online user documentation.

2. In Worklight Studio, create and deploy an adapter.
   a. In the Project Explorer tab, right-click the project name.
   b. Click **New** > **Worklight Adapter**. The New Worklight Adapter window opens.
   c. Select the appropriate adapter type, enter an adapter name, and select the **JSON Data available offline check box**.
   d. Optional: To change the suggested procedure names, type over them.
   e. Click **Finish**. A standard adapter structure is created.
   f. Deploy the adapter.
3. Retrieve a JSON object with the adapter:
   a. Right-click the adapter name.
   b. Click **Run As** > **Invoke Worklight Procedure**. The Edit Configuration Window is displayed
   c. Select the procedure that is used for retrieving JSON data and click **Run**.

   The JSON object that is returned by the procedure is displayed.
4. Create a local JSON store:
   a. In Worklight Studio, click **File** > **New** > **Worklight JSON Store** and select the project and app names. The Create JSON Collection wizard starts.
   b. Follow the instructions in the wizard to invoke the adapter, name the collection, and specify the searchable fields.
   c. Optional: To encrypt collections for an application, select the **Encrypt collections** check box in the wizard.

   The wizard creates a JavaScript file named `*collection_name*Collection.js` in the application's `common/js` directory, where *collection_name* is the name you specified in the wizard.
5. Review the `*collection_name*Collection.js` file and include its content manually in your application's `.js` file.

Encrypting collections

You can choose to encrypt collections for an app, but you cannot switch between encrypted and plain-text formats, or mix formats within an app.

About this task

You can specify that all collections created for an app are encrypted in the JSON data store. It is not possible to have some collections encrypted and others not encrypted in the same app. When a single collection is encrypted, the underlying storage mechanism encrypts all further collections. After a collection is created in either clear text or encrypted, you cannot convert it from one format to the other. A collection that is created in clear text would have to be destroyed and re-created with the usePassword option and reloaded with data and vice versa.

Procedure

To specify that the collections are to be encrypted use the usePassword method when you create a collection. For more information, see usePassword.

Troubleshooting information for synchronization

There are various ways in which a synchronization can fail. A status code is returned.

Experimental IBM Worklight user documentation
This document is provided "as-is".
In case of issues, refer to the online user documentation.

The **push** and **pushSelected** methods operate similarly, except that **push** takes no parameters (it synchronizes all the documents that were modified locally) and **pushSelected** takes the ID of a specific document or documents. Either function fails if the wrong parameters are passed to it.

They can also fail if something goes wrong while native code is running to get the documents or verify that they are unsynchronized.

After the documents are retrieved, a Worklight adapter is called, using the appropriate option (`add`, `replace`, or `remove`). The adapter then tries to contact the back-end server. This attempt can fail if the adapter or procedure name does not exist (perhaps because it is misspelled) or if the Worklight Server or back-end server cannot be reached.

If the server is successfully contacted, you can check the status code from the synchronization procedure and determine whether to mark that document as synchronized. This step uses native code and can fail.

All the failure paths are handled and return status codes to help you to mitigate failure conditions. Status codes are listed at "List of error codes" on page 218.

Client-side log capture

Applications in the field occasionally experience problems that require a developer's attention to fix. It is often difficult to reproduce problems in the field because developers who worked on the code for the problem application often do not have the environment or exact device with which to test. In these situations, it is helpful to be able to retrieve debug logs from the client devices as the problems occur in the environment in which they happen.

To make debug logs effective, developers should produce meaningful log messages with an appropriate level. For example:

```
[WARN] Procedure sayHello timed out due to a network connection failure.
```

Questions to consider

Consider the following questions, and make the appropriate API calls to the native client logger API to achieve your goals:

- When should you turn on log capture in your client applications?
  - Leave log capture on?
  - Turn log capture on selectively for applications or operating systems that are known to be problematic?
  - Turn log capture on the second Tuesday of every month?
- When should you call `send()` to upload any captured client logs?
  - On a specific time interval?
  - In application lifecycle events (like pause and resume events)?
  - Batched with other application network activity (to be friendly to the device radio or letting it sleep and preserve battery)?
- What level and above do you want to capture?
  - DEBUG is verbose, while INFO is nearly quiet.
  - The JavaScript level is controlled independently from the native level, but the native level can be set by using the `WL.Logger.setNativeOptions` JavaScript API call.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

- How large should you let the capture buffer grow before you stop capturing?
- Where can you strategically place log API calls to see the required data to solve problems in the field?
- How can you process the uploaded logs at the server?
  - Forward them to an analytics product?
  - Print them into the server-side log file?

What is provided on the client side?

Android
: `com.worklight.common.Logger`

  Note: Native Android code that calls the `android.util.Log.*` API is not captured in the client-side logs. Developers must use `com.worklight.common.Logger` to capture client-side logs. For more information about the `com.worklight.common.Logger` API, see "Java client-side API for native Android apps" on page 271.

iOS
: `OCLogger`

  Note: Native iOS code that calls `nslog` directly is not captured in the client-side logs. Developers must use `OCLogger` to capture client-side logs. For more information about the `OCLogger` API, see "Objective-C client-side API for native iOS apps" on page 271.

JavaScript
: `WL.Logger`

  Note: JavaScript code that calls `console.log` directly is not captured in the client-side logs. Developers must use `WL.Logger` to capture client-side logs. For more information about the `WL.Logger` API, see "The WL.Logger object" on page 241.

Server preparation for uploaded log data

You must prepare your server for uploaded log data.

By default, the client logger, when it is instructed to send logs, sends the logs to an adapter that the customer must implement. The adapter must be an HTTP adapter that is named `WLClientLogReceiver`, and have at least one procedure. The procedure must be named `log`. The `log` procedure is passed two parameters: `deviceInfo` (a JSON object) and `logMessages` (a JSON array). For more information about implementing adapter procedures, see "Implementing adapter procedures" on page 115.

The following example shows an implementation of the `log` procedure in the `WLClientLogReceiver-impl.js` file:

```
function log(deviceInfo, logMessages) {

  /* The adapter can choose to process the parameters,
     for example to forward them to a backend server for
     safekeeping and further analysis.

     The deviceInfo object may look like this:
     {
       "appName":       "wlapp",
       "appVersion":    "1.0",
       "deviceId":      "66eed0c9-ecf7-355f-914a-3cedac70ebcc",
       "model":         "Galaxy Nexus - 4.2.2 - API 17 - 720x1280",
       "systemName":    "Android",
       "systemVersion": "4.2.2",
       "os.arch":       "i686",           // Android only
```

Experimental IBM Worklight offline user documentation
The document is provided "as is".
In case of issues refer to the online user documentation.

```
       "os.version":    "3.4.0-qemu+"     // Android only
      }
      The logMessages parameter is a JSON array
      that contains JSON object elements, and might look like this:

        [{
          "timestamp"    : "17-02-2013 13:54:23:745",  // "dd-MM-yyyy hh:mm:ss:S"
          "level"        : "ERROR",                    // ERROR || WARN || INFO || LOG || DEBUG
          "package"      : "your_tag",                 // typically a class name, app name, or JavaScript object name
          "msg"          : "the message",              // a helpful log message
          "threadid"     : 42,                         // (Android only) id of the current thread
          "metadata"     : { "$src" : "js" }           // additional metadata placed on the log call
        }]

  */

  return true;

}
```

The procedure element in the `WLClientLogReceiver.xml` file for `log`:

```
<procedure name="log" securityTest="wl_unprotected" audit="true" />
```

The security test must be `wl_unprotected` because apps in the field might be uploading data before the application successfully authenticated. This scenario might occur in the case of crashes during the first time that the app starts.

The `audit="true"` flag means that uploaded parameters are recorded in the server log. It is a convenient way to get uploaded client logs without having to implement anything in the adapter. You can call `WL.Server.log` manually in the adapter log procedure implementation.

Client-side logging in client apps

You can take advantage of the client-side logging feature in your client apps.

Log capture

Log capture is enabled by default, but can be turned on or off with native or JavaScript API calls.

Logger native options can be controlled statically by the `initOptions.js` file in a IBM Worklight generated app. Customers can inspect these values to ensure that they are set as wanted. The processing of `initOptions` changes the Logger native options. You can affect the log capture configuration programmatically by using the `WL.Logger` API or statically by specifying the options in the `initOptions.js` file, but not both at the same time.

Client logs are always captured, but not sent, for first-time start of every IBM Worklight application. To change this behavior, you can specifically place an API call to disable the capture in Android or IOS native code.

Android
The API call to affect the capture setting is:

```
Logger.setCapture(true);
```

iOS
The API call to affect the capture setting is:

```
[OCLogger setCapture: YES]
```

JavaScript
The API call to affect the capture setting is:

```
WL.Logger.setNativeOptions({'capture': true});
```

Experimental IBM Worklight offline user documentation.
This document is provided "as-is".
In case of issues, refer to the online user documentation.

Uncaught exception capture

An uncaught exception that is permitted to pass all the way out of an application at run time appears to the user as an application crash.

Uncaught exceptions on Android and iOS are recorded when log capture is turned on. This data is recorded to the same persistent buffer as all other normal log calls. As well all other persistently captured log data, it is only sent to the server on demand. You can place an API call early in your application lifecycle to send the data to the server before the same exception occurs during the next user attempt to start the application.

Android

```
// placed as the first line in the main Activity's onCreate method
Logger.sendIfUnCaughtExceptionDetected(this);
```

iOS

```
// placed as first line in
// - (BOOL)application:(UIApplication *)application
// didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
[OCLogger sendIfUnCaughtExceptionDetected];
```

The `sendIfUnCaughtExceptionDetected` method is an optimization, and behaves the same as the `send` method. The difference is that the `sendIfUnCaughtExceptionDetected` method does what the name implies; sends only if the persistent log capture buffer contains an uncaught exception entry.

Sending captured logs to the server

You must create and deploy an adapter with a specific name and procedure as part of your application to receive uploaded logs at the IBM Worklight Server. For more information about these requirements, see "Server preparation for uploaded log data" on page 159. Captured logs are not automatically sent to the server.

Android

You can add code to the main Activity's `onCreate` method (and any other lifecycle methods):

```
// send log to server if anything was captured
Logger.send();
```

For more information about the `Logger.send` API, see "Java client-side API for native Android apps" on page 271.

iOS You can add code to the application lifecycle events in the Application Delegate to call:

```
OCLogger.send();
```

For more information about the `OCLogger.send` API, see "Objective-C client-side API for native iOS apps" on page 271.

JavaScript

You can call:

```
// must wait until the Cordova deviceReady event fires,
// or in wlCommonInit, which is the equivalent
WL.Logger.send();
```

For more information about the `WL.Logger.send` API, see "WL.Logger.send" on page 243.

Experimental IBM Worklight documentation. This document is provided "as-is". In case of issues, refer to the online user documentation.

Polling an adapter or other endpoint to affect logger configuration

You can write client-side code to poll a customer-written adapter. The adapter can reply with an appropriate response, the client must parse the response, and call the appropriate `WL.Logger`, `OCLogger`, or `Logger` API to affect the wanted configuration change.

For example, an adapter implementation might have the following procedure:

```
config() {
  return {capture: true};
}
```

The client code invokes the procedure by using the standard IBM Worklight `invokeProcedure` function as follows:

```
WL.Client.invokeProcedure({
  adapter: 'WLClientLogReceiver',
  procedure: 'config',
  parameters: []
}, {
  onSuccess: function() {
    WL.Logger.setNativeOptions({capture: res.invocationResult.capture}).then(WL.Logger.send);
  }
});
```

You can conditionally return options from the adapter config procedure that is based on some client metadata. To do so, send the metadata through the parameters of the `invokeProcedure` call as follows:

```
environment : WL.Client.getAppProperty(WL.AppProp.ENVIRONMENT)
appName : WL.Client.getAppProperty(WL.AppProp.APP_DISPLAY_NAME)
appVersion : WL.Client.getAppProperty(WL.AppProp.APP_VERSION)
```

Process the incoming data in the adapter config procedure as follows:

```
config(metadata) {
  // turn capture on for Android clients only
  if (metadata.environmen == "android") {
    return {capture: true};
  }
  return {capture: false};
}
```

Experimental IBM Worklight online user documentation.
This document is provided "as is".
In case of issues, refer to the online user documentation.

Chapter 3. Integration with other IBM Mobile Foundation products

To integrate IBM Worklight with other IBM Mobile Foundation products you implement adapters and authentication features.

This topic is intended for developers and administrators who want to understand the various integration options available as part of the larger IBM Worklight V5.0 offering, specifically concerning IBM Endpoint Manager for Mobile Devices, IBM WebSphere Cast Iron, IBM WebSphere DataPower®, and IBM Security Access Manager (ISAM).

Introducing the IBM Worklight platform

Understanding the integration options available as part of the larger IBM Worklight® V5.0 offering. Intended primarily for developers and administrators.

The IBM Worklight platform provides extensible connectivity options to external resources by using the adapter technology available in IBM Worklight. The IBM Worklight platform also provides a flexible authentication framework to support existing security requirements through the authenticator or login modules.

Figure 27 on page 164 gives a high-level view of the topology context for an app on a device that connects to IBM Worklight. It also shows how IBM Worklight uses the adapter model to connect to existing back ends and other Internet or intranet sources. IBM Mobile Foundation Enterprise Edition is the specific offering which provides this capability, and consists of IBM Worklight bundled with IBM Endpoint Manager for Mobile Devices and IBM WebSphere Cast Iron. In addition, there are other IBM products that provide integration options for enterprise connectivity and enterprise security, such as IBM WebSphere DataPower and IBM Security Access Manager (ISAM).

Item	**Description**
A	App
D	Device
N	Network
I/i	Internet or intranet
WL	IBM Worklight
EBE	Existing back ends
I	Other Internet sources

Experimental IBM Worklight online user documentation.
This document is provided "as is".
In case of issues, refer to the online user documentation.

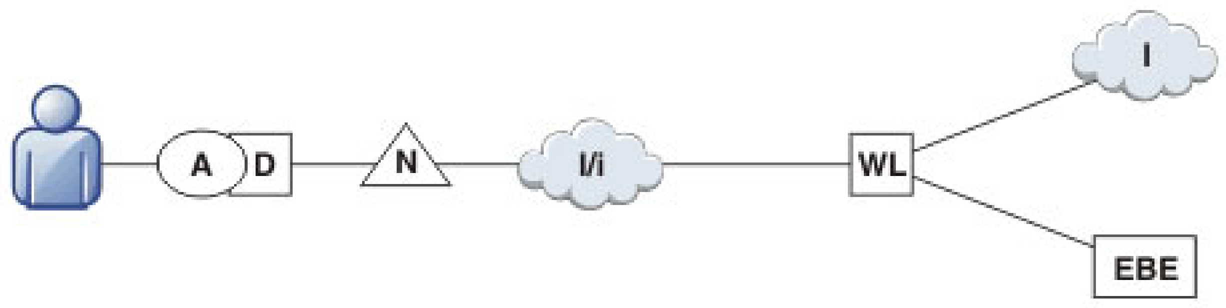


Figure 27. Overall Topology

Figure 28 shows where these products fit within the typical IBM Worklight topology diagram shown in Figure 27.

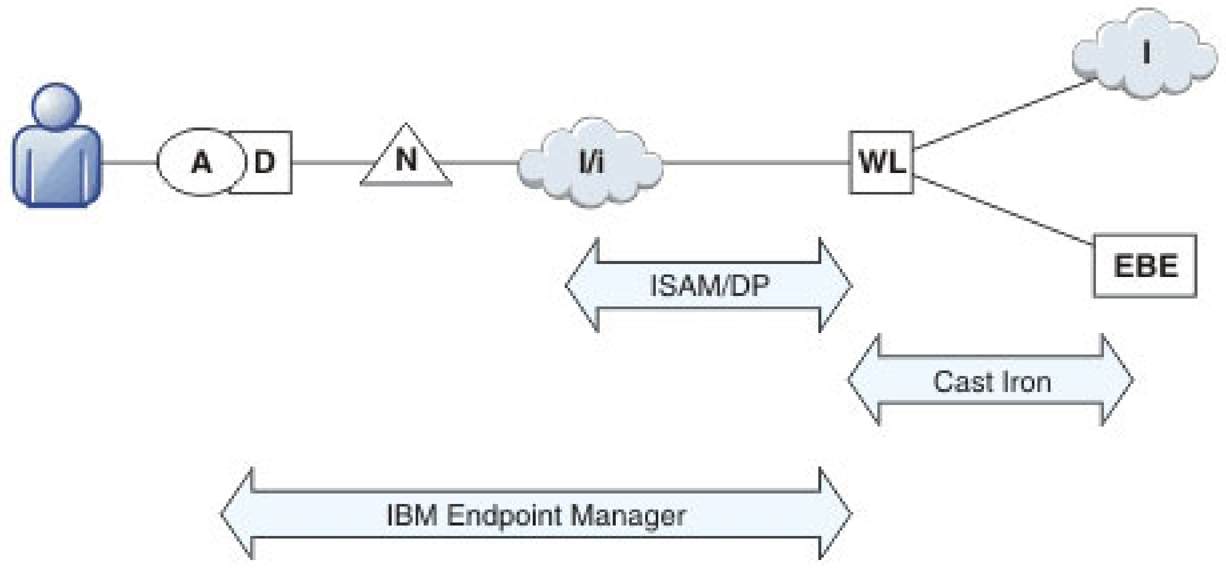


Figure 28. Integration Points

Integration with Cast Iron

An overview of the use of IBM WebSphere Cast Iron to enable enterprise connectivity within an IBM Worklight environment.

There are four adapters supported as part of the IBM Worklight platform:
- SQL
- HTTP
- Cast Iron
- JMS

The Cast Iron adapter provides first-class integration with all of the cloud-based, hardware appliance, or software-based hypervisor editions of IBM WebSphere Cast Iron.

IBM WebSphere Cast Iron enables companies to integrate applications, regardless of whether the applications are located on-premise or in public or private clouds.

WebSphere Cast Iron provides an approach to integrating applications that does not require any programming knowledge. You can build integration flows in WebSphere Cast Iron Studio, which is a graphical development environment that is installed on a personal computer. With Cast Iron Studio, you can create an integration project that contains one or more orchestrations. Each orchestration is built with a number of activities that define the flow of data. You can define the details of an activity from the configuration panes within Cast Iron Studio.

Figure 29 shows how the topology in Figure 1 in *Introducing the IBM Worklight platform* changes to reflect the use of Cast Iron, with the IBM Worklight Cast Iron adapter represented by the thicker line between IBM Worklight and Cast Iron.

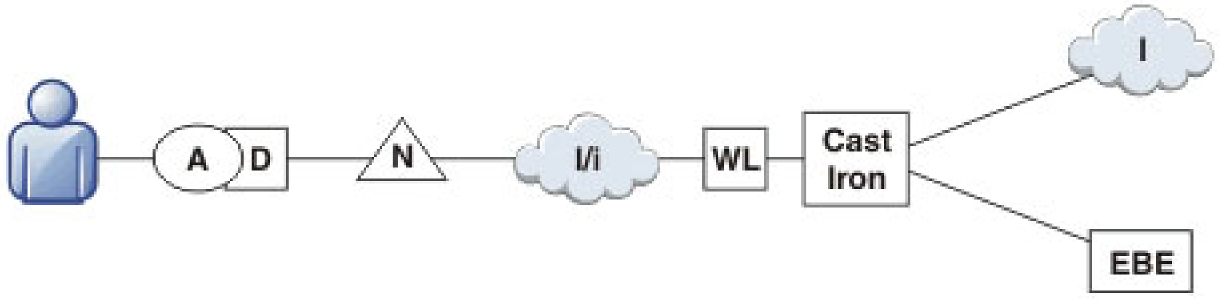


Figure 29. Integration with Cast Iron

For more information about Cast Iron adapters, see the module *Cast Iron adapter - Communicating with Cast Iron*, under category 4, *Worklight server-side development*, in "Getting started tutorials and samples" on page 29.

Integration with reverse proxy

An overview of the use of a reverse proxy to enable enterprise connectivity within an IBM Worklight environment.

Reverse proxies typically front IBM Worklight run times as part of the deployment shown in Figure 30, and follow the gateway pattern.

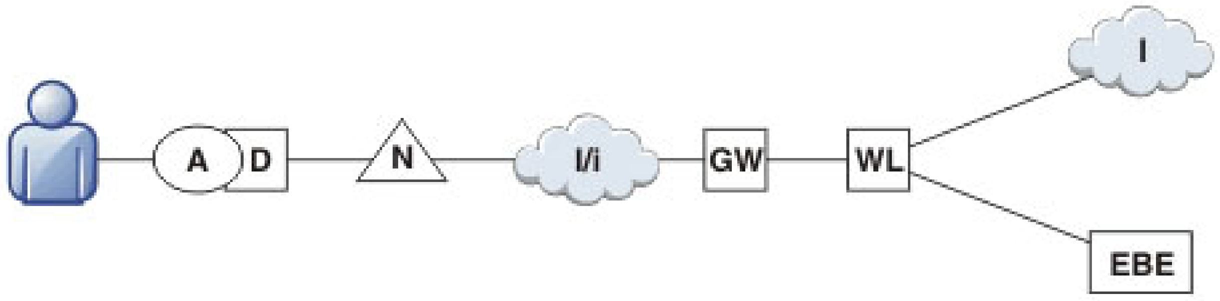


Figure 30. Integration with reverse proxy

The gateway icon (GW) represents a reverse proxy such as WebSphere DataPower, or ISAM. In addition to protecting IBM Worklight resources from the Internet, the reverse proxy provides termination of SSL connections and authentication. The reverse proxy, in effect, can also act as a policy enforcement point (PEP).

When using a gateway, app (A) on device (D) uses the public URI advertised by the gateway instead of the internal IBM Worklight URI. The public URI can be exposed as a setting as part of the app or can be built in during promotion of the app to production before publishing the app to public or private app stores.

Authentication at the gateway

Use of a reverse proxy to provide authentication to IBM Worklight.

If authentication is terminated at the gateway, IBM Worklight can be informed of the authenticated user by a shared context, such as a custom HTTP header or a cookie. By using the extensible authentication framework, IBM Worklight can be configured to use the user identity from one of these mechanisms and establish a successful login. A typical authentication flow is shown in Figure 31.

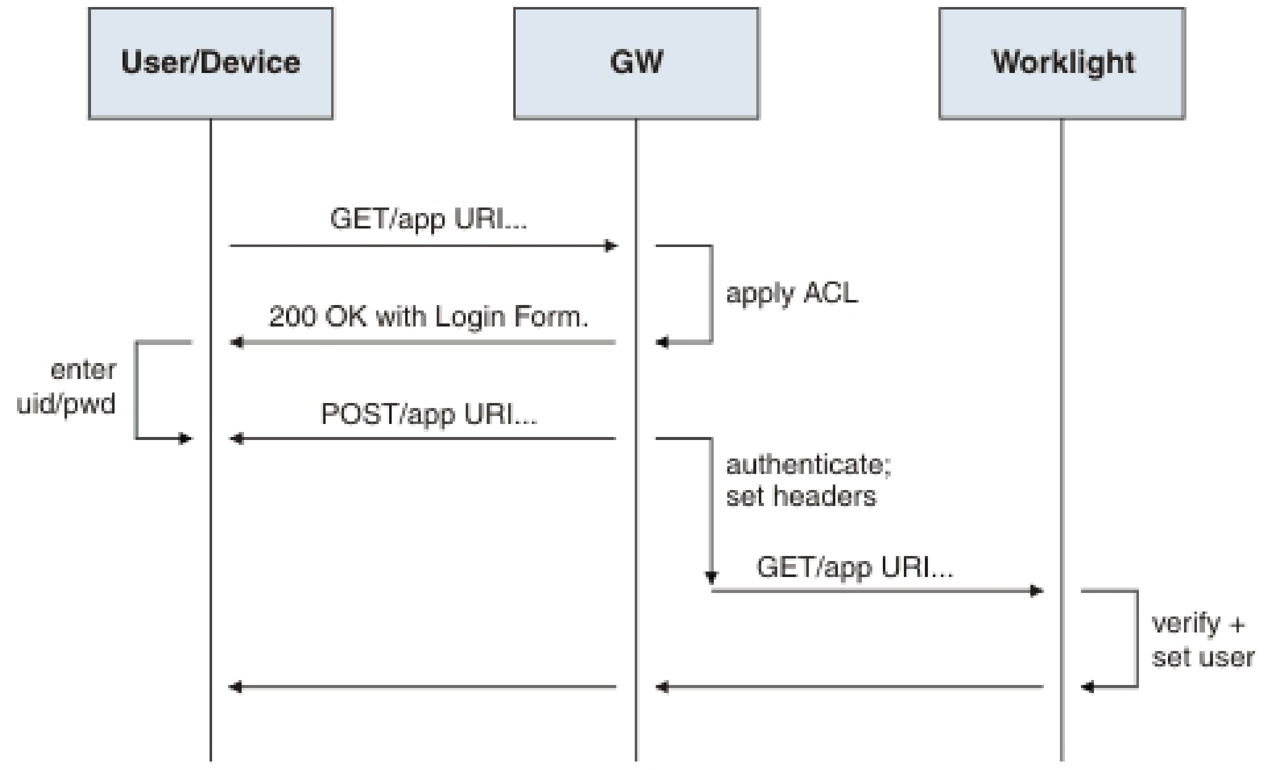


Figure 31. Authentication flow

This configuration was tested with DataPower and ISAM for header-based authentication and LTPA-based authentication.

Header-based authentication

Use of header-based authentication to log in to IBM Worklight through a reverse proxy.

- On successful authentication, the gateway forwards a custom HTTP header with the user name or ID to IBM Worklight.
- IBM Worklight is configured to use `HeaderAuthenticator` and `HeaderLoginModule` on either Tomcat or WebSphere Application Server

LTPA-based authentication

Use of LTPA-based authentication to log in to IBM Worklight through a reverse proxy.

- On successful authentication, the gateway forwards an LTPA token (in the form of an HTTP cookie) to IBM Worklight
- IBM Worklight on WebSphere Application Server is configured to use `WebSphereFormBasedAuthenticator` and `WebSphereLoginModule`.

Managing end points with IBM Endpoint Manager

An overview of the use of IBM Endpoint Manager to manage devices within an IBM Worklight environment.

IBM Worklight provides app management capabilities as part of the platform. IBM Endpoint Manager provides specific device management capabilities. The app can also use certain device functions which leads to an overlap in some of the management aspects between IBM Worklight and IBM Endpoint Manager, as shown in Figure 32.

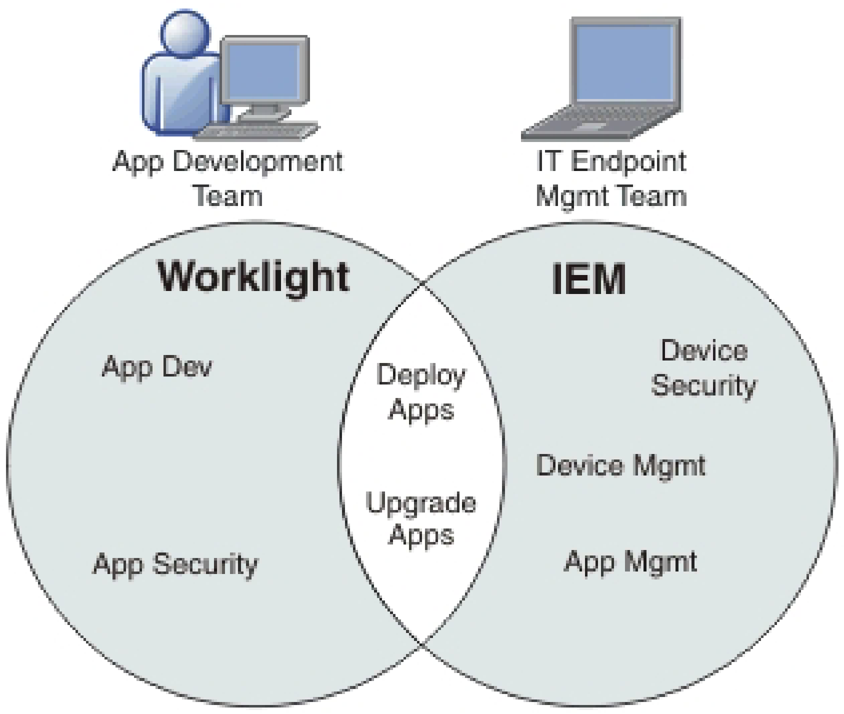


Figure 32. IBM Worklight and IBM Endpoint Manager management capabilities

For devices that must be managed as enterprise assets and devices that must be controlled across applications, IBM Endpoint Manager provides the following mobile device management capabilities:

- Safeguard of enterprise data
- Flexible management
- Maintained compliance
- Unified infrastructure

Safeguard of Enterprise Data

- Selectively wipes enterprise data when devices are lost or stolen.
- Configures and enforces passcode policies, encryption, VPN, and more.

Flexible Management

- Secures and manages employee-owned and corporate-owned mobile devices by using a combination of email-based and agent-based management, while preserving the native device experience.

Maintained Compliance

- Automatically identifies non-compliant devices.
- Denies email access or issues user notifications until corrective actions are implemented.

Unified Infrastructure

- Uses a single infrastructure to manage and secure all of your enterprise devices; that is, smartphones, media tablets, desktops, notebooks, and servers.

Useful links

Other resources on integration with IBM WebSphere Cast Iron, IBM Endpoint Manager, IBM WebSphere DataPower, and IBM Security Access Manager are available from the product websites and IBM Redbooks® website.

For more information, use the following links:

IBM WebSphere Cast Iron

http://www.redbooks.ibm.com/redpapers/pdfs/redp4840.pdf

http://www.redbooks.ibm.com/abstracts/sg248004.html?Open

IBM Endpoint Manager

http://www-01.ibm.com/software/tivoli/solutions/endpoint/mdm/

IBM WebSphere DataPower

http://www.redbooks.ibm.com/abstracts/redp4790.html?Open

http://www.redbooks.ibm.com/abstracts/sg247620.html?Open

IBM Security Access Manager

http://www.redbooks.ibm.com/abstracts/redp4621.html?Open

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Chapter 4. Migrating from the WebSphere Application Server Feature Pack

To migrate from the WebSphere Application Server Feature Pack, select an appropriate scenario and follow its procedures.

About this task

This topic is intended for developers who want to migrate applications developed with the *Feature Pack for Web 2.0 and Mobile* to IBM Worklight.

You can migrate applications that use the client programming model, the server programming model, JAX-RS, JSON-RPC, or proxies.

Procedure

1. Select the scenario that your application uses.
2. Follow the steps.
3. Refer to the Dojo Showcase example for support.

Migration scenarios

This information is intended for developers who want to migrate applications developed with the WebSphere Application Server Feature Pack for Web 2.0 and Mobile to IBM Worklight.

A mobile web application created with the *Feature Pack for Web 2.0 and Mobile* uses open standards such as HTML, CSS, and JavaScript. It might connect to SOA-based services by using JAX-RS and JSON-RPC.

You can use IBM Worklight to package these mobile web applications as native apps and make them available in an application store. In its simplest form, this migration consists of repackaging the apps within IBM Worklight. You can also use device APIs (through Apache Cordova) and IBM Worklight client APIs.

Migrating an application that uses the client programming model

Migrate a mobile app by using the IBM Worklight client programming model, where the mobile web application is repackaged as a mobile hybrid application.

About this task

IBM Worklight assumes that the application is packaged with HTML, JavaScript, or CSS and that it can be updated in static form to the native shell, by using direct update features. To migrate the app, complete the steps in the Procedure section.

These steps describe a minimal migration. After migration, you can package the app and deploy it to an app store.

Experimental IBM Worklight offline user documentation
This content is provided "as is".
In case of issues, refer to the online user documentation.

To maximize the reuse of services and user interface code, you can refactor the code to use Environment and Skin support. Keep the basic code in the `common` directory and create overrides for each environment. Use dojo/has feature detection for skin-specific behaviors.

Finally, you can extend your mobile app to use advanced features of IBM Worklight. For example, you can use Cordova to control device features such as cameras, and you can use IBM Worklight client APIs to control security.

Procedure

1. Create an IBM Worklight application, selecting the appropriate target environments. A `common` directory and a directory for each environment are generated.
2. Optional. Because devices vary in the features or functions they have, you can use IBM Worklight *application skins* to provide a finer distinction than environments. The IBM Worklight application skin is a user interface variant of an application that can be applied during run time based on runtime device properties. These properties include operating system version, screen resolution, and form factor. For example, within the Android environment folder, you might create a subfolder for Android 4.0 to take advantage of features only available in Android 4.0.
3. Migrate the project structure to the IBM Worklight Environment Model:
   a. Copy common web resources to the common directory.
   b. Continue to use "has" feature detection (dojo/has).
   c. Continue to use the deviceTheme feature (dojox/mobile/theme) for default Dojo mobile themes.

Migrating an application that uses the server programming model

Migrate a mobile app by using the IBM Worklight server programming model, which shows how to extend apps to use IBM Worklight server-side facilities.

About this task

The server programming model is an alternative model to the client programming model. Applications use the server programming model if they use server-side generated web content, such as JSPs and JSF for rendering HTML. Compared to natively packaged apps, remote loading of resources reduces network performance. Complete the following steps to migrate the app:

Procedure

1. Create a project structure according to the IBM Worklight Environment Model. This step is required for creating a native shell for each mobile platform. The shell is a simple Cordova instance that loads a remote resource from the application server.
2. Continue to use the deviceTheme feature (`dojox/mobile/theme`) for default Dojo mobile themes.
3. Use "has" feature detection (`dojo/has`) for device operating-system-specific behaviors.
4. Determine dependencies for your mobile application:
   a. Create a custom Dojo layer for core Dojo and Dojox mobile libraries.
   b. Create a custom Dojo layer for common application Dojo libraries.

Experimental IBM Worklight offline documentation. This document is provided as is. In case of issues refer to the online user documentation.

c. Create a custom Dojo layer for any platform-specific Dojo libraries.

Considerations for applications that use JAX-RS, JSON-RPC, or proxying

Mobile web applications that connect to SOA-based services by using JAX-RS, JSON-RPC, or through a proxy, might have additional steps when being migrated from the WebSphere Application Server Feature Pack for Web 2.0 and Mobile to IBM Worklight.

Migrating an application that uses JAX-RS

If your application contains services that were written using JAX-RS hosted on WebSphere Application Server, consider the following points:

- No change is required to continue to use the WebSphere Security Model. You can use existing REST services from the app.
- To integrate security with IBM Worklight, you must proxy existing REST services that use JSON-RPC through an HTTP adapter.
- Services are hosted in a separate EAR or WAR file from the IBM Worklight Application. However, there might be restrictions on host name and port because the services and application are in the same sandbox domain.

Migrating an application that uses JSON-RPC

If your application contains services that were written using JSON-RPC hosted on WebSphere Application Server, consider the following points:

- No change is required to continue to use the WebSphere Security Model. You can use existing RPC services from the app.
- To integrate security with IBM Worklight, you must proxy existing RPC services that use JSON-RPC through an HTTP adapter.
- Services are hosted in a separate EAR or WAR file from the IBM Worklight Application. However, there might be restrictions on host name and port because the services and application are in the same sandbox domain.

Migrating an application that uses proxying

If your application contains external services that require an Ajax Proxy on WebSphere Application Server, consider the following points:

- No change is required to continue to use the WebSphere Security Model. You can use the existing Ajax Proxy from the app.
- To integrate security with IBM Worklight you must proxy existing HTTP requests that use JSON-RPC through an HTTP adapter
- Services are hosted externally of the IBM Worklight Application. However, you can use the Ajax Proxy for advanced features.

Example: Migrating the Dojo showcase sample

Demonstrate the steps required to migrate the Dojo showcase sample from the WebSphere Application Server Feature Pack for Web 2.0 and Mobile to IBM Worklight®.

Experimental IBM Worklight offline documentation. This document is provided "as-is". In case of issues refer to the online user documentation.

About this task

The application to be migrated is the Dojo showcase which is a mobile web application that demonstrates the capabilities of the Dojo toolkit. You can run the demonstration at http://demos.dojotoolkit.org/demos/mobileGallery/demo-iphone.html

To migrate the application, complete the following steps:

Procedure

1. Create an IBM Worklight project and create an IBM Worklight application in the project.
2. Copy all the resources for the web application to the `common` directory of the IBM Worklight application.
3. The Dojo showcase application contains only static html pages. If you have remote dynamic server pages, you can either use the JavaScript templating library or use web view. The JavaScript templating library renders the pages locally on devices. Web view loads the remote server pages.
4. Change the <mainfile> element in the `application-descriptor.xml` file in the IBM Worklight application. Make sure the content of `<mainfile>` element points to the correct startup html page.
5. If you have startup JavaScript logic that initializes the web application when the browser loads it, move the startup logic to the **`wlCommonInit`** method of the `.js` file in the `common/js` directory. IBM Worklight initializes its own library and runtime environment during startup and the application startup logic follows the IBM Worklight initialization process.
6. To keep the application as small as possible, do not add any other IBM Worklight skin to the application. Adding a skin results in the duplication of all the web application resources in the final package built.
7. Minimize the required Dojo modules into one `.js` file so that the file size of the application is minimal. IBM Worklight does not provide any javascript shrinking in the build process.

Experimental IBM Worklight documentation. This document is not supported as IBM documentation. In case of issues, refer to the online user documentation.

Chapter 5. API reference

To develop your native or hybrid applications, refer to the classes and methods of the JavaScript, Objective-C, Java, and Java Micro Edition APIs.

Purpose

To enable you to develop IBM Worklight applications in JavaScript, Java Micro Edition, Java for Android, and Objective-C for iOS.

IBM Worklight client-side API

This collection of topics contains a description of the application programming interface (API) for use in writing client applications with IBM Worklight.

JavaScript client-side API

You can use JavaScript API to develop apps for all environments.

`WLClient` JavaScript client library

This collection of topics lists the public methods of the IBM Worklight runtime client API for mobile apps, desktop, and web.

`WLClient` is a JavaScript client library that provides access to IBM Worklight capabilities. You can use `WLClient` to perform the following types of functions:

- Initialize and reload the application
- Manage authenticated sessions
- Obtain general application information
- Retrieve and update data from corporate information systems
- Store and retrieve user preferences across sessions
- Internationalize application texts
- Specify environment-specific user interface behavior
- Store custom log lines for auditing and reporting purposes in special database tables
- Write debug lines to a logger window
- Use functions specific to iPhone, iPad, Android, BlackBerry 6 and 7, Windows Phone 7.5, and Windows Phone 8 devices

Note: Although JavaScript does not support encapsulation, do not use any method or member not listed in this document. Their semantics or existence is not guaranteed in future versions of the IBM Worklight Client API.

Calls to the Worklight Server:

`WLClient` uses asynchronous JavaScript calls, which accept an **`options`** parameter. Success and failure handlers receive a **`response`** parameter. The API consists of many calls, listed here.

`WLClient` uses asynchronous JavaScript calls to access the IBM Worklight Server. Each asynchronous method accepts an **`options`** parameter, which includes success and failure handlers to communicate the results of the call. If you want to be

Experimental IBM Worklight online user documentation.
This document is provided "as is".
In case of issues, refer to the online user documentation.

notified when an asynchronous function returns, you must supply these callback functions within the **options** parameter when you call the function.

Augmented Options

The **options** parameter often contains additional properties applicable to the specific method that is being called. These additional properties are detailed in the description of the specific method.

Augmented Response

The success and failure handlers of all asynchronous calls always receive a **response** parameter that contains a common set of properties. Some calls pass additional properties in the `response` object of the success and failure handlers. In such cases, these additional properties are detailed in the description of the specific method.

Quick Reference

The Worklight Client API consists of the following API methods:

- General application methods
  - Lifecycle: “WL.Client.init” on page 189, “WL.Client.reloadApp” on page 200
  - Connectivity: “WL.Client.setHeartBeatInterval” on page 201, “WL.Client.connect” on page 182, Connectivity-related JavaScript Events
  - Session management methods: “WL.Client.getUserName” on page 187, “WL.Client.getLoginName” on page 186, “WL.Client.login” on page 197, “WL.Client.logout” on page 198, “WL.Client.isUserAuthenticated” on page 196, “WL.Client.getUserInfo” on page 187, “WL.Client.updateUserInfo” on page 202
  - Data access methods: “WL.Client.invokeProcedure” on page 194
  - Activity logging methods: “WL.Client.logActivity” on page 197
  - User preference methods: “WL.Client.setUserPref” on page 201, “WL.Client.getUserPref(key)” on page 188
  - Application properties methods: “WL.Client.getEnvironment” on page 186, “WL.Client.getAppProperty” on page 184
  - Error handling: “WL.App.getErrorMessage” on page 178
  - Debugging: “The WL.Logger object” on page 241
- Mobile functionality and UI
  - Push notification API: “WL.Client.Push.isPushSupported” on page 244, “WL.Client.Push.isPushSMSSupported” on page 244, “WL.Client.Push.onReadyToSubscribe” on page 245, “WL.Client.Push.registerEventSourceCallback” on page 245, “WL.Client.Push.subscribe” on page 246, “WL.Client.Push.subscribeSMS” on page 247, “WL.Client.Push.unsubscribe” on page 248, “WL.Client.Push.unsubscribeSMS” on page 249
  - Network details: “WL.Device.getNetworkInfo” on page 203
  - Opening a URL: “WL.App.openURL” on page 178
  - Options menu: WL.OptionsMenu
  - Tab bar: Tab Bar API
  - Badge: “WL.Badge.setNumber” on page 181
  - Toast: “WL.Toast.show” on page 206

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

- Globalization: “WL.App.getDeviceLocale” on page 178, “WL.App.getDeviceLanguage” on page 178
  - Back button: “WL.App.overrideBackButton” on page 180, “WL.App.resetBackButton” on page 181
  - Dialog box: “WL.SimpleDialog” on page 258
  - Busy indicator: “WL.BusyIndicator (constructor)” on page 207
  - Closing an app: “WL.App.close” on page 177
  - Accessing native pages on mobile apps: “WL.NativePage.show” on page 205
  - Switching between HTML Pages: “WL.Fragment.load” on page 261,“Class WL.Page” on page 262
  - Encrypted offline cache: WL.EncryptedCache
  - Clipboard: “WL.App.copyToClipboard” on page 203
- Web and desktop widget methods
  - Desktop window state: “WL.Client.onDock, WL.Client.onUndock” on page 199, “WL.Client.onShow, WL.Client.onHide” on page 199, “WL.Client.close” on page 182, “WL.Client.minimize” on page 198
  - Globalization: “WL.Client.getLanguage” on page 213
- Mechanisms used by the WLClient methods
  - “The `options` object” on page 249, Timeout

WL.App.BackgroundHandler.setOnAppEnteringBackground

Define the behavior of the application before it enters the background.

Syntax

```
WL.App.BackgroundHandler.setOnAppEnteringBackground (handler)
```

Description

Applies for iOS 4 and above.

Defines the behavior of the application just before iOS takes a screen capture of it before moving it to the background.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Parameters

Table 25. WL.App.BackgroundHandler.setOnAppEnteringBackground parameters

Parameter	Description
`handler`	Function. The function that is called upon receiving the event from iOS that the application is about to enter background. Values: **WL.App.BackgroundHandler.defaultIOSBehavior** Use the default behavior of iOS (which is equivalent to not doing anything). **WL.App.BackgroundHandler.hideView** Display a black screen instead of the browser component. To ensure that the view is hidden also when the application returns to the foreground, Choose between one of the following options: • Do not call `setOnAppEnteringForeground`: this option automatically shows the application after it returns from the background. • Call `setOnAppEnteringForeground`, but customize what the application does when it returns from the background. Return `WL.App.BackgroundHandler.hideViewToForeground()` at the end of the function. For an example, see "WL.App.BackgroundHandler.setOnAppEnteringFor on page 177. **WL.App.BackgroundHandler.showSplashScreen** Show a splash screen instead of the application **WL.App.BackgroundHandler.hideElements** Hide all HTML elements that have the style WLHideOnEnteringBackground **Custom function**

Examples

Example: Use hideElements

```
// CSS
<span id="moneyInTheBank" class="WLHideOnEnteringBackground"> ... </span>
// JavaScript
WL.App.BackgroundHandler.setOnAppEnteringBackground(
WL.App.BackgroundHandler.hideElements);
```

Example: Use custom function

Experimental IBM Worklight offline user documentation.
This document is provided as is
In case of issues, refer to the online user documentation.

```
// JavaScript
WL.App.BackgroundHandler.setOnAppEnteringBackground(myFunc);
```

WL.App.BackgroundHandler.setOnAppEnteringForeground

Define the behavior of the application before it enters the foreground

Syntax

```
WL.App.BackgroundHandler.setOnAppEnteringForeground (handler)
```

Description

Applies for iOS 4 and later.

Defines the behavior of the application just before it enters the foreground.

Parameters

Table 26. WL.App.BackgroundHandler.setOnAppEnteringForeground parameters

Parameter	Description
`handler`	Mandatory. Function. The function that is called upon receiving the event from iOS that the application is about to enter foreground.

Example

```
WL.App.BackgroundHandler.setOnAppEnteringForeground(myFunc);
```

Example: Customizing what the application does when it returns from the background to ensure that the view is hidden. (See also "WL.App.BackgroundHandler.setOnAppEnteringBackground" on page 175.)

```
WL.App.BackgroundHandler.setOnAppEnteringForeground( function () {
  alert("This is my custom code");
  return WL.App.BackgroundHandler.hideViewToForeground();
});
```

WL.App.close

Quits the application.

Note: Note: According to iOS Human Interface Guidelines, an iOS app must not contain code that exits the app. The device's **Home** button is used for this purpose instead. Therefore `WL.App.close()` API has no effect in iOS applications (tapping a button that is implemented with this API has no effect).

Syntax

```
WL.App.close();
```

Parameters

None.

Return value

None.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

WL.App.getDeviceLanguage

Returns the language code.

Syntax

```
WL.App.getDeviceLanguage()
```

Description

Returns the language code according to user device settings, for example: `en`.

Parameters

None.

WL.App.getDeviceLocale

Returns the locale code (or device language on BlackBerry)

Syntax

```
WL.App.getDeviceLocale()
```

Description

Returns the locale code according to user device settings, for example: `en_US`.

Note: On BlackBerry 6 and 7, this method returns the device language (for example, `en`), not the device locale.

Parameters

None.

WL.App.getErrorMessage

Extracts a string that contains an error message.

Syntax

```
WL.App.getErrorMessage(exception)
```

Description

Extracts a string that contains the error message within the specified exception object. Use for exceptions that are thrown by the IBM Worklight client runtime framework.

Parameters

Table 27. WL.App.getErrorMessage parameters

Parameter	Description
`exception`	Mandatory. The exception object from which the error string is extracted.

WL.App.openURL

Open a URL. The behavior depends on the application platform.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Syntax

```
WL.App.openURL(url, target, options)
```

Description

Opens the specified URL according to the specified target and options (specs). The behavior of this method depends on the application environment, as follows:

Table 28. WL.App.openURL application environments and behavior

Environment	Description
Adobe AIR	Opens a new browser window at the specified URL. The **target** and **options** parameters are ignored.
Android	Replaces the application with a new default browser window at the specified URL. The **target** and **options** parameters are ignored. The application is not closed; pressing Back on the phone brings the user back to the application.
Apple OSX dashboard	Opens a new browser window at the specified URL. The **target** and **options** parameters are ignored.
BlackBerry 6 and 7	Replaces the application with a new default browser window at the specified URL. The **target** and **options** parameters are ignored.
iPhone, iPad	Replaces the application with a new Safari window at the specified URL. The **target** and **options** parameters are ignored.
Mobile web apps	Opens a new browser window at the specified URL. Whether the **target** and **options** parameters are ignored or not depends on the specific mobile browser.
Vista Sidebar	Opens a new browser window at the specified URL. The **target** and **options** parameters are NOT ignored.
Windows Phone 8	Replaces the application with a new Internet Explorer window at the specified URL. The **target** and **options** parameters are ignored.
Windows Phone 7.5	Replaces the application with a new Internet Explorer window at the specified URL. The **target** and **options** parameters are ignored.
Windows 8	Replaces the application with a new Internet Explorer window at the specified URL. The **target** and **options** parameters are ignored.
Other environments	If the value of the **target** parameter is `_self` or unspecified, replaces the application iframe with the specified URL. Otherwise, opens a new browser window with the specified URL. The **target** and **options** parameters are NOT ignored.

Experimental IBM Worklight offline user documentation.
This document is provided as is.
In case of issues, refer to the online documentation.

Parameters

Table 29. WL.App.openURL parameters

Parameter	Description
url	Mandatory. The URL of the web page to be opened.
target	Optional. The value to be used as the target (or name) parameter of the JavaScript window.open method. If no value is specified, `_self` is used.
options	Optional. The value to be used as the options (or specs) parameter of the JavaScript window.open method. If no value is specified, the following options are used: status=1, toolbar=1, location=1, menubar=1, directories=1, resizable=1, scrollbars=1

Return Value

A reference to the newly opened window, or NULL if no window was opened.

WL.App.overrideBackButton

Overrides the default behavior of the Back button on Android, Windows Phone 7.5, and Windows Phone 8.

Note: This method applies to Android, Windows Phone 7.5, and Windows Phone 8 only.

Syntax

```
WL.App.overrideBackButton (callback)
```

Description

Overrides the default behavior of the Back button on Android, Windows Phone 7.5, and Windows Phone 8 devices, calling the callback function whenever **Back** is pressed.

Parameters

Table 30. WL.App.overrideBackButton parameters

Parameter	Description
callback	Mandatory. Function. The function that is called when **Back** is pressed.

Return Value

None

```
WL.App.overrideBackButton(backFunc);
function backFunc(){
alert('you hit the back key!');
}
```

Experimental IBM Worklight offline user documentation. This document is provided "as-is". In case of issues, refer to the online user documentation.

WL.App.resetBackButton

Resets the original Back button behavior.

Note: This method applies to Android, Windows Phone 7.5, and Windows Phone 8 only.

Syntax

```
WL.App.resetBackButton()
```

Description

Resets the original Back button behavior after it was changed by the overrideBackButton method.

Parameters

None

Return Value

None

WL.Badge.setNumber

Sets the application badge to the number provided.

Note: This object is only applicable to iOS applications.

Syntax

```
WL.Badge.setNumber(number)
```

Description

Sets the application badge to the number provided.

Parameters

Table 31. WL.Badge.setNumber parameters

Parameter	Description
number	Mandatory. Integer. An integer that is displayed as a badge over the application icon. A value of 0 or below removes the application badge. Values which are too long to be displayed entirely (5 or more digits in an iPhone device) are truncated with ellipsis.

Return Value

None.

WL.Client.addGlobalHeader

This method adds an HTTP header to be used in server requests issued by an IBM Worklight framework

Syntax

```
WL.Client.addGlobalHeader(headerName, headerValue)
```

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Description

Adds an HTTP header to be used in server requests issued by an IBM Worklight framework.

The HTTP header is used in all requests until removed by the `WL.Client.removeGlobalHeader` API call.

Parameters

Parameter	Description
headerName	Mandatory. The name of the header to be added..
headerValue	Mandatory. The value of the header to be added.

Return Value

None.

Example

```
WL.Client.addGlobalHeader("MyCustomHeader","abcdefgh");
```

WL.Client.close

Close a widget on Adobe AIR.

Syntax

```
WL.Client.close()
```

Description

Note: This method is only applicable to widgets that are running on Adobe AIR.

Closes the AIR widget (making it exit).

Parameters

None.

WL.Client.connect

This method establishes a connection to the Worklight Server.

Syntax

```
WL.Client.connect(options)
```

Description

The connect() method tries to establish a connection to the Worklight Server. You must call this method before calling any other WL.Client method that communicates with the Worklight Server.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Parameters

Parameter	Description
options	Optional. A JSON block with the following additional properties: **onSuccess** A callback function invoked when the connection to the IBM Worklight Server is established **onFailure** A callback function invoked when the `WL.Client.connect` method fails to establish connection with the IBM Worklight server. The callback receives one parameter of type `WL.FailResponse`, which might be null if `connect` is called while a previous call to connect has not yet returned. **timeout** Integer. Number of milliseconds to wait for the server response before failing with a request timeout.

Connectivity-related JavaScript Events

The IBM Worklight runtime framework fires two events, to which you can listen to capture changes in connectivity. The events are fired only on change of connectivity state.

- `WL.Events.WORKLIGHT_IS_CONNECTED`: fired when the application connects to the Worklight Server
- `WL.Events.WORKLIGHT_IS_DISCONNECTED`: fired when loss of connectivity to Worklight Server is detected

```
document.addEventListener(WL.Events.WORKLIGHT_IS_CONNECTED , handleConnectionUp, false);
document.addEventListener(WL.Events.WORKLIGHT_IS_DISCONNECTED, handleConnectionDown, false);
```

WL.Client.deleteUserPref

Delete a user preference key.

Syntax

```
WL.Client.deleteUserPref(key, options)
```

Description

An asynchronous function that deletes a specified user preference key.

Note: The local user preferences in the application are updated only when a successful response is received from the server.

Parameters

Table 32. Parameters for user preference method WL.Client.deleteUserPref

Parameter	Description
key	Mandatory. The user preference key. Can be up to 128 characters long.

Experimental IBM Worklight offline user documentation. This document is provided as-is. In case of issues, refer to the online user documentation.

Table 32. Parameters for user preference method WL.Client.deleteUserPref (continued)

Parameter	Description
options	Optional. A standard `options` object.

Return Value

None.

WL.Client.getAppProperty

Returns the value of a property.

Syntax

```
WL.Client.getAppProperty (property)
```

Description

This method returns the value of the specified property.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Parameters

Table 33. Values returned for a specified property

Property	Description
property	Mandatory. One of the following values: **WL.AppProperty.AIR_ICON_16x16_PATH** For AIR widgets only; the relative path to the AIR icon. **WL.AppProperty.AIR_ICON_128x128_PATH** For AIR widgets only; the relative path to the AIR icon. **WL.AppProperty.DOWNLOAD_APP_LINK** For desktop widgets only; the URL for downloading an updated version of the application. **WL.AppProperty.APP_DISPLAY_NAME** The application display name, as defined in the application descriptor. **WL.AppProperty.APP_LOGIN_TYPE** The application login type, as defined in the application descriptor: `never`, `onstartup`, or `ondemand` **WL.AppProperty.APP_VERSION** The application version, as defined in the application descriptor (a newer version might be available on the Worklight Server) **WL.AppProperty.HEIGHT** For web and desktop environments only. **WL.AppProperty.IID** The application instance ID (string). All mobile environments use "0". **WL.AppProperty.LANGUAGE** The application's language locale. **WL.AppProperty.LATEST_VERSION** The latest application version available on the Worklight Server. **WL.AppProperty.LOGIN_DISPLAY_TYPE** The login display type, as defined in the `worklight.properties` file: `popup` or `embedded`. **WL.AppProperty.LOGIN_POPUP_HEIGHT** The login window height, as defined in the `worklight.properties` file; relevant when **LOGIN_DISPLAY_TYPE** is `popup`. **WL.AppProperty.LOGIN_POPUP_WIDTH** The login window width, as defined in the `worklight.properties` file; relevant when **LOGIN_DISPLAY_TYPE** is `popup`. **WL.AppProperty.MAIN_FILE_PATH** For web environments only; the absolute URL to the main application file. **WL.AppProperty.SHOW_IN_TASKBAR** For AIR widgets only; a Boolean stating whether the Air application shows in the taskbar, as defined in the descriptor

Experimental IBM Worklight offline use documentation.
This document is provided "as-is".
In case of issues, refer to the online documentation.

WL.Client.getEnvironment

Identifies the type of environment in which the application is running.

Syntax

`WL.Client.getEnvironment()`

Description

Identifies the type of environment in which the application is running, such as iPhone, Facebook, Windows, or iGoogle.

Parameters

None.

Return Value

A constant that identifies the type of environment. The valid values are defined in the *WL.Environment* variable in the `worklight.js` file, and are as follows:

- `WL.Environment.ADOBE_AIR`
- `WL.Environment.ANDROID`
- `WL.Environment.EMBEDDED`
- `WL.Environment.FACEBOOK`
- `WL.Environment.IGOOGLE`
- `WL.Environment.IPAD`
- `WL.Environment.IPHONE`
- `WL.Environment.MOBILE_WEB`
- `WL.Environment.OSX_DASHBOARD`
- `WL.Environment.PREVIEW` (when the application runs in Preview mode)
- `WL.Environment.VISTA_SIDEBAR`
- `WL.Environment.WINDOWS_PHONE_8`
- `WL.Environment.WINDOWS_PHONE` (Windows Phone 7.5)
- `WL.Environment.WINDOWS8`

When an app is running in Preview mode, this method returns `WL.Environment.PREVIEW`, regardless of the previewed environment. There are two reasons for this behavior:

- Environment-specific code can fail when invoked from the browser (because the environment might support features that are not available in the browser).
- `WL.Client` behaves differently in different environments (for example, cookie management).

A good practice is to rely on the IBM Worklight UI optimization framework and separate environment-dependent JS to separate files rather than using the `WL.Client.getEnvironment()` function.

WL.Client.getLoginName

Returns the login name of the user who is currently logged in.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Syntax

```
WL.Client.getLoginName(realm)
```

Description

Note: This method is applicable only to applications that support login.

This method returns the login name of the user who is logged in. The login name is the name that the user entered when logging in.

Parameters

Table 34. Parameters for session management method WL.Client.getLoginName

Parameter	Description
realm	Optional. The name of a realm that is defined in the `authenticationConfig.xml` file. If specified, the realm must be a Facebook realm. The use of Facebook realms is deprecated in Worklight version 5.0.5. Support might be removed in any future version. If no value is specified, the method returns the login name in the resource realm that is assigned to the application when it was deployed.

Return Value

The login name of the user who is logged in, or `NULL` if the login name is unknown.

WL.Client.getUserInfo

This method returns a user property.

Syntax

```
WL.Client.getUserInfo(realm, key)
```

Description

This method returns a user property with the specified key in the specified authentication realm.

Parameters

Parameter	Description
realm	Mandatory. The name of a realm name defined in the `authenticationConfig.xml` file.
key	Optional. A standard `options` object.

WL.Client.getUserName

This method returns the user name of the user who is currently logged in.

Experimental IBM Worklight offline user documentation
The document is provided as is
In case of issues refer to the online user documentation.

Syntax

```
WL.Client.getUserName(realm)
```

Description

Note: This method is only applicable to applications that support login.

This method returns the user name of the user who is currently logged in, as defined by the login module used to authenticate the user.

Parameters

Parameter	Description
realm	Optional. The name of a realm defined in the `authenticationConfig.xml` file. If specified, the realm must be a Facebook realm. The use of Facebook realms is deprecated in Worklight version 5.0.5. Support might be removed in any future version. If no value is specified, the method returns the user name in the resource realm assigned to the application when it was deployed.

Return Value

The user name of the user who is currently logged in, or `NULL` if the user name is unknown.

WL.Client.getUserPref(key)

Returns the local value of a user preference.

Description

This method returns the local value of a specified user preference.

Parameters

Table 35. Parameters for user preference method WL.Client.getUserPref(key)

Parameter	Description
key	Mandatory. The user preference key.

Return Value

The value of the user preference or `NULL` if there is no user preference with the specified key.

Exceptions

An exception is thrown when invalid parameters are passed to the function.

Experimental IBM Worklight offline user documentation. This document is provided "as-is". In case of issues, refer to the online user documentation.

WL.Client.hasUserPref

Checks whether a user preference is defined locally in the application.

Syntax

```
WL.Client.hasUserPref(key)
```

Description

This method checks whether a specified user preference is defined locally in the application.

Parameters

Table 36. Parameters for user preference method WL.Client.hasUserPref

Parameter	Description
key	Mandatory. The user preference key.

Return Value

Returns `true` if the preference exists, `false` otherwise.

Exceptions

An exception is thrown when invalid parameters are passed to the function.

WL.Client.init

This method initializes the `WL.Client` object.

Syntax

```
WL.Client.init({options})
```

Description

This method initializes the `WL.Client` object. The options of this method reside in the `initOptions.js` file.

Parameters

An optional `options` object, as described in "The `options` object" on page 249, augmented with the following optional properties:

Property	Description
Timeout	An integer value, denoting the timeout in milliseconds. The timeout affects all calls from the app to the Worklight Server. If not specified, a timeout of 30,000 milliseconds (30 seconds) is used.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Property	Description
enableLogger	A Boolean value, indicating whether the `WL.Logger.debug()` outputs data. If set to `true`, `WL.Logger.debug()` outputs data to the respective log (for example, to the Xcode console for iOS, to LogCat for Android, and to the developer console for desktop browsers). If set to `false`, `WL.Logger.debug()` does not output data. The default value, unless specified otherwise, is `true`. **Note:** The logger is for development purposes only. Log lines can be added to the logger by using the `WL.Logger` object, as described in "The WL.Logger object" on page 241.
messages	A dictionary object for localizing texts, in the `messages.js` file. If not specified, the default object `Messages` (in the same file) is used.
authenticator	An object that implements the Authenticator API. If not specified, `Authenticator` is used.
heartBeatIntervalInSecs	An integer value that denotes the interval in seconds between heartbeat messages that are automatically sent by `WLClient` to the Worklight Server. The default value is 420 (7 minutes).
minAppWidth	Relevant only for iGoogle. The use of iGoogle is deprecated in Worklight version 5.0.5. Support might be removed in any future version. The minimum width for the application in pixels. If the application is contracted to less than this width, IBM Worklight automatically displays a message, which asks the user to expand the application. The default value is 170.
connectOnStartup	A Boolean value that indicates whether to connect to the Worklight Server. The default if no value is specified is `true`. However, the default value that is set in the `initOptions.js` file is `false`. The value `false` is appropriate if your app does not retrieve any corporate data on startup. Note, though, that any server features such as Remote Disable or Direct Update are only available after the app connects to the server. The value `true` is appropriate if your app must receive data from the server when it starts. However, the app might start more slowly.

Experimental IBM Worklight offline user documentation. This document is provided as is. In case of issues, refer to the online user documentation.

Property	Description
onConnectionFailure	A failure handling function that is invoked when connection to the Worklight Server, performed on initialization by default, or if the connectOnStartup flag is true, fails.
onUnsupportedVersion	A failure handling function that is invoked when the current version of the application is no longer supported (a newer application was deployed to the server). For more information about the signature of failure handling functions, see "The options object" on page 249.
onRequestTimeout	A failure handling function that is invoked when the init() request times out. For more information about the signature of failure handling functions, see "The options object" on page 249.
onUnsupportedBrowser	A failure handling function that is invoked when the application is running in an unsupported browser. For more information about the signature of failure handling functions, see "The options object" on page 249.
onDisabledCookies	A failure handling function that is invoked when cookies are displayed in the user's browser. For more information about the signature of failure handling functions, see "The options object" on page 249.
onUserInstanceAccessViolation	A failure handling function that is invoked when the user is trying to access an application that was provisioned to a different user. For more information about the signature of failure handling functions, see "The options object" on page 249.

Experimental IBM Worklight offline user documentation.
This document is provided as is.
In case of issues, refer to the online user documentation.

Property	Description

`onErrorRemoteDisableDenial`	A failure-handling function that is invoked when the server denies access to the application, according to rules defined in the Worklight Console. If this function is not provided, the application opens a dialog box, which displays an error message that is defined in the Worklight Console. When used, the function can provide an application-specific dialog box, or can be used to implement additional behavior in situations where the server denies access to the application. It is important to ensure that the application remains offline (not connected).
	
	You can add the following parameters:
	
	`message`
	This parameter contains the notification text that you defined in the Worklight Console, which indicates that an application is denied access to the Worklight Server.
	
	`downloadLink`
	This parameter contains the URL that you defined in the Worklight Console to download the new version of the application, which users can find in the appropriate application store.
	Example:
	`var wlInitOptions = {`
	`  connectOnStartup : true,`
	`  onErrorRemoteDisableDenial : function (message, download`
	`    WL.SimpleDialog.show(`
	`      "Application Disabled",`
	`      message,`
	`      [{text: "Close application", handler: function() {WL`
	`       {text: "Download new version", handler: function()`
	`    );`
	`  }`
	`};`

`onErrorAppVersionAccessDenial`	A failure-handling function that is invoked when the server denies access to the application, according to rules defined in the Worklight Console. If this function is used, the developer takes full ownership of the implementation and handling if Remote Disable took place. If the failure-handling function is not provided, the application opens a dialog box, which displays an error message that is defined in the IBM Worklight Console. **Note:** `onErrorAppVersionAccessDenial` is deprecated since V5.0.6. Instead, use `onErrorRemoteDisableDenial`.
`validateArguments`	A Boolean value, indicating whether the IBM Worklight Client runtime library validates the number and type of method parameters. Default is `true`.

Experimental IBM Worklight offline user documentation.
This document is provided as is.
In case of issues, refer to the online documentation.

Property	Description
`updateSilently`	A Boolean value, indicating whether Direct Update is performed without notifying the user before downloading new application resources. Default is `false`.

`onGetCustomDeviceProvisioningProperties`	A callback function that is invoked during the provisioning process of the device ID created by the app on the device. Typical implementation collects an out-of-band provisioning token from the user.
	
	The function receives a **`resumeDeviceProvisioningProcess`** argument, which must be called to resume the provisioning process, and transfers the custom provisioning data as a JSON hash map.
	
	Example:
	`In initOptions.js:`
	`var wlInitOptions = {`
	`  ...`
	`  ...`
	`  onGetCustomDeviceProvisioningProperties: collectCusto`
	`  ...`
	`}`
	
	`In application JavaScript file:`
	`function collectCustomProvisioningProperties (`
	`  resumeDeviceProvisioningProcess) {`
	`    // Collect provisioning token from user resumeDevic`
	`    {`
	`      token: token`
	`    }`
	`  );`
	`}`

`showCloseOnDirectUpdateFailure`	A Boolean value, indicating whether the **Close** button is shown in dialogs that are displayed after a Direct Update failure. Set this value to `false` to ensure that any dialogs displayed after a Direct Update failure do not show the **Close** button; that is, dialogs are modal for all mobile operating systems. This prevents users from continuing to use an application until a Direct Update succeeds or until the user completes a new installation of the application from the application store. The default is `true`.

Experimental IBM Worklight offline user documentation.
This document is provided "as is".
In case of issues, refer to the online user documentation.

Property	Description
showCloseOnRemoteDisableDenial	A Boolean value. If you set the value to `true`, whenever you use the Remote Disable feature from the Worklight Console to remotely disable an application, the dialog that is presented to users includes a **Get new version** button and a **Close** button. Clicking **Close** closes the dialog, but allows the user to continue working offline, with no connection to the Worklight Server. If you set the value to `false`, the behavior is as follows: • If you disable the application on the Worklight Console and specify a link to the new version, the dialog displays only the **Get new version** button. The **Close** button is not shown. The user has no choice but to update the application, and the user is forced to exit the application. • If you disable the application and do not specify a link to a new version, the dialog displays only the **Close** button. The default is `true`.

Note:

The `onSuccess` function is used to initialize the application.

If an `onFailure` function is not passed, a default `onFailure` function is called. If `onFailure` is passed, it overrides any specific failure-handling function.

Return Value

None.

WL.Client.invokeProcedure

This method invokes a procedure that is exposed by an IBM Worklight adapter

Syntax

```
WL.Client invokeProcedure (invocationData, options)
```

Parameters

Parameter	Description
invocationData	Mandatory. A JSON block of parameters. For a description of the structure of the parameter block, see "The WL.Client invokeProcedure JSON Parameter Block" on page 195.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues refer to the online user documentation.

Parameter	Description
options	Optional. A standard `options` object, as defined in “The `options` object” on page 249, augmented with the following property: • `timeout`: Integer. Number of milliseconds to wait for the server response before failing with a request timeout. The success handler of this call receives an augmented response that is described in “The WL.Client.invokeProcedure Success Handler Response Object” The failure handler of this call is called in two cases: • The procedure was called but failed. In this case, the `invocationResult` property is added to the response received by the failure handler. This property has the same structure as the `invocationResult` property returned to the success handler, but the value of the `isSuccessful` attribute is `false`. For the structure of the `invocationResult` property, see invocationResult. • A technical failure resulted in the procedure not being called. In this case, the failure handler receives a standard response object.

The WL.Client invokeProcedure JSON Parameter Block

The `WL.Client invokeProcedure` function accepts the following JSON block of parameters:

```
{
 adapter: 'adapter-name',
 procedure: 'procedure-name',
 parameters: []
}
```

The `JSON` block contains the following properties:

Property	Description
adapter	Mandatory. A string that contains the name of the adapter as specified when the adapter was defined.
procedure	Mandatory. A string that contains the name of the procedure as specified when the adapter was defined.
parameters	Optional. An array of parameters that is passed to the back-end procedure.

The WL.Client.invokeProcedure Success Handler Response Object

The success handler `response` object can contain the following properties:

Property	Description
invocationContext	The `invocationContext` object that was originally passed to the Worklight Server in the callback object.

Experimental IBM Worklight offline user documentation
This document is provided "as is"
In case of issues, refer to the online user documentation.

Property	Description

`invocationResult`	An object that contains the data that is returned by the invoked procedure, and the invocation status. Its format is as follows:
	`invocationResult = {`
	` isSuccessful: Boolean,`
	` errors: "Error Message"`
	` // Procedure results go here`
	`}`
	Where:
	• `isSuccessful` – Contains `true` if the procedure invocation succeeded, `false` otherwise. If the invocation failed, the failure handler for the request is called.
	• `errors` – An optional string array that contains the error messages.

Return Value

None.

Deprecated WL.Client.isConnected

This method is deprecated.

Syntax

```
WL.Client.isConnected()
```

Description

Note: This method is deprecated as of version 4.1.3. Use **WL.Device.getNetworkInfo** instead.

Returns `true` if the application is connected to the IBM Worklight Server.

Parameters

None.

WL.Client.isUserAuthenticated

This method checks whether the user is authenticated.

Syntax

```
WL.Client.isUserAuthenticated(realm)
```

Description

Checks whether the user is authenticated in a specified resource realm, or in the resource realm that was assigned to the application when it was deployed.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Parameters

Parameter	Description
realm	Optional. The name of a realm name defined in the authenticationConfig.xml file. If no value is specified, the method uses the resource realm assigned to the application when it was deployed.

Return Values

- `true` if the user is authenticated in the realm
- `false` otherwise

WL.Client.logActivity

This method is used to report user activity.

Syntax

```
WL.Client.logActivity(activityType)
```

Description

This method is used to report user activity for auditing or reporting purposes.

The IBM Worklight Server maintains a separate database table to store application statistics. For more information, see "Using raw data reports" on page 375.

Note: To ensure that the activity is stored in the database, set **reports.exportRawData** to `true` in the `worklight.properties` file.

Parameters

Parameter	Description
activityType	Mandatory. A string that identifies the activity.

Return Value

None.

WL.Client.login

This method logs in to a specific realm.

Syntax

```
WL.Client.login(realm, options)
```

Description

An asynchronous function. Logs in to a specific realm.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues refer to the online user documentation.

Parameters

Parameter	Description
`realm`	Mandatory. A realm that defines how the login process is performed. The realm can either be the one defined in the application descriptor, or a Facebook realm. **Note**: To log in to Facebook, the realm must be a realm which uses a Facebook authenticator, and therefore its name must start with `facebook.`.
`options`	Optional. A standard `options` object.

Return Value

None.

WL.Client.logout

This method logs out of a specified realm.

Syntax

```
WL.Client.logout(realm, options)
```

Description

An asynchronous function that logs out of a specified realm.

Parameters

Parameter	Description
`realm`	Optional. The realm to be logged out of. Specify `NULL` to log out of the resource realm assigned to the application when it was deployed.
`options`	Optional. A standard `options` object.

Return Value

None.

WL.Client.minimize

Minimize a widget on Adobe Air.

Syntax

```
WL.Client.minimize()
```

Description

Note: This method is only applicable to widgets that are running on Adobe AIR.

This method minimizes the AIR widget to the taskbar, or to the tray, as defined in the application descriptor.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

Parameters

None.

WL.Client.onDock, WL.Client.onUndock

Specify widget behavior on docking and undocking, in Windows.

Syntax

```
myOnDock = function () {
// Code here
}
myOnUndock = function () {
// Code here
}
```

Description

Note: These methods are applicable only to widgets that are running in Vista Sidebar.

To specify widget behavior on docking and undocking, provide an implementation for the WL.Client.onDock and WL.Client.onUndock callback functions. Neither of these methods receive any parameters.

In your initialization function, assign myOndock() and myOnUnDock() to WL.Client.onDock() and WL.Client.onUndock():

```
WL.Client.onDock = myOnDock;
WL.Client.onUndock = myOnUndock;
```

WL.Client.onShow, WL.Client.onHide

Specify widget behavior on showing and hiding, in Apple OS X.

The use of Apple OS X widgets is deprecated in Worklight version 5.0.5. Support might be removed in any future version.

Syntax

```
myOnShow = function () {
// Code here
}
myOnHide = function () {
// Code here
}
```

Description

Note: These methods are applicable only to widgets that are running on Apple OS X Dashboard.

Widgets running on Apple OS X Dashboard can be shown or hidden by pressing F12 on the Apple computer keyboard. Developers of OS X Dashboard widgets are instructed to stop any background processing while the widgets are hidden.

To specify the widget behavior on showing and hiding it, provide an implementation for the WL.Client.onShow and WL.Client.onHide methods. Neither of these methods take any parameters.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

In your initialization function, assign `myOnShow()` and `myOnHide()` to `WL.Client.onShow()` and `WL.Client.onHide()`:

```
WL.Client.onShow = myOnShow;
WL.Client.onHide = myOnHide;
```

WL.Client.reloadApp

This method reloads the application.

Syntax

```
WL.Client.reloadApp()
```

Description

This method reloads the application. It can be used to recover an application from errors. It is preferable to avoid using it and to use alternative error handling mechanisms instead. The method is mainly available for compatibility with earlier versions.

Note: The Apple OS X Dashboard does not allow a widget to automatically reload. Therefore, in this environment, the `reloadApp` method displays a dialog box that tells the user how to manually reload the widget.

Parameters

None.

Return Value

None.

WL.Client.removeGlobalHeader

This method removes the global HTTP header added by the `WL.Client.addGlobalHeader` API call

Syntax

```
WL.Client.removeGlobalHeader(headerName)
```

Description

Removes the global HTTP header added by the `WL.Client.addGlobalHeader` API call.

Parameters

Parameter	Description
headerName	Mandatory. The name of the header to be removed..

Return Value

None.

Example

```
WL.Client.removeGlobalHeader("MyCustomHeader");
```

Experimental IBM Worklight offline user documentation
This document is provided "as is". In case of issues, refer to the online user documentation.

WL.Client.setHeartBeatInterval

This method sets the interval of the heartbeat signal.

Syntax

`WL.Client.setHeartBeatInterval(`*`interval`*`)`

Description

Sets the interval of the heartbeat signal sent to the IBM Worklight Server to the specified number of seconds. The heartbeat is used 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`).

Parameters

Parameter	Description
`interval`	Mandatory. An integer value, denoting the interval in seconds between heartbeat messages automatically sent by `WLClient` to the IBM Worklight Server. An interval value of `-1` disables the heartbeat:`WL.Client.setHeartBeatInterval(-1)`

WL.Client.setUserPref

This method creates a user preference, or updates the value of an existing user preference.

Syntax

`WL.Client.setUserPref(key, value, options)`

Description

An asynchronous function that creates a user preference, or updates the value of an existing user preference, as follows:

- If a user preference with the specified user key is already defined, the user preference value is updated.
- If there is no user preference defined with the specified key, a new user preference is created with the specified key and value. However, if there are already 100 preferences, no preference is created, and the failure handler of the method is called.

Note: The local user preferences in the application are updated only when a successful response is received from the server.

Parameters

Parameter	Description
`key`	Mandatory. The user preference key. Can be up to 128 characters long.
`value`	Mandatory. The value of the user preference. Can be up to 3072 characters long.
`options`	Optional. A standard `options` object.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues refer to the online user documentation.

Return Value

None.

WL.Client.setUserPrefs

This method creates or updates one or more user preferences.

Syntax

```
WL.Client.setUserPrefs({key1:value1, key2:value2, ...}, options)
```

Description

An asynchronous function that creates one or more new user preferences, updates the values of one or more existing user preferences, or both. For each user preference key and value pair provided, the following occurs:

- If a user preference with the specified user key is already defined, the user preference value is updated.
- If there is no user preference defined with the specified key, a new user preference is created with the specified key and value.

If adding the new user preferences would result in the number of user preferences exceeding 100, then no user preferences are added or updated, and the failure handler of the method is called.

Note: The local user preferences in the application are updated only when a successful response is received from the server.

Parameters

Parameter	Description
`{key1:value1, key2:value2, ...}`	Mandatory. A hash object that contains user preference key and value pairs. The key can be up to 128 characters long. The value can be up to 3072 characters long.
`options`	Optional. A standard `options` object.

Return Value

None.

WL.Client.updateUserInfo

This method refreshes user data after an exception.

Syntax

```
WL.Client.updateUserInfo (options)
```

Description

Use this method when the application receives an exception after calling the `invokeProcedure()` method. The method refreshes the data for the following methods:

- `WL.Client.getUserName(realm)`

Experimental IBM Worklight online user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

- `WL.Client.getLoginName(realm)`
- `WL.Client.isUserAuthenticated(realm)`

After such an exception, you can verify the user authentication status by calling this function first, and then the `isUserAuthenticated()` method.

Parameters

Parameter	Description
`options`	Optional. A standard `options` object.

Return Value

None.

WL.App.copyToClipboard

Copy a string to the clipboard.

Syntax

```
WL.App.copyToClipboard(string)
```

Description

This method is applicable to iOS and Android.

It copies the specified string to the clipboard.

Parameters

Table 37. WL.App.copyToClipboard parameters

Parameter	Description
`string`	Mandatory. String. The text to be copied to the clipboard
`callback`	Optional. For Android environments only. The callback function that is called after the data is copied to the clipboard.

WL.Device.getNetworkInfo

Get network information from the device

Syntax

```
WL.Device.getNetworkInfo (callback)
```

Description

Fetches network information from the device and returns it to the specified callback function. Available on Android and iOS.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Parameters

Table 38. WL.Device.getNetworkInfo parameters

Parameter	Description
`callback`	Mandatory. The function that is called after the data is copied to the clipboard.

Return Value

The callback function receives a JSON structure, as described in the following table:

Table 39. WL.Device.getNetworkInfo properties available on Android and iOS

Property	Description	Availability on Android	Availability on iOS
isNetworkConnected	Mandatory. Whether the device has an IP address (that is, it is connected through Wi-Fi or a mobile network)	Yes	Yes
isAirplaneMode	Mandatory. Whether the device is in airplane mode or not	Yes	No
isRoaming	Mandatory. Whether the device is roaming (not on its home mobile network)	Yes	No
networkConnectionType	Mandatory. Returns `mobile` or `WIFI`	Yes	Yes
wifiName	Mandatory. Name of the Wi-Fi network, if connected	Yes	No
telephonyNetworkType	Mandatory. Type of the mobile network (such as `HSDPA` or `EDGE`)	Yes	No
carrierName	Mandatory. Name and ID of the mobile carrier	Yes	No

Experimental IBM Worklight online user documentation.
This document is provided "as is".
In case of issues, refer to the online user documentation.

Table 39. WL.Device.getNetworkInfo properties available on Android and iOS (continued)

Property	Description	Availability on Android	Availability on iOS
ipAddress	Mandatory. IP address of the device **Note:** The value **`ipAddress`** is set to the first non-null value of the possible four IP addresses in this order: • IPv4 Wi-Fi Address • IPv4 3GAddress • IPv6 Wi-Fi Address • IPv6 3GAddress For example, if both IPv4 Addresses are not present, the value **`ipAddress`** takes the value of the IPv6 Wi-Fi address.	Yes	Yes
Ipv4Addresses	Optional array that contains key value pairs: • `wifiAddress` - The IPv4 Wi-Fi address if it is present • `3GAddress` - The IPv4 3G address if it is present	Yes	Yes
Ipv6Addresses	Optional array that contains key value pairs: • `wifiAddress` - The IPv6 Wi-Fi address if it is present (only on iOS) • `3GAddress` - The IPv6 3G address if it is present	Yes	Yes

Example

```
WL.Device.getNetworkInfo(function (networkInfo) {
  alert (networkInfo.ipAddress);
  }
);
```

WL.NativePage.show

Switches the currently displayed, web-based screen with a natively written page

Syntax

```
WL.NativePage.show(className, callback, data);
```

Experimental IBM Worklight online user documentation
This document is provided "as is"
In case of issues, refer to the online user documentation.

Parameters

Table 40. WL.NativePage.show parameters

Parameter	Description
`className`	Mandatory. String. The name of the native class. For iOS, the name of the class (for example, `BarCodeController`). For Android, the complete name of the class and package (for example, `com.neebula.barcode.Scanner`).
`callback`	Mandatory. Function. A function object that is called when the native page switches back to the web view. This function is passed a single JSON object parameter when invoked.
`data`	Optional. Object. A JSON object that is sent to the native class. For iOS, The data must be single string or a flat record of strings.

Examples

```
// Good
WL.NativePage.show("com.scan.BarCode", function(data){alert(data);}, {key1 : 'value1'});
WL.NativePage.show("com.scan.BarCode", function(data){alert(data);}, {key1 : 'value1', key2 : 'value

// Bad
WL.NativePage.show("com.scan.BarCode", function(data){alert(data);}, {key1 : 'value1', innerStruct :
```

WL.Toast.show

Displays an Android toast box with the specified string.

Note: This object is only applicable to Android applications.

Syntax

```
WL.Toast.show (string)
```

Description

Displays an Android toast box with the specified string.

Parameters

Table 41. WL.Toast.show parameters – Android Only

Parameter	Description
`string`	Mandatory. String. The text to display in the Android toast.

Return Value

None

WL.BusyIndicator (object)

Display an indication that the application is busy.

Use the `WL.BusyIndicator` object to display a modal, dynamic graphical image when the application is temporarily "busy", that is, not responsive to user input. `WL.BusyIndicator` is implemented natively on iOS, Android, Windows Phone 7.5,

Experimental IBM Worklight offline user documentation.
This document is provided "as is".
In case of issues, refer to the online user documentation.

Windows Phone 8, and Windows 8. In other environments, it is implemented by using JavaScript in the `Busy.js` file. The implementations differ in their `option` parameters.

To change the appearance of the busy indicator, you can override the following CSS selectors: `#WLbusyOverlay`, `#WLbusy`, and`#WLbusyTitle`.

Example

```
var busyInd = new WL.BusyIndicator('content', {text : 'Loading...'});
```

WL.BusyIndicator (constructor):

Syntax of the WL.BusyIndicator constructor

Syntax

```
WL.BusyIndicator (containerId, options)
```

Parameters

Table 42. WL.BusyIndicator parameters

Parameter	Description
`containerId`	Optional string. The name of the HTML element in which the indicator is displayed. The indicator is centered horizontally and vertically within the element. If not provided or `null`, the element with ID `content` is used. Not relevant where the busy indicator is implemented natively, that is, on iOS, Android, Windows Phone 7.5, Windows Phone 8, and Windows 8.
`options`	Optional. A JSON hash object. See details in the following section.

Options for iPhone

text String.

bounceAnimation
Boolean.

Show a bounce animation when the busy indicator is displayed. Default: false.

opacity
Float.

Number in the range 0 - 1.

textColor
String.

Color name or color notation, such as "00FF00" or "green". Default: white.

strokeOpacity
Float.

fullScreen
Boolean.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Show the overlay over the entire screen. Default: false.

boxLength
Float.

Height and width of the overlay, when fullScreen is false. The Height value is automatically calculated based on the Width value provided.
Example:

```
var busy;
busy = new WL.BusyIndicator("content", {text: "Loading...", boxLength: 255.5};
```

duration
Double

Duration in seconds.

minDuration
Integer

Minimum duration in seconds.

Options for Android

text String.

Options for Windows Phone 7.5 and Windows Phone 8

None.

Options for Windows 8

None.

Options for Other Environments

text String.

Showing and Hiding the Busy Indicator:

Methods of WL.BusyIndicator

After the indicator is instantiated with a constructor, you can use the following functions:

To show the busy indicator:

```
busyInd.show();
```

To hide the busy indicator:

```
busyInd.hide();
```

To test whether the busy indicator is visible:

```
if (busyInd.isVisible()) {...};
```

Encrypted offline cache

Encrypted offline cache is a mechanism for storing sensitive data on the client application.

You can also use the JSONStore feature to obtain reliable secure on-device storage of data. If you previously used the Encrypted offline cache (EOC) feature, you can now use this improved on-device storage method for offline access. In addition,

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

the JSONStore provides the ability to populate and update data from an adapter on the server. This technique provides a better alternative for storing adapter data offline and synchronizing with a server the changes that were done when offline. If you are developing a Worklight hybrid app to target both iOS and Android, consider using JSONStore rather than EOC. HTML5 cache, as used in EOC, is not guaranteed to be persistent on future iOS versions. JSONStore uses the same encryption form and security mechanisms (PBKDF2 for key derivation from user password and AES 256) as EOC. EOC continues to be supported as a cross-platform on-device data store mechanism for iOS, Android, Windows, and BlackBerry, but no major technical updates will be made to the EOC feature set.

For more information about JSONStore, see “Data synchronization with JSONStore” on page 155

The cache uses HTML5 local storage to store user data. HTML5 imposes a limit of 5 MB, which is equivalent to approximately 1.3 MB of unencrypted text. If you exceed this limit, the behavior is undefined. If you use a large amount of cache, you might experience delays in processing it.

Data is stored in key-value pairs. Data is encrypted by using a 256-bit encryption key. The encryption key is itself encrypted, using a separate 256-bit encryption key. That key is generated from the user’s password by using the PKCS #5 PBKDF2 function.

The encrypted offline cache is available for mobile, desktop, and web environments that support HTML5.

As an alternative to encrypted offline cache, you can use a JSONStore object. For more information, see “Developing an app that uses data synchronization” on page 156.

WL.EncryptedCache – Exceptions

The following exceptions can be thrown by `WL.EncryptedCache` methods:

WL.EncryptedCache.ERROR_NO_EOC
: Thrown when **`create_if_none`** is `false` but no encrypted cache was previously initialized.

WL.EncryptedCache.ERROR_LOCAL_STORAGE_NOT_SUPPORTED
: Thrown when the HTML5 local storage interface is unavailable.

WL.EncryptedCache.ERROR_KEY_CREATION_IN_PROGRESS
: Thrown when the encrypted storage is processing an **`open`** or **`changeCredentials`** request.

WL.EncryptedCache.ERROR_EOC_CLOSED
: Thrown when the encrypted cache was not properly initialized by using **`WL.EncryptedCache.open`**.

WL.EncryptedCache.close:

Close encrypted cache.

Syntax

```
WL.EncryptedCache.close(onCompleteHandler, onErrorHandler)
```

Experimental IBM Worklight offline user documentation
This document is provided as is
In case of issues refer to the online user documentation.

Description

Closes the cache. The cache must be reopened with the user's credentials to be used again.

Parameters

Parameter	Description
onCompleteHandler	Mandatory. Function. A callback method that is invoked when the encrypted cache is ready for use. Signature: `successCallback` (*status*), where *status* can be `WL.EncryptedCache.OK`.
onErrorHandler	Mandatory. Function. A callback method that is invoked when the action fails. Signature: `failureCallback(`*status*`)`, where *status* can be `WL.EncryptedCache.ERROR_KEY_CREATION_IN_PROGRESS`.

Return Value

None.

WL.EncryptedCache.destroy:

Deletes encrypted cache.

Syntax

```
WL.EncryptedCache.destroy(onCompleteHandler, onErrorHandler)
```

Description

Completely deletes the encrypted cache and its storage. The cache does not need to be opened before it is destroyed.

Parameters

Parameter	Description
successCallback	Mandatory. Function. A callback method that is invoked when the action succeeds. Signature: `successCallback` (*status*), where *status* can be `WL.EncryptedCache.OK`.
failureCallback	Mandatory. Function. A callback method that is invoked when the action fails. Signature: `successCallback` (*status*), where *status* can be `WL.EncryptedCache.ERROR_KEY_CREATION_IN_PROGRESS`.

Return Value

WL.EncryptedCache.OK
The encryption data was successfully removed from memory.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

WL.EncryptedCache.open:

Open an existing cache, or create a cache.

Syntax

```
WL.EncryptedCache.open(credentials, create_if_none, onCompleteHandler, onErrorHandler)
```

Description

Opens an existing cache, or creates a cache, which is encrypted using the provided credentials. This method runs asynchronously because the key generation process is a lengthy process.

The process of creating a cache involves obtaining a random number from the IBM Worklight Server. Hence, the action of creating a cache requires that the app is connected to the IBM Worklight Server. After a cache is created, it can then be opened without a connection.

Parameters

Table 43. WL.EncryptedCache.open parameters

Parameter	Description
`credentials`	Mandatory. String. The credentials that are used to encrypt the stored data.
`create_if_none`	Mandatory. Boolean. Whether to create an encrypted cache if one does not exist.
`onCompleteHandler`	Mandatory. Function. A callback method that is invoked when the encrypted cache is ready for use. The signature of this method is `onCompleteHandler(`*`status`*`)`. The possible value for *status* is `WL.EncryptedCache.OK`
`onErrorHandler`	Mandatory. Function. A callback method that is invoked when the action fails. The signature of this method is `onErrorHandler(`*`status`*`)`. Possible values for *status* are: `WL.EncryptedCache.ERROR_KEY_CREATION_IN_PROGRESS`, `WL.EncryptedCache.ERROR_LOCAL_STORAGE_NOT_SUPPORTED`, `WL.EncryptedCache.ERROR_NO_EOC`, `WL.EncryptedCache.ERROR_COULD_NOT_GENERATE_KEY`, `WL.EncryptedCache.ERROR_CREDENTIALS_MISMATCH`

Return Value

None

WL.EncryptedCache.read:

Decrypts the value that is associated with the specified key.

Syntax

```
WL.EncryptedCache.read(key, successCallback, failureCallback)
```

Experimental IBM Worklight offline user documentation.
This document is provided "as is".
In case of issues, refer to the online user documentation.

Parameters

Table 44. WL.EncryptedCache.read parameters

Parameter	Description
Key	Mandatory. String. The key whose value needs to be decrypted.
successCallback	Mandatory. Function. A callback method that is invoked when the action succeeded. Signature: `successCallback (`*value*`)`, where *value* is the result of the read action.
failureCallback	Mandatory. Function. A callback method that is invoked when the action fails. Signature: `failureCallback(`*status*`)`, where *status* can be `WL.EncryptedCache.ERROR_EOC_CLOSED`.

Return Value

Decrypted value of the specified key.

WL.EncryptedCache.remove:

Removes a key-value pair from the cache.

Syntax

```
WL.EncryptedCache.remove(key, successCallback, failureCallback)
```

Description

Removes the key-value pair that is associated with *key*. Same as `WL.EncryptedCache.write(key, null)`.

Parameters

Table 45. WL.EncryptedCache.remove parameters

Parameter	Description
Key	Mandatory. String. The key to remove.
successCallback	Mandatory. Function. A callback method that is invoked when the action succeeded. Signature: `successCallback (`*status*`)`, where *status* can be `WL.EncryptedCache.OK`.
failureCallback	Mandatory. Function. A callback method that is invoked when the action fails. Signature: `failureCallback(`*status*`)`, where *status* can be `WL.EncryptedCache.ERROR_EOC_CLOSED`.

Return Value

None

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

WL.EncryptedCache.write:

Store a key-value pair in the cache.

Syntax

```
WL.EncryptedCache.write(key, value, successCallback, failureCallback)
```

Description

Stores the key-value pair, encrypting **`value`** and associating it with **`key`** for later retrieval.

Parameters

Table 46. WL.EncryptedCache.write parameters

Parameter	Description
`key`	Mandatory. String. The key to associate the data (**`value`**) with.
`value`	Mandatory. String. The data to encrypt. When set to `null`, the key is removed.
`successCallback`	Mandatory. Function. A callback method that is invoked when the action succeeds. Signature: `successCallback (`*`status`*`)`, where *status* can be `WL.EncryptedCache.OK`.
`failureCallback`	Mandatory. Function. A callback method that is invoked when the action fails. Signature: `failureCallback(`*`status`*`)`, where *status* can be `WL.EncryptedCache.ERROR_EOC_CLOSED`.

Return Value

None

WL.Client.getLanguage

Return the language code of the language being used.

Syntax

```
WL.Client.getLanguage()
```

Description

Note: This method is not relevant for mobile operating systems. Use mobile locale methods instead.

This method returns the language or dialect code of the language currently being used for the application.

Parameters

None.

Experimental IBM Worklight offline user documentation
This document is provided "as-is".
In case of issues refer to the online user documentation.

Return Value

The language or dialect code of the currently set language, or NULL if no language is set. The language or dialect code has the format *ll* or *ll-cc*, where *ll* is a two-letter ISO 639-1 language code and *cc* is a two-letter ISO3166-1-alpha-2 country code.

WL.JSONStore

Provides the application programming interface (API) for storing JSON data locally, it can be linked to an adapter for data synchronization.

The JSONStore feature is only available on iOS and Android devices, and simulators.

Definitions

WL.JSONStore
Creates JSON Document collections by the **init** method.

Collection
A group of related documents.

Document
A JavaScript object that has an **_id** key that holds an integer and a **JSON** key that holds a JavaScript object. Document is an internal structure that we generate when you add or store data, do not modify **_id**.

```
var doc = {_id : 0, json: {fn : 'carlos', age : 99, active : false}};
```

Array of Documents
An array that holds only Documents.

```
var doc = [{_id : 0, json : {fn : 'carlos', age : 99, active : false}}];
```

searchFields

Defines the keys in JavaScript objects that are indexed, thus determining what you can query in a collection. The keys in the searchFields object must correspond to paths in the stored JSON object. Search field keys are applied to the JSON objects in a style similar to `object['keyPart1]['keyPart2']`. When a searchField is in the JavaScript object, it is indexed only if the value is a simple type (integer, number, boolean, or string). The values for search fields are type hints and must be one of 'string', 'integer', 'number', or 'boolean'. The type declared in the searchField need not to match the type of the item that is matched at runtime, but the better the match the better the optimization that can be done.

In this example, the fields **`fn`**, **`age`**, **`gpa`** and **`active`** match keys that are found only at the top level of the JavaScript object: `var myObj = { age: 42 }`, and would not match `var myObj2 = { person : {age : 18 } }`, the search field must be **`person.age`** to match this case.

```
var searchfields = {    fn : 'string',
                age : 'integer',
                gpa : 'number', //floating point or int
                active : 'boolean',
                'address.state' : 'string' };
```

Arrays are handled in a pass-through fashion, meaning that you cannot index an array or a specific index of the array (`arr[n]`) but you can index objects inside an array. For example:

Experimental IBM Worklight online user documentation
This content is provided "as is".
In case of issues, refer to the online user documentation.

```
var myObj = {
    customers : [
        { fn: "tim", age: 31 },
        { fn: "carlos", age: 11 }
    ]
};
```

searchField keys `customers.fn` and *`customers.age`* then match the values in the objects inside the customers array. However, 'customers' would not be matched, as the value is an array.

Query

A JavaScript object that does not contain nested objects or arrays. Keys must be specified in the search fields or be the special _id identifier.

```
var query = {_id : 0};
var query = {fn : 'carlos'};
var query = {age : 99};
var query = {active : false};
var query = {'address.state' : 'TX'};
```

Adapter

Collections might be linked to an adapter, which is used to push data by starting the specified adapter procedures that are based on the local modifications performed. For example: A document that was removed from the local collection is passed to the remove procedure in the adapter. There is an optional function that you can pass by the **accept** key to the adapter object that determines whether the document we try to push is marked as pushed in local storage. **`timeout`** is an optional parameter that if specified passes a value (in millis) to the WL.Client.invokeProcedure function during push operations. See the "WL.Client.invokeProcedure" on page 194 for more details.

You might optionally include a **`load`** key with an object that tells the collection where to load the initial set of data. The procedure name must be part of the adapter you linked to the collection, you can pass any number of parameters through the parameters array to the procedure or an empty array for no parameters and you must supply a key that is used to determine what you want to store in the invocation result (`response.invocationResult[key]`). See WL.JSONStore.load for more details.

```
var adapter = {name: 'customerAdapter',
    add: 'addProcedureInCustomerAdapterName',
    remove: 'removeProcedureInCustomerAdapterName',
    replace: 'replaceProcedureInCustomerAdapterName',
    load: {
        procedure: 'getCustomers',
        params: [],
        key: "customers"
    },
    accept: function (data) {
                return (data.status === 200);
            }
    },timeout: 3000
   };
```

Options

A JavaScript object that contains additional options you can pass to a specific method. The *onSuccess* and *onFailure* keys you pass by using the options object, are deprecated in favor of Promises. These success and

Experimental IBM Worklight documentation. This document is intended for online user documentation. In case of issues refer to the online user documentation.

failure callbacks are specific to the function you are calling, for example the *onSuccess* function that is passed to **initCollection** is called only when **initCollection** is successful.

```
var win =   function (data) { };
var fail =  function (data) { };
var options = {onSuccess: win, onFailure: fail};
```

Promises

All the asynchronous functions in the API currently support jQuery compatible promises. A promise object is returned after a JSONStore asynchronous operation is called (find, add, remove, and so on). Promises have the following methods:

```
  .then(success callbackFunction, failure callbackFunction);
  .done(success callback);
  .fail(failure callback);
```

see jQuery's API documentation for more details. The failure callback, passed as either the second parameter of **`.then`** or the first parameter of **`.fail`** returns an error object, which contains some of these keys: source, error code, message, collection name, user name, document, and response from the server. A failure is then initiated to the nearest error handler. You can use **`WLJQ.when(promise1, promise2).then(success callback, failure callback)`** if you need **`promise1`** and **`promise2`** to finish before calling the callbacks. The deprecated **`WL.JSONStore.initCollection`** is a special case, you must call **`.promise`** on the collection instance.

```
var collections = {
   customers : {
    searchFields : { fn: 'string' }
   }
  };

  WL.JSONStore.init(collections)

  .then(function () {
   //collection is initialized at this point
   return WL.JSONStore.get('customers').add({name: 'carlos'});
  })
  .then(function (res) {
   //res = 1, since one document was added to the collection
   return WL.JSONStore.get('customers').count();
  })
  .done(function (res) {
   //res = 1, since count returns the total number of documents in the collection
  })
  .fail(function (obj) {
   WL.Logger.debug(obj.toString()); //obj may contain some of these keys:
   //obj.src = operation that failed (eg. 'add', 'count', etc.)
   //obj.err = error code (eg. -50)
   //obj.msg = error code message ('PERSISTENT_STORAGE_FAILURE')
   //obj.col = collection name (eg. 'collection')
   //obj.usr = username (eg. 'jsonstore')
   //obj.doc = document (eg. {_id: 1, jsonstore: {name: 'carlos'}})
   //obj.res = response from the server
  });
```

Events

You can listen to events and capture successful and failure status codes and data. The following assumes jQuery >1.7 or using WLJQ.

```
$(document.body).on('WL/JSONSTORE/SUCCESS', function (evt, data, src, collectionName, more){
        if(src === 'find'){
            console.log(status);
            if(typeof data !== 'undefined'){
```

Experimental IBM Worklight online user documentation
This document is provided for the online documentation.
In case of issue, refer to the online documentation.

```
              console.log(data);
          }
      }
  });
```

You can also listen to the WL/JSONSTORE/FAILURE event.

additionalSearchFields

Defines additional fields that are searchable without modifying the stored document. **Usecases** for additionalSearchFields include“ tagging” data and forming relationships.

```
//Note that this example has certain elements omitted for brevity,
//see the documentation for init, add, and find
//for complete examples of those functions.

var orders = [
 {
  orderid : 23,
  item : 'tasty coffee'
 },
 {
  orderid : 99,
  item : 'good book'
 }
];
//Appear in objects to add to the collection
var searchFields = { orderid: 'integer', item: 'string' };

//Do not appear in objects to add to the collection
var addSearchFields = { customerId : 'string' };

var orderCollection = WL.JSONStore.init({
 orders: {
  searchFields: searchFields,
  additionalSearchFields : addSearchFields
 }
});

//call this after init finishes
orderCollection.store(orders, {additionalSearchFields : { customerId: 'abc123'} },<store

//call this after init finishes
orderCollection.find({customerId: 'abc123'}, <find options>);
```

Find calls the onSuccess callback with a parameter that contains the following data:

```
[
 {
  _id : 1,
  json : {
   orderid : 23,
   item : 'tasty coffee'
  }
 },
 {
  _id : 2,
  json : {
   orderid :99,
   item : 'good book'
  }
 }
];
```

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

Notice how the 'customerId' field was not added to the actual document, but is available as a searchable field in the find function.

List of error codes

-50 = "PERSISTENT_STORE_NOT_OPEN";

-40 = "FIPS_ENABLEMENT_FAILURE";

-12 = "INVALID_SEARCH_FIELD_TYPES";

-11 = "OPERATION_FAILED_ON_SPECIFIC_DOCUMENT";

-10 = "ACCEPT_CONDITION_FAILED";

-9 = "OFFSET_WITHOUT_LIMIT";

-8 = "INVALID_LIMIT_OR_OFFSET";

-7 = "INVALID_USERNAME";

-6 = "USERNAME_MISMATCH_DETECTED";

-5 = "DESTROY_REMOVE_PERSISTENT_STORE_FAILED";

-4 = "DESTROY_REMOVE_KEYS_FAILED";

-3 = "INVALID_KEY_ON_PROVISION";

-2 = "PROVISION_TABLE_SEARCH_FIELDS_MISMATCH";

-1 = "PERSISTENT_STORE_FAILURE";

0 = "SUCCESS";

1 = "BAD_PARAMETER_EXPECTED_INT";

2 = "BAD_PARAMETER_EXPECTED_STRING";

3 = "BAD_PARAMETER_EXPECTED_FUNCTION";

4 = "BAD_PARAMETER_EXPECTED_ALPHANUMERIC_STRING";

5 = "BAD_PARAMETER_EXPECTED_OBJECT";

6 = "BAD_PARAMETER_EXPECTED_SIMPLE_OBJECT";

7 = "BAD_PARAMETER_EXPECTED_DOCUMENT";

8 = "FAILED_TO_GET_UNPUSHED_DOCUMENTS_FROM_DB";

9 = "NO_ADAPTER_LINKED_TO_COLLECTION";

10 =
"BAD_PARAMETER_EXPECTED_DOCUMENT_OR_ARRAY_OF_DOCUMENTS";

11 =
"INVALID_PASSWORD_EXPECTED_ALPHANUMERIC_STRING_WITH_LENGTH_GREATER_T

12 = "ADAPTER_FAILURE";

13 = "BAD_PARAMETER_EXPECTED_DOCUMENT_OR_ID";

14 = "CAN_NOT_REPLACE_DEFAULT_FUNCTIONS";

15 = "COULD_NOT_MARK_DOCUMENT_PUSHED";

16 = "COULD_NOT_GET_SECURE_KEY";

17 = "FAILED_TO_LOAD_INITIAL_DATA_FROM_ADAPTER";

18 =
"FAILED_TO_LOAD_INITIAL_DATA_FROM_ADAPTER_INVALID_LOAD_OBJ";

19 = "INVALID_KEY_IN_LOAD_OBJECT";

20 = "UNDEFINED_PUSH_OPERATION";

21 = "INVALID_ADD_INDEX_KEY";

22 = "INVALID_SEARCH_FIELD";

23 = "CLOSING_ALL";

24 = "CHANGING_PASSWORD";

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

25 = "DURING_DESTROY";

26 = "CLEARING_COLLECTION";

27 = "INVALID_PARAMETER_FOR_FIND_BY_ID";

Typical Usage

The JSONStore can be used to create a collection either from an adapter or otherwise. When tied to an adapter, the API supports a convention of tying the various sync operations (pushing to server) based on the action the user can perform on the local collection in the JSONStore.

- Decide whether the collections, as part of the JSONStore, must be encrypted. If there is a requirement to secure the data at rest, send a password via the options object when you call WL.JSONStore.init.
- If you need multiple stores, then send a *username* via the options object when you call **`WL.JSONStore.init`**.
- Start with defining the collections, then initialize them with WL.JSONStore.init, see, WL.JSONStore.init. This action includes defining adapter configuration, collection name, and the **searchfields** options.
- After you initialize your collections, you can **`get`** them with **`WL.JSONStore.get`** ('collection-name') and it returns the right JSONStoreInstance.
- You can load data from an adapter by using load and store new data by calling add on the JSONStoreInstance.
- Your users can then find and work with the collection locally: they can replace, add or remove JSON Documents.
- Calling push on a JSONStoreInstance sends data that has been changed to your backend via an adapter. isPushRequired, getPushRequired and pushRequiredCount provide further information about the state of the collection.
- You can optionally close the collection after you use it by closeAll which closes the JSONStore and the collections in it.

WL.JSONStore.add:

Adds data to a collection.

Syntax

`add(data,[options])Promise`

Description

Adds data to a collection, creates a new Document or Documents. The documents require “WL.JSONStore.push” on page 234, unless `{push: false}` is specified.

Parameters

`data`

Object or Array of Objects. Data to be added to the collection.

`options`

Optional.

Options.

Additional options: additionalSearchFields: {}'

Experimental IBM Worklight offline documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Returns

`Promise:`
`OnSuccess`
Integer with the amount of data stored

`onFailure`
An error code

Example

```
//See .init for context
WL.JSONStore.get('customers').add({fn: 'carlos'})
.then(function (res) {
//res => number of documents added
})
.fail(function (errobject) {
WL.Logger.debug(errobject.toString());
});
//Deprecated Example:
var data = {fn: 'jeremy', age: 88, active: true};
collection.add(data, options);
```

WL.JSONStore.changePassword:

Changes the password for the internal storage.

Syntax

```
changePassword(oldPW, newPW, user,[options]) Promise static
```

Description

Changes the password for the internal storage. You must have a collection initialized before calling change password. Deprecated but currently supported function signature: **`changePassword(oldPW, newPW, options)`**, the user is assumed to be the default user: **jsonstore**.

Parameters

`oldPW`

String

The old password. Must be alphanumeric ([a⌂z, A⌂Z, 0⌂9]) with at least 1 character.

`newPW`

String

The new password. Must be alphanumeric ([a⌂z, A⌂Z, 0⌂9]) with at least 1 character.

`user`

String

The user name. Must be an alphanumeric string ([a⌂z, A⌂Z, 0⌂9]) with length greater than 0. See WL.JSONStore.initCollection for more details.

`options`

Optional

Options

Experimental IBM Worklight offline user documentation.
This document is provided "as is".
In case of issues, refer to the online user documentation.

Returns

`promise`
  promise

Example

```
var oldPW = 'myOldPassword',
newPW = 'newSecret',
user = 'tim'; //optional, default 'jsonstore'
WL.JSONStore.changePassword(oldPW, newPW, user)
.then(function () {
//the password has been changed
})
.fail(function (errobject) {
WL.Logger.debug(errobject.toString());
});
//Deprecated Example:
var win = function () {
console.log('SUCCESS');
};
var fail = function (err) {
console.log('FAILURE');
};
WL.JSONStore.changePassword(oldPW, newPW, user, {onSuccess: win, onFailure: fail});
```

WL.JSONStore.clearPassword:

Deprecated. Clears the password

Syntax

```
clearPassword() Boolean deprecated
```

Description

Removes the password from memory. This function is deprecated.

Deprecated

Use WL.JSONStore.init

Returns

`Boolean`
  `true` if the password stored in memory was set to null, `false` if there was no password in memory or if it was not set to null.

Example

```
WL.JSONStore.clearPassword();
```

WL.JSONStore.closeAll:

Closes the persistent store.

Syntax

```
closeAll ([options]) Promise static
```

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Description

Closes all the collections in the JSONStore. After a **`closeAll`**, each collection in the store must have WL.JSONStore.init called again before that collection can be used.

Note: If the collections in the persistent store are password protected, the password must be specified during *init*. See "WL.JSONStore.initCollection" on page 229.

Parameters

`options`

Optional

Options

Returns

`promise`
promise

Example

```
WL.JSONStore.closeAll()
.then(function () {
//close all finished
})
.fail(function (errobject) {
WL.Logger.debug(errobject.toString());
});
//Deprecated Example:
var win = function () {
console.log('SUCCESS');
};
var fail = function (err) {
console.log('FAILURE');
};
WL.JSONStore.closeAll({onSuccess: win, onFailure: fail});
```

WL.JSONStore.count:

Number of documents in the collection

Syntax

```
count([options])Promise
```

Description

Number of documents in the collection (not including those marked 'removed').

Parameters

`options`
Optional

Options

Returns

`Promise:`
`onSuccess`
Integer with the number of documents in the collection.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

onFailure
An error code.

Example

```
//See .init and .add for context
WL.JSONStore.get('customers').count()
.then(function (res) {
//res => number of documents inside the collection
})
.fail(function (errobject) {
WL.Logger.debug(errobject.toString());
});
//Deprecated Example:
var win = function (data) {
console.log(data);
=> 5
};
var options = {onSuccess: win, onFailure: fail};
collection.count(options);
```

WL.JSONStore.destroy:

A complete data wipe for all users, destroys the internal storage and clears security artifacts.

Syntax

```
destroy([options]) Promise static
```

Parameters

options
Optional

Options

Returns

Promise
Promise

Example

```
WL.JSONStore.destroy()
.then(function () {
//all the stores and keys for decrypting the store are removed from disk
})
.fail(function (errobject) {
WL.Logger.debug(errobject.toString());
});
//Deprecated Example:
var win = function (status) {
console.log('SUCCESS');
};
var fail = function (status) {
console.log('FAILURE');
};
WL.JSONStore.destroy({onSuccess: win, onFailure: fail});
```

WL.JSONStore.documentify:

Creates a Document.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Syntax

```
documentify (id, data) Document static
```

Parameters

id Integer

ID for the document

data
Object

JSON data for the Document

Returns

Document
Returns the document.

onFailure
Returns an error code.

Example

```
var doc = WL.JSONStore.documentify(1, {fn: 'carlos', age: 99, active: false});
console.log(doc);
=> {_id: 1, json: {fn: 'carlos', age: 99, active: false}}
```

WL.JSONStore.enhance:

Add a new function to a collection's prototype.

Syntax

```
enhance(name, func,) Integer
```

Parameters

name
String

Function name

func
Function

The function to add

Returns

Integer
0 if success, or an error code

Example

```
//Definition
collection.enhance('findByName', function (name) {
return this.find({fn: name});
});
//Usage - see .init for context
WL.JSONStore.get('customers').findByName('carlos')
.then(function (res) {
//res => all documents that have a fn (first name) of 'carlos'
```

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

```
})
.fail(function (errobject) {
WL.Logger.debug(errobject.toString());
});
```

WL.JSONStore.erase:

Removes Documents from a collection.

Syntax

```
erase(doc, [options]) OnSuccess deprecated
```

Description

Same as **remove**, but really does remove the document from the internal storage instead of marking it for removal and then really removing it when you call **push** or **pushSelected** with that specific document.

Deprecated, use "WL.JSONStore.remove" on page 237

Parameters

doc
: Document or Array of Documents or Query or Integer.

  The Integer is an **_id**.

options
: Optional

  Options

Returns

OnSuccess
: Integer with the number of documents removed.

onFailure
: An error code.

Example

```
var doc = {_id : 0, json: {fn : 'carlos', age : 99, active : false}};
collection.erase(doc, options); //Remove a Document
//or
collection.erase([doc], options); //Remove an Array of Documents
//or
collection.erase(1, options); //Remove by _id
//or
collection.erase({fn: 'carlos'}, options); //Remove all Documents that match {fn: 'carlos
```

WL.JSONStore.find:

Returns documents that are stored in the collection that match the query. Query matching returns partial results. It is not an exact match. For example, the query *{name: 'carl'}* matches "carlos" and "carl". To find all documents use the following query: *var query = {}*.

Syntax

```
find(query, [options]) Promise
```

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Parameters

query
Query

options
Optional

Options

Returns

Promise:
OnSuccess
An Array of Documents or an empty Array if no matches

onFailure
An error code.

Example

```
//See .init and .add for context
var query = {fn: 'carlos'}
WL.JSONStore.get('customers').find(query)
.then(function (res) {
//res => results from find
})
.fail(function (errobject) {
WL.Logger.debug(errobject.toString());
});
//Deprecated Example:
var query = {fn: 'carlos'};
var win = function (data) {
console.log(data);
=> [{_id : 0, json: {fn : 'carlos', age : 99, active : false}}];
};
var options = {onSuccess: win, onFailure: fail};
collection.find(query, options);
```

WL.JSONStore.findAll:

Returns all the documents stored in the JSON Store.

Syntax

```
findAll ([options]) Promise
```

Parameters

options
Optional

Options

Returns

Promise:
OnSuccess
An Array of Documents or an empty Array if the collection is empty.

onFailure
An error code.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Example

```
//See .init and .add for context
WL.JSONStore.get('customers').findAll()
.then(function (res) {
//res => results from findAll
})
.fail(function (errobject) {
WL.Logger.debug(errobject.toString());
});
//Deprecated Example:
var win = function (data) {
console.log(data);
=> [{_id : 0, json: {fn : 'carlos', age : 99, active : false}}];
};
var options = {onSuccess: win, onFailure: fail};
collection.findAll(options);
```

WL.JSONStore.findById:

Returns one or more documents that match the ID or ID's supplied to the function.

Syntax

```
findById (id, [options]) Promise
```

Parameters

`id` Integer or Array of Integers

Integer values must be greater than 0.

`options`
Optional

Options

Returns

`Promise:`
`OnSuccess`
An Array of Documents or an empty Array if no matches

`onFailure`
An error code.

Example

```
//See .init and .add for context
WL.JSONStore.get('customers').findById(1)
.then(function (res) {
//res => results from find
})
.fail(function (errobject) {
WL.Logger.debug(errobject.toString());
});
//Deprecated Example:
var id = 1;
//You can also pass id = [1,2,3] if you want the first 3 documents in the JSONStore
var win = function (data) {
console.log(data);
=> [{_id : 1, json: {fn : 'carlos', age : 99, active : false}}];
};
var options = {onSuccess: win, onFailure: fail};
collection.findById(id, options);
```

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

WL.JSONStore.get:

Returns a *JSONStoreInstance* linked to a collection

Syntax

```
get (collection) JSONStoreInstance static
```

Description

Returns a *JSONStoreInstance* linked to a collection, or undefined if the instance is not found. The instances are populated with *WL.JSONStore.init*. The following methods clear the instances that are stored: *WL.JSONStore.init* with {clear: true}, **`WL.JSONStore.destroy`** and **`WL.JSONStore.closeAll`**. Instances must not be altered. To update values call *WL.JSONStore.init* again.

Parameters

`collection`
: string

  Name of the collection instance you want to get

Returns

`JSONStoreInstance`

Example

```
WL.JSONStore.get('customers') //returns the JSONStoreInstance
.findAll() //example of an operation performed on a JSONStoreInstance
.then(function (res) {
//res => array of all documents inside the 'customers' collection
});
.fail(function (errobject) {
WL.Logger.debug(errobject.toString());
});
```

WL.JSONStore.getErrorMessage:

Returns the message associated with a status code.

Syntax

```
getErrorMessage(statusCode) String static
```

Parameters

`statusCode`
: Integer

Returns

`String`
: The Error Message that is associated with the status code or 'Not Found' if you pass an invalid value (non-integer) or a non-existent status code.

Example

```
WL.JSONStore.getErrorMessage(-50);
=> "PERSISTENT_STORE_NOT_OPEN"
```

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

WL.JSONStore.getPushRequired:

Get all the Documents that are unpushed.

Syntax

```
getPushRequired([options]) Promise
```

Parameters

`options`
: Optional

  Options

Returns

`Promise:`
`onSuccess`
: Array of Documents that are not pushed.

`onFailure`
: An error code.

Example

```
//See .init and .add for context
WL.JSONStore.get('customers').getPushRequired()
.then(function (res) {
//res => array of documents that need to be pushed
})
.fail(function (errobject) {
WL.Logger.debug(errobject.toString());
});
//Deprecated Example:
var win = function (data) {
console.log(data);
=> [ {_id: 1, json: {fn: 'jeremy', age: 88, active: true} }]
};
var options = {onSuccess: win, onFailure: fail};
collection.getPushRequired(options)
```

WL.JSONStore.initCollection:

Creates a new object to interact with a single collection.

Syntax

```
initCollection (name, searchfields, [options]) JSONStoreInstance deprecated static
```

Description

Creates a new object to interact with a single collection. **`initCollection`** must be called sequentially, meaning the previous **`initCollection`** must finish before trying to call **`initCollection`** again. If local storage for the collection does not exist, it is provisioned with the **`searchFields`**. Otherwise, the **`searchFields`** are validated against the **`searchFields`** used to originally provision the collection. If you are using *usernames*, you must call **`WL.JSONStore.closeAll`** to "logout" the current user, then you can call **`WL.JSONStore.initCollection`** with another *username*. The following example shows how to supply a username and a password, both are optional. If no username is passed, it uses the default one. You cannot use the following default *usernames*: *jsonstore*, *JSONStoreKey*, *dpk*.

Experimental IBM Worklight online user documentation
This document is provided "as is".
In case of issues refer to the online user documentation.

Deprecated, use **WL.JSONStore.init**

Parameters

name
: String

  Collection name.

searchfields
: searchFields

options
: Optional

  Options

Additionally, you can link a collection to an adapter. You can also pass **load:true** and it checks whether the collection is empty, and load data by using the adapter that you defined to get data. You can pass a *username* by using alphanumeric strings only: [a-z, A⌂Z, 0⌂9]) and a password.

Returns

JSONStoreInstance
: The collection will not be usable until the promise is resolved, or the successful callback is called.

onSuccess
: Called if successful.

onFailure
: Called if it was unsuccessful with an error code.

Example

```
var name = 'customers';
var searchFields = { fn: 'string',
age: 'integer',
active: 'boolean' };
var adapterDefinition = { name: 'customerAdapter',
add: 'addProcedureInCustomerAdapterName',
remove: 'removeProcedureInCustomerAdapterName',
replace: 'replaceProcedureInCustomerAdapterName',
load: {
procedure: 'getCustomers',
params: [],
key: "customers"
},
accept : function (data, doc) { //doc is the document pushed via the adapter
return (data.status === 200); //data is what we got back from the adapter
}
};
var options = {adapter: adapterDefinition};
//[Optional] You may assign a username to the store:
options.username = 'carlos';
//[Optional] If you want encryption you need to supply a password:
options.password = '12345';
var c = WL.JSONStore.initCollection(name, searchFields, options);
c.promise
.then(function (res) {
//res is 0 if a new collection was created, or 1 if an existing collection was opened
})
.fail(function (errobject) {
WL.Logger.debug(errobject.toString());
});
```

Experimental IBM Worklight offline user documentation
In case of issues, refer to the online user documentation.
This document is provided "as is".

```
//Deprecated Example:
//Create success and failure callbacks
var win = function (status) {
console.log('SUCCESS');
if (status === 1) {
console.log('Collection already existed');
} else if (status == 0) {
console.log('New Collection');
}
};
var fail = function (err) {
console.log('FAILURE');
//Display the error message:
console.log(WL.JSONStore.getErrorMessage(err));
//Calling getErrorMessage is equivalent to something like this:
//if (err === -1) {
// console.log('PERSISTENT_STORE_FAILURE');
//} else if (err === -2){
// console.log('PROVISION_TABLE_SEARCH_FIELDS_MISMATCH');
//} else if (err === -3) {
// console.log('INVALID_KEY_ON_PROVISION');
//} else if(err == 16) {
// console.log('COULD_NOT_GET_SECURE_KEY');
//}
};
//Add the success and failure callbacks to options
var options = {adapter: adapterDefinition, onSuccess: win, onFailure: fail};
var collection = WL.JSONStore.initCollection(name, searchFields, options);
```

WL.JSONStore.init:

Syntax

`init (collections, [options]) Promise static`

Description

Initializes a set of collections. See get to retrieve *JSONStoreInstances*. There is minimal overhead in initializing all the collections when an application starts. Search fields are given a type hint and represent values we index on a specific collection. Refer to the JSONStore Overview for further details on the collections object, such as **`additionalSearchfields`**. You can call **`init`** multiple times with different collections and it initializes without affecting collections that are already initialized. Passing **`{clear: true}`** clears the *JSONStore* instances without removing its contents from the store. For encrypted collection sets, the password is only required the first time **`init`** is called. See WL.JSONStore.closeAll and WL.JSONStore.destroy to logout the current user or destroy the contents of the store. See removeCollection to remove the contents of a specific collection from disk.

Parameters

`Collections`

Object

Collections that can be initialized. See example for format.

`options`

Optional

Options

Username (string [a-z, A⌂Z, 0⌂9]), password (string) and clear (boolean).

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Returns

`Promise`

The *Promise* is resolved when all collections are initialized. If any collection fails to initialize, the *Promise* is rejected and no *JSONStoreInstances* are available.

Example

```
var collections = {
customers : {
searchFields : {fn: 'string', age: 'integer', active: 'boolean'},
//Adapter is optional:
adapter : { name: 'customerAdapter',
add: 'addProcedureInCustomerAdapterName',
remove: 'removeProcedureInCustomerAdapterName',
replace: 'replaceProcedureInCustomerAdapterName',
load: {
procedure: 'getCustomers',
params: [],
key: "customers"
},
accept : function (data, doc) { //doc is the document pushed via the adapter
return (data.status === 200); //data is what we got back from the adapter
}
}
},
orders: {
searchFields : {name: 'string', stock: 'boolean'}
}
};
var options = { //all optional
username: 'carlos', //default: 'jsonstore'
password: '123' //default: no encryption
}
WL.JSONStore.init(collections, options)
.then(function (res) {
//res => Mutable object of all the JSONStoreInstances
return WL.JSONStore.get('customers').add({fn: 'carlos', age: 99, active: true});
})
.then(function (res) {
//res => number of documents added to the collection
})
.fail(function (errobject) {
WL.Logger.debug(errobject.toString());
});
```

WL.JSONStore.isPushRequired:

Determines if a Document is pushed.

Syntax

```
isPushRequired(doc, [options]) Promise
```

Parameters

`doc`

Document or Integer. I

The integer is an **`_id`**.

`options`

Optional.

Options.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Returns

`Promise:`

`onSuccess`
: `true` if it is pushed and `false` otherwise

`onFailure`
: An error code

Example

```
//See .init and .add for context
WL.JSONStore.get('customers').isPushRequired(0) //{_id : 0}
.then(function (res) {
//res => true if document needs to be pushed, false otherwise
})
.fail(function (errobject) {
WL.Logger.debug(errobject.toString());
});
//Deprecated Example:
var doc = {_id : 0, json: {fn : 'carlos', age : 99, active : false}};
var win = function (data) {
console.log(data);
=> false
};
var options = {onSuccess: win, onFailure: fail};
collection.isPushRequired(doc, options);
//or
collection.isPushRequired(0, options);
```

WL.JSONStore.load:

Gets data defined in load portion of the adapter. This is analogous to invoking an Adapter using **`WL.Client.invokeProcedure`** and calling the add method in **JSONStore** with the *{push : false}* flag with the data returned by the adapter.

Syntax

```
load([options]) Promise
```

Parameters

`options`
: Optional

  Options

Returns

`Promise:`

`OnSuccess`
: Number of documents stored

`onFailure`
: An error code.

Example

```
//See .init for context
WL.JSONStore.get('customers').load()
.then(function (res) {
//res => number of documents stored
})
.fail(function (errobject) {
```

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

```
WL.Logger.debug(errobject.toString());
});
//Deprecated Example:
customers.load(options)
```

WL.JSONStore.push:

Push the collection with an Adapter.

Syntax

```
push([options]) Promise
```

Description

Push the collection with an Adapter. For every Document marked requiring push, call the corresponding Adapter procedure linked to the collection. The Documents will be processed on the client by order of their last modification date. Error handling for **`push`** is more involved than other methods as a result of sending data to the server. Errors such as input validation or invalid states in the local collection will go to the promise's fail function, this class of error implies the push operation as a whole is unable to complete. Any documents that fail the actual process of being pushed to the server Adapter, such as a network error, server rejection or failure by the user written accept function will go to the promise's then or done function. To check the number of records to push, see "WL.JSONStore.getPushRequired" on page 229.

Parameters

`options`

Optional

Options or Array of Documents or Document

You can specify a document or an array of documents you want to push.

Returns

`Promise`

The success callback will be called when all the documents have been pushed. If you get an empty array it means everything was pushed, if something fails that array will contain error objects.

The following is deprecated behavior:

`onSuccess` called if it was successful or there where you records to push (you can check the number of records to push with the *getPushRequired* function),

`onFailure` returns an error code. The success callbacks are called once per document. If you try to push ten documents, your success callback might get called nine times and the failure callback once.

Example

```
//See .init and .add for context
WL.JSONStore.get('customers').push()
.then(function (res) {
//res => Empty array if everything worked or Array of error objects if something failed
})
.fail(function (errobject) {
//Normal errors: collection is closed, invalid data sent to push, ...
```

Experimental IBM Worklight online documentation
This document is provided for user's convenience.
In case of issues, refer to the online user documentation.

```
WL.Logger.debug(errobject.toString());
});
//Deprecated Example:
collection.push(options);
```

WL.JSONStore.pushRequiredCount:

Returns the number of documents not pushed. (Includes Documents marked as 'removed'.)

Syntax

```
pushRequiredCount([options]) Promise
```

Parameters

`options`
Optional

Options

Returns

`Promise:`
`OnSuccess`
Returns the number of documents only changed locally.

`onFailure`
Returns an error code.

Example

```
//See .init and .add for context
WL.JSONStore.get('customers').pushRequiredCount()
.then(function (res) {
//res => array of documents that need to be pushed
})
.fail(function (errobject) {
WL.Logger.debug(errobject.toString());
});
//Deprecated Example:
//Assumes that 1 document has been modified in the collection.
var win = function (data) {
console.log(data);
=> 1
};
var options = {onSuccess: win, onFailure: fail};
collection.pushRequiredCount(options);
```

WL.JSONStore.pushSelected:

Push the selected Documents

Syntax

```
pushSelected(doc, [options])
```

Description

Pushes only the selected Documents. See "WL.JSONStore.push" on page 234. The Document passed will not be sent to the Adapter (pushed) if it is not marked unpushed.

Deprecated, use push(doc)

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Parameters

`doc`
Document or Array of Documents

`options`
Optional

Options

Returns

See WL.JSONStore.push.

Example

```
var doc = {_id : 0, json: {fn : 'carlos', age : 99, active : false}};

collection.pushSelected(doc, options);
collection.pushSelected([doc], options);
```

WL.JSONStore.refresh:

Replaces a Document with another Document.

Syntax

```
refresh(doc, [options]) OnSuccess deprecated
```

Description

Replaces a Document with another Document just like **`replace`**, but it does not mark that change to push to the back end via an adapter. Compare with "WL.JSONStore.replace" on page 238.

Deprecated, use replace

Parameters

`doc`
Document or Array of Documents

`options`
Optional

Options

Returns

`OnSuccess`
Integer with the amount of Documents replaced.

`onFailure`
An error code.

Example

```
var doc = {_id : 0, json: {fn : 'carlos', age : 99, active : false}};
doc.json.age = 100;
collection.refresh(doc, options);
//or
collection.refresh([doc], options);
```

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

WL.JSONStore.remove:

Marks Documents as removed from a collection.

Syntax

```
remove(doc, [options]) Promise
```

Description

Marks one or more Documents as removed from a collection. Removed Documents are not returned by **`find`**, see "WL.JSONStore.find" on page 225 or **`count`**, see "WL.JSONStore.count" on page 222. The actual Documents are not deleted from the collection until successfully pushed. This action requires **`push`**, unless *{push: false}* is specified.

Parameters

`doc`
: Document or Array of Documents or Query or Integer.

  The Integer is an *_id*.

`options`
: Optional

  Options

Returns

`Promise:`
`OnSuccess`
: Integer with the number of documents removed.

`onFailure`
: An error code.

Example

```
//See .init and .add for context
var doc = {_id : 0, json: {fn : 'carlos', age : 99, active : false}};
WL.JSONStore.get('customers').remove(doc)
.then(function (res) {
//res => number of documents removed
})
.fail(function (errobject) {
WL.Logger.debug(errobject.toString());
});
//Deprecated Example:
var doc = {_id : 0, json: {fn : 'carlos', age : 99, active : false}};
collection.remove(doc, options); //Remove a Document
//or
collection.remove([doc], options); //Remove an Array of Documents
//or
collection.remove(1, options); //Remove by _id
//or
collection.remove({fn: 'carlos'}, options); //Remove all Documents that match {fn: 'carlos'}
```

WL.JSONStore.removeCollection:

Remove the collection.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Syntax

```
removeCollection([options]) Promise
```

Removes the collection locally, to use a collection with the same name you must call **WL.JSONStore.init**, see WL.JSONStore.init. This action does not call **`push`** before the operation. In order to remove specific documents see "WL.JSONStore.remove" on page 237 function.

Parameters

`options`
: Optional

  Options

Returns

`Promise:`

`OnSuccess`
: Boolean if the operation succeeded

`onFailure`
: An error code

Example

```
//See .init for context
WL.JSONStore.get('customers').removeCollection()
.then(function () {
// the collection was removed
})
.fail(function (errobject) {
WL.Logger.debug(errobject.toString());
});
//Deprecated Example:
collection.removeCollection(options);
```

WL.JSONStore.replace:

Replaces a Document with another Document.

Syntax

```
replace(doc, [options]) Promise
```

Description

Replaces a Document with another Document. This action requires **`push`**, unless *{push: false}* is specified.

Parameters

`doc`
: Document or Array of Documents

`options`
: Optional

  Options

Returns

`Promise:`

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues refer to the online user documentation.

OnSuccess
Integer with the number of Documents replaced.

onFailure
An error code.

Example

```
//See .init and .add for context
var doc = {_id : 0, json: {fn : 'carlos', age : 99, active : false}};
doc.json.age = 100;
WL.JSONStore.get('customers').replace(doc)
.then(function (res) {
//res => number of documents replaced
})
.fail(function (errobject) {
WL.Logger.debug(errobject.toString());
});
//Deprecated Example:
var doc = {_id : 0, json: {fn : 'carlos', age : 99, active : false}};
doc.json.age = 100;
collection.replace(doc, options);
//or
collection.replace([doc], options);
```

WL.JSONStore.store:

Load JSON objects into a collection as Documents.

Syntax

`store(data, [options]) OnSuccess deprecated`

Description

Used to initially load JSON objects into a collection as Documents. Stores data marked as **pushed**, see "WL.JSONStore.add" on page 219 to store Documents as **unpushed**.

Deprecated, use add

Parameters

data
Object or Array of Objects.

Data to be added the collection.

options
Optional

Options

Additional options:**additionalSearchFields : {}**

Returns

OnSuccess
Integer with the amount of data stored.

onFailure
An error code.

Experimental IBM Worklight offline user documentation
This document is provided "as is"
In case of issues, refer to the online user documentation.

Example

```
//Store an Object
var data = {fn: 'carlos', age: 99, active: false};
collection.store(data, options);
//Store Multiple Objects
var dataArray = [ {fn: 'Tim', age: 88, active: true},
{fn: 'Jeff', age: 77, active: false} ];
collection.store(dataArray, options);
//Store Multuple Objects without the Array
var data1 = dataArray[0];
var data2 = dataArray[1];
collection.store(data1, {onSuccess: function(){
collection.store(data2, {onSuccess: win});
}});
```

WL.JSONStore.toString:

Prints the contents of the collection by using **WL.Logger.debug** asynchronously.

Syntax

```
toString ( limit , offset )
```

Parameters

limit Integer
How many documents to print. 0 for none, if it is missing it prints up to the first 100 documents.

offset Integer
How many documents to skip. Requires a valid limit.

Example

```
collection.toString() // Print up to the first 100 documents
collection.toString(10) //Prints up to the first 10 documents
collection.toString(10,10) //Prints up to the first 10 documents after the first 10
collection.toString(0) //Prints no documents, only the collection metadata
    (name, searchFields and adapter)

//Equivalent to:
collection.findAll().done(function(data){console.log(JSON.stringify(data))})
```

WL.JSONStore.usePassword:

Deprecated. Sets a password.

Syntax

```
usePassword(pwd) Boolean deprecated static
```

Description

Sets the password that is used to generate keys to encrypt date that is stored locally on the device. This function is deprecated.

Deprecated, use WL.JSONStore.init

Parameters

pwd
String

String containing the password

Experimental IBM Worklight offline user documentation
This content is provided "as is".
In case of issues, refer to the online user documentation.

Returns

Boolean
: true if the password is a string, false otherwise.

Example

```
var pwd = prompt('What is your password?');
WL.JSONStore.usePassword(pwd);
```

The WL.Logger object

The `Logger` object prints log messages to the log for the environment.

- In mobile apps, messages are printed to a log file displayed in the mobile OS SDK.
- In web environments, they are printed to the browser log.
- In desktop environments, they are printed to the applicable debug console of each environment. See details in the following section.

Setting Logger Windows for Desktop Environments

Adobe AIR

Include the `AIRIntrospector.js` file in your project and call it in your HTML file.

Apple OS X Dashboard

Open the application at `/Users/`*`user name`*`/Library/Widgets/`*`widget name`* with Dashcode and run it. The log is displayed in the Run Log View. You can also view the debug messages in the system log in the Console.

Windows 8

No configuration is required. Debug messages are displayed in the Microsoft Visual Studio 2012 console.

Windows 7 and Vista

Disable the following check boxes in Internet Explorer: **Internet Options** > **Advanced** > **Browsing**

- **Disable script debugging (Internet Explorer)**
- **Disable script debugging (Other)**

In your code, add a line `debugger();`. Run the application. When the interrupter hits that line, you are asked to open a debug tool.

The WL.Logger.debug and WL.Logger.error methods:

These methods output a debug or error message. The developers that are creating applications for all platforms and wanting to write common code must use WL.Logger.debug and WL.Logger.error methods.

Syntax

```
WL.Logger.debug(msg, ex)
WL.Logger.error(msg, ex)
```

Description

The WL.Logger.debug and WL.Logger.error methods output a specified debug or error message to the environment log.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

For iPhone and iPad, these methods print the message to the debugger console of Xcode, the development environment for iPhone and iPad.

For Android, these methods print the message to the Android LogCat, accessible in the Android development environment. Error messages are displayed in red; debug messages are displayed in the default log line color.

Android developers can call two more functions **info** and **warn**:

```
WL.Logger.info(msg, ex);
WL.Logger.warn(msg, ex);
```

and they accept the same parameters as **debug** and **error**. LogCat differentiates all message levels by colors:

- debug = blue
- error = red
- information = green
- warning = yellow

and also allows filtering of messages by level. Therefore, calling these methods has some additional value for developers.

The **log** method is a helper that must not be called by developers. For example, it does not work for Windows Phone environments.

For BlackBerry 6 and 7, these methods print the message to the BlackBerry event log. The event log can be viewed on the device and in the BlackBerry Eclipse simulator and debugger by pressing the key sequence Alt+LGLG.

To disable logging, include `enableLogger: false` in the object wlInitOptions in the `initOptions.js` file.

```
var wlInitOptions = {
  // # Should application produce logs
  // # Default value is true
  enableLogger: false
}
```

Parameters

Table 47. Parameters for the WL.Logger.debug and WL.Logger.error methods

Parameter	Description
msg	Mandatory. The string to be displayed in the logger window.
ex	Optional. A JavaScript exception. If specified, the file name and line number are appended to the message.

Return Value

None.

```
try {
  myNeverDeclaredFunction();
} catch (ex) {
  WL.Logger.error("This method was never declared", ex);
}
```

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

WL.Logger.send:

Send any logs that are collected up to this point in time to the `WLClientLogReceiver` adapter.

Syntax

```
WL.Logger.send();
```

Description

Send any logs that are collected up to this point in time to the `WLClientLogReceiver` adapter.

Parameters

None

Return Value

None

WL.Logger.setNativeOptions:

Set native options for logging.

Syntax

```
WL.Logger.setNativeOptions(object);
```

Description

Set native options for logging.

Parameters

Parameter	Description

`object`	Optional. An object that can contain any of the following key/value pairs:
	`maxFileSize: integer`
	Minimum value is 10000 (bytes)
	`level: String`
	Any of the following values: `debug`, `log`, `info`, `warn`, `error`
	`capture: boolean`
	`true` or `false`

Return Value

None

Mobile push notification methods

IBM Worklight provides a number of methods for supported push notification mechanisms.

Push notifications are supported on iOS, Android, and Windows Phone 8 devices.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

SMS push notifications are supported on iOS, Android, Windows Phone 7.5, Windows Phone 8, and BlackBerry devices that support SMS functions.

Note: Subscription, and unsubscription, to SMS notifications can also be performed by making HTTP GET requests to the subscribe SMS servlet. The subscribe SMS servlet can be used for SMS subscriptions without the requirement for a user to have an app installed on their device. See "Subscribe SMS servlet" on page 398 for information about configuration.

WL.Client.Push.isPushSMSSupported:

Checks whether SMS push notifications are supported.

Syntax

```
WL.Client.Push.isPushSMSSupported();
```

Description

Returns **`true`** if the IBM Worklight JavaScript API supports SMS push notifications in the current environment.

Parameters

None.

WL.Client.Push.isPushSupported:

Checks whether push notification is supported.

Syntax

```
WL.Client.Push.isPushSupported();
```

Description

Returns **`true`** if the IBM Worklight JavaScript API supports push notifications in the current environment.

Parameters

None.

WL.Client.Push.isSMSSubscribed:

Checks whether current user is subscribed to an SMS event source.

Syntax

```
WL.Client.Push.isSMSSubscribed(alias)
```

Description

Returns whether the currently logged-in user is subscribed to the SMS event source alias

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

Parameters

Table 48. WL.Client.Push.isSMSSubscribed parameters

Parameter	Description
alias	Mandatory string. The event source alias.

WL.Client.Push.isSubscribed:

Checks whether current user is subscribed to an event source.

Syntax

```
WL.Client.Push.isSubscribed(alias)
```

Description

Returns whether the currently logged-in user is subscribed to the specified event source alias

Parameters

Table 49. WL.Client.Push.isSubscribed parameters

Parameter	Description
alias	Mandatory string. The event source alias.

WL.Client.Push.onReadyToSubscribe:

A callback function to notify that a device is ready to subscribe.

Syntax

```
WL.Client.Push.onReadyToSubscribe = { /*callback function code*/ };
```

Description

Implement this callback function to be notified when the device is ready for subscribing to push notifications. You must declare it outside any function.

Parameters

None.

```
WL.Client.Push.onReadyToSubscribe= function () {
// You can enable the Subscribe button here or call WL.Client.Push.subscribe().
// This callback is useful in case your app needs to call subscribe() upon startup.

WL.Client.Push.registerEventSourceCallback ('myAlias', 'myAdapter', 'myEventSource', notificationA
}

function notificationArrived(props, payload){
alert("Provider notification data: " + Object.toJSON(props));
alert("Application notification data: " + Object.toJSON(payload));
}
```

WL.Client.Push.registerEventSourceCallback:

Registers a callback method that is called whenever a notification arrives from the specified event source.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Syntax

```
WL.Client.Push.registerEventSourceCallback (alias, adapter, eventSource, callback);
```

Description

iOS and Android
: Registers a callback method that is called whenever a notification arrives from the specified event source. If the notification arrives while the application is not running, the mobile OS starts the application at the specified callback.

Windows Phone 8
: Registers a callback method that is called whenever a raw notification or a toast notification arrives and the application is running. If the notification arrives when the application is not running, then the callback method is not called. This behavior is defined in the Microsoft OS and cannot be changed.

Parameters

Table 50. WL.Client.Push.registerEventSourceCallback parameters

Parameter	Description
`alias`	Mandatory string. A short ID that is used to identify the event source when the push notification arrives. Because notification text is usually limited in length, providing a short alias, rather than the entire adapter and event source names, can free more space in the notification text.
`adapter`	Mandatory string. The name of the adapter that contains the event source.
`eventSource`	Mandatory string. The name of the event source.
`callback`	Mandatory function. The function that is called if a notification arrives. The function receives two parameters when invoked: **props** A `JSON` block, containing the notification properties from the platform **payload** A `JSON` block, containing other data that is sent from the IBM Worklight Server

Experimental IBM Worklight offline user documentation.
This document is provided "as is".
In case of issues, refer to the online user documentation.

WL.Client.Push.subscribe:

Subscribe to an event source.

Syntax

```
WL.Client.Push.subscribe(alias, options)
```

Description

Subscribes the user to the event source with the specified alias.

Parameters

Table 51. WL.Client.Push.subscribe parameters

Parameter	Description
alias	Mandatory string. The event source alias, as defined in the invocation of `WL.Client.Push.onReadyToSubscribe`.
options	Optional JSON block, containing: • Custom subscription parameters that are supported by the event source in the adapter • `onFailure`: a JavaScript function, called when registration for the notification service failed. The function receives one string parameter with the failure description. The default value is a function that prints the failure reason to the log. • `onSuccess`: JavaScript function, called when the subscription was successful. Default value is an empty function.

```
if (WL.Client.Push.isPushSupported()){
WL.Client.Push.subscribe( 'myAlias', {foo: 'bar', onFailure : notificationSubscriptionError});
}

function notificationSubscriptionError(error) {
alert("Error registering for push notifications. " + error);
}
```

WL.Client.Push.subscribeSMS:

Subscribe to an SMS event source.

Syntax

```
WL.Client.Push.subscribeSMS(alias, adapterName, eventSource, phoneNumber, options)
```

Description

Subscribes the user to the SMS event source with the specified alias.

Parameters

Table 52. WL.Client.Push.subscribeSMS parameters

Parameter	Description
alias	Mandatory string. A short ID defining the event source.
adapterName	Mandatory String. The name of the adapter that sets up the event source and communicates with the Worklight server.
eventSource	Mandatory String. The name of the event source.

Experimental IBM Worklight online user documentation. This document is provided in case of issues, refer to the online user documentation.

Table 52. WL.Client.Push.subscribeSMS parameters (continued)

Parameter	Description
phoneNumber	Mandatory string. User phone number to which SMS notifications are sent. The phone number is provided by the user and can contain digits (0-9), plus sign (+), minus sign (-), and space (␣) characters only.
options	Optional JSON block, containing: • Custom subscription parameters that are supported by the event source in the adapter • `onFailure`: a JavaScript function, called when registration for the notification service failed. The function receives one string parameter with the failure description. The default value is a function that prints the failure reason to the log. • `onSuccess`: a JavaScript function, called when the subscription was successful. The default value is an empty function.

```
if (WL.Client.Push.isPushSMSSupported()){
 WL.Client.Push.subscribeSMS( "myAlias","SMSAdapter","SMSEventSource", "1234567890",
 {onSuccess: notificationSubscriptionSuccess,
 onFailure : notificationSubscriptionError
 });
}

function notificationSubscriptionSuccess(){
 alert("Registered for SMS push notification");
}

function notificationSubscriptionError(error) {
 alert("Error registering for SMS push notifications. " + error);
}
```

WL.Client.Push.unsubscribe:

Unsubscribe from an event source.

Syntax

`WL.Client.Push.unsubscribe(alias, options)`

Description

Sends the device token (obtained by `subscribe`) and the event source name to the IBM Worklight Server.

Parameters

Table 53. WL.Client.Push.unsubscribe parameters

Parameter	Description
alias	Mandatory string. The event source alias, as defined in the invocation of `WL.Client.Push.onReadyToSubscribe`.

Experimental IBM Worklight online user documentation. This document is provided as is. In case of issues, refer to the online user documentation.

Table 53. WL.Client.Push.unsubscribe parameters (continued)

Parameter	Description
options	Optional JSON block, containing: • Custom subscription parameters that are supported by the event source in the adapter • `onFailure`: a JavaScript function, called when registration for the notification service failed. The function receives one string parameter with the failure description. The default value is a function that prints the failure reason to the log. • `onSuccess`: a JavaScript function, called when the subscription was successful. Default value is an empty function.

WL.Client.Push.unsubscribeSMS:

Unsubscribe from an SMS event source.

Syntax

```
WL.Client.Push.unsubscribeSMS(alias, options)
```

Description

Unsubscribes the user from the SMS event source with the specified alias.

Parameters

Table 54. WL.Client.Push.unsubscribeSMS parameters

Parameter	Description
alias	Mandatory string. The alias defined when subscribing.
options	Optional JSON block, containing: • `onFailure`: a JavaScript function, called when communication with the IBM Worklight Server did not succeed, or when there are no subscriptions to unsubscribe. Default value is a function that logs the failure. • `onSuccess`: a JavaScript function, called when unsubscription was successful. Default value is a function that logs the success.

The `options` object

The `options` object contains properties that are common to all methods. It is used in all asynchronous calls to the Worklight Server

Pass an `options` object for all asynchronous calls to IBM Worklight Server. The `options` object contains properties that are common to all methods. Sometimes it is augmented by properties that are only applicable to specific methods. These additional properties are detailed as part of the description of the specific methods.

Experimental IBM Worklight offline user documentation
This document is provided "as is"
In case of issues, refer to the online user documentation.

The common properties of the `options` object are as follows:

```
options = {
  onSuccess: success-handler-function(response),
  onFailure: failure-handler-function(response),
  invocationContext: invocation-context
};
```

The meaning of each property is as follows:

Table 55. `options` object properties

Property	Description
`onSuccess`	Optional. The function to be invoked on successful completion of the asynchronous call. The syntax of the `onSuccess` function is: `success-handler-`*`function`*`(`*`response`*`)` where *response* is an object that contains at a minimum the following property: **`invocationContext`** The `invocationContext` object that was originally passed to the Worklight Server in the `options` object, or `undefined` if no `invocationContext` object was passed. **`status`** The HTTP response status **Note:** For methods for which the *response* object contains additional properties, these properties are detailed as part of the description of the specific method.
`onFailure`	Optional. The function to be invoked when the asynchronous call fails. Such failures include both server-side errors, and client-side errors that occurred during asynchronous calls, such as server connection failure or timed out calls. **Note:** The function is not called for client-side errors that stop the execution by throwing an exception. The syntax of the `onFailure` function is: `failure-handler-`*`function`*`(`*`response`*`)` where *response* is an object that contains the following properties: **`invocationContext`** The `invocationContext` object that was originally passed to the Worklight Server in the `options` object, or `undefined` if no `invocationContext` object was passed. **`errorCode`** An error code string. All error codes that can be returned are defined as constants in the `WL.ErrorCode` object in the `worklight.js` file. **`errorMsg`** An error message that is provided by the Worklight Server. This message is for the developer's use only, and should not be displayed to the user. It will not be translated to the user's language. **`status`** The HTTP response status **Note:** For methods for which the *response* object contains additional properties, these properties are detailed as part of the description of the specific method.

Experimental IBM Worklight user documentation
This documentation is provided "as is."
In case of issues refer to the online user documentation.

Table 55. `options` object properties (continued)

Property	Description
`invocationContext`	Optional. An object that is returned to the success and failure handlers. The `invocationContext` object is used to preserve the context of the calling asynchronous service upon returning from the service. For example, the invokeProcedure method might be called successively, using the same success handler. The success handler needs to be able to identify which call to invokeProcedure is being handled. One solution is to implement the invocationContext object as an integer, and increment its value by one for each call of invokeProcedure. When it invokes the success handler, Worklight passes it the invocationContext object of the `options` object associated with the invokeProcedure method. The value of the invocationContext object can be used to identify the call to invokeProcedure with which the results that are being handled are associated.

Options Menu and Application Bar API

IBM Worklight supplies a number of methods for manipulating the Android options menu and the Windows Phone 7.5, Windows Phone 8, and Windows 8 apps application bar.

This section applies to Android, Windows Phone 7.5, Windows Phone 8, and Windows 8 apps only.

The Android options menu and the Windows Phone 7.5, Windows Phone 8, and Windows 8 apps application bar are accessible by pressing **Menu** on the device. IBM Worklight provides a client-side API for managing the menu and application bar.

Note: If your application targets Android 3.0 (API level 11) or higher, WL.OptionsMenu might have no effect, depending on the device. For more information, see http://developer.android.com/guide/topics/ui/menus.html#options-menu.

WL.Item Methods:

Change an item's callback function, title, or icon, or enables or disables an item.

Syntax

```
WL.Item.setTitle (title)
WL.Item.setImagePath (imagePath)
WL.Item.setEnabled (enabled)
```

Description

Changes the item's callback function, title, or icon, or enables or disables it.

Note: You cannot instantiate a new WL.Item object; you can receive one as a result of calling WL.OptionsMenu.getItem().

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues refer to the online user documentation.

Parameters

Table 56. WL.Item method parameters

Parameter	Description
`callbackFunction`	Mandatory function. The callback function that is invoked when the user selects the item.
`title`	Mandatory string. The title of the item.
`iconPath`	Mandatory string. The path to the icon. See WL.OptionsMenu.addItem for an explanation of the icon path and format for Android, Windows Phone 7.5, Windows Phone 8, and Windows 8.
`enabled`	Mandatory Boolean. Defines whether the item is enabled or disabled.

Return Value

None

This example disables the first item.

```
var itemOne = WL.OptionsMenu.getItem('first');
itemOne.setEnabled(false);
```

WL.OptionsMenu.addItem:

Adds an item to the options menu or application bar.

Syntax

```
WL.OptionsMenu.addItem(id, callbackFunction, title, options)
```

Description

Adds an item to the options menu or application bar. Can be called only after the menu is initialized. Items are placed in the menu in the order in which they are added. If you add an item with an existing ID, the new item replaces the existing one.

Parameters

Table 57. WL.OptionsMenu.addItem parameters

Parameter	Description
`id`	Mandatory string. Identifies the item.
`callbackFunction`	Mandatory JavaScript function. The callback function that is invoked when the user selects the item in the options menu.
`title`	Mandatory string. The title of the item.

Experimental IBM Worklight offline user documentation
This document is provided "as is"
In case of issues, refer to the online user documentation.

Table 57. WL.OptionsMenu.addItem parameters (continued)

Parameter	Description
`options`	The **`options`** parameter is mandatory, and the **`image`** property within it is also mandatory. Contains the following fields: **`image`** For Android, this field contains the name of the resource that contains the icon image for the item. For Windows Phone 7.5, Windows Phone 8, and Windows 8, this field contains the path to the icon image for the item. For Android, the image is located under the Android `res/drawable*` folders of the application. You can provide multiple images and place them in the `drawable*` folders that belong to the device densities your application supports. For Windows Phone 8, the path starts from the folder `/nativeResources/applicationBar`. Do not explicitly mention this folder within the path. For Windows Phone 7.5 and Windows 8, the path starts from the folder `/Resources/applicationBar`. Do not explicitly mention this folder within the path. The same set of images can be used for Android, Windows Phone 7.5, Windows Phone 8, and Windows 8. For Android, the image size depends on the density of the device. See the Android Options Menu documentation for details. For Windows Phone 7.5, Windows Phone 8, and Windows 8, these images are 48 pixels by 48 pixels and have a white foreground on a transparent background that uses an alpha channel. The Application Bar colorizes the icon according to the current style settings and colored icons can cause this effect to display unpredictably. **`Enabled`** Optional Boolean. Defines whether the item is enabled or disabled.

Experimental IBM Worklight offline user documentation
This document is provided as is
In case of issues, refer to the online user documentation.

Return Value

None

```
// Android
WL.OptionsMenu.addItem("first", function(){alert("hello one")}, 'one', {image: 'one', enabled: true}

// Windows Phone 7.5 and Windows Phone 8
WL.OptionsMenu.addItem("first", function(){alert("hello one")}, one', {image: one.png', enabled: tru

// Windows 8
WL.OptionsMenu.addItem("first", function(){alert("hello one")}, one', {image: one.png', enabled: tru
```

WL.OptionsMenu.getItem:

Returns an item.

Syntax

```
WL.OptionsMenu.getItem(id)
```

Description

Returns the item with the specified ID. You can use `Item` methods to change the properties of the item.

Parameters

Table 58. WL.OptionsMenu.getItem parameters

Parameter	Description
id	Mandatory string. The ID of the required item.

Return Value

An Item object. If the specified ID is not found, the method returns null.

```
var itemOne = WL.OptionsMenu.getItem('first');
```

WL.OptionsMenu.init:

Initializes and enables the options menu or application bar.

Syntax

```
WL.OptionsMenu.init()
```

Description

Initializes the options menu or application bar and enables it. Must be called before it is used. On Windows Phone 7.5 and Windows Phone 8, the default opacity of the application bar is 1.0 (opaque).

Parameters

A JSON block with the following properties:

Table 59. WL.OptionsMenu.init parameters

Property	Description
msg	Mandatory. The string to be displayed in the logger window.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

Table 59. WL.OptionsMenu.init parameters (continued)

Property	Description
ex	Optional. A JavaScript exception. If specified, the file name and line number are appended to the message.

Return Value

None

WL.OptionsMenu.init({opacity: "0.5"});

WL.OptionsMenu.isEnabled:

Check whether the options menu or application bar is enabled.

Syntax

```
WL.OptionsMenu.isEnabled (callback)
```

Description

Returns whether the options menu or application bar is enabled. Can be called only after the menu is initialized.

Parameters

@callback, a callback method that accepts the state **`enabled`** as a parameter.

Return Value

- In Android environments: `true` if the menu is enabled; `false` if it is not.
- In Windows Phone 7.5 and Windows Phone 8 environments: none. If the callback is null or undefined, the method fails and sends a message to the debugger console.

```
WL.OptionsMenu.isEnabled(isEnabledCallback);

function isEnabledCallback(enabled) {
      if (enabled) {
            // do something
      }
}
```

WL.OptionsMenu.isVisible:

Check whether the options menu or application bar is visible.

Syntax

```
WL.OptionsMenu.isVisible (callback)
```

Description

Returns whether the options menu or application bar is visible. Can be called only after the menu is initialized.

Experimental IBM Worklight online user documentation
This document is provided "as is".
In case of issues refer to the online user documentation.

Parameters

@callback, a callback method that accepts the state **`visible`** as a parameter.

Return Value

- In Android environments: `true` if the menu is visible; `false` if it is not.
- In Windows Phone 7.5 and Windows Phone 8 environments: none. If the callback is null or undefined, the method fails and sends a message to the debugger console.

```
WL.OptionsMenu.isVisible(isVisibleCallback);

function isVisibleCallback(visible) {
       if (visible) {
               // do something
       }
}
```

WL.OptionsMenu.removeItem:

Remove an item from the options menu or application bar.

Syntax

```
WL.OptionsMenu.removeItem(id)
```

Description

Removes the item with the indicated ID from the options menu or application bar. Can be called only after the menu is initialized.

If no item is found with the specified ID, nothing happens.

Parameters

Table 60. WL.OptionsMenu.removeItem parameters

Parameter	Description
`id`	Mandatory string. Identifies the item to be removed.

Return Value

None

```
WL.OptionsMenu.removeItem("first");
```

WL.OptionsMenu.removeItems:

Removes all items from the options menu or application bar.

Syntax

```
WL.OptionsMenu.removeItems()
```

Description

Removes all items from the options menu or application bar. Can be called only after the menu is initialized.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

Parameters

None

Return Value

None

```
WL.OptionsMenu.removeItems();
```

WL.OptionsMenu.setEnabled:

Enable or disable the options menu or application bar.

Syntax

```
WL.OptionsMenu.setEnabled(isEnabled)
```

Description

This method enables or disables the options menu or application bar. When the menu or bar is disabled, it might still be visible, but all its items are disabled. The function disables all the items but does not change the enabled state of each Item.

Parameters

Table 61. `WL.OptionsMenu.setEnabled` parameters

Parameter	Description
`isEnabled`	Mandatory Boolean. `true`: Enable the menu `false`: Disable the menu

Return Value

None.

```
WL.OptionsMenu.setEnabled(false);
```

WL.OptionsMenu.setOpacity:

Sets the opacity of the Windows Phone 7.5 and Windows Phone 8 application bar.

Syntax

```
WL.OptionsMenu.setOpacity(number)
```

Description

Sets the value for the application bar opacity.

Note: This method is applicable only for the Windows Phone 7.5 and Windows Phone 8 application bar.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues refer to the online user documentation.

Parameters

Table 62. `WL.OptionsMenu.setOpacity` parameters

Parameter	Description
`number`	Mandatory. Float between 0.0 and 1.0. 0.0 is transparent; 1.0 is opaque. When the application bar is not opaque, Windows Phone 7.5 and Windows Phone 8 devices display it as an overlay on the application.

Return Value

None.

```
WL.OptionsMenu.setOpacity(0.0);
```

WL.OptionsMenu.setVisible:

Make the options menu or application bar visible or invisible.

Syntax

```
WL.OptionsMenu.setVisible(isVisible)
```

Description

Determines whether the options menu or application bar is visible. Can be called only after the options menu is initialized.

Note: This method is not supported on Windows 8.

Parameters

Table 63. `WL.OptionsMenu.setVisible` parameters

Parameter	Description
`isVisible`	Mandatory Boolean. `true`: Makes the menu visible on pressing **Menu** `false`: Makes the menu invisible

Return Value

None

```
WL.OptionsMenu.setVisible(true);
```

WL.SimpleDialog

The simple dialog box object

`WL.SimpleDialog` implements a common API for showing a dialog with buttons for the application. The implementation depends on the environment. On iPhone, Android, BlackBerry 6 and 7, and Windows 8, WL.SimpleDialog opens a native dialog box. In other environments, it opens an HTML dialog box.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

`WL.SimpleDialog` supports up to three buttons.

WL.SimpleDialog.show:

Display a dialog box.

Syntax

```
WL.SimpleDialog.show(title, text, buttons, options)
```

Description

Displays a dialog box.

Note: the dialog is displayed without blocking the JavaScript thread.

Parameters

Table 64. WL.SimpleDialog.show parameters

Parameter	Description
`title`	Mandatory string. The title of the dialog box.
`text`	Mandatory string. The text to show in the dialog box.
`buttons`	Mandatory array of JSON objects, each corresponding to a button. The array must have at least one item and no more than three items. Each array item contains the following properties: **text** Mandatory string. The text of the button. **handler** Optional function. The function that is invoked when the button is pressed.

`options`	Ignored on iPhone and Android.
	Optional. An object of the following form:
	`{`
	`title: string,`
	`text: string,`
	`}`

Returned Values

None.

Example

Example

```
WL.SimpleDialog.show(
"My Title", "My Text",
[{text: "First Button", handler: function() {WL.Logger.debug("First button pressed"); }
}]
)
```

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Splitting Your Code between HTML Pages

WL.Page and WL.Fragment APIs are deprecated. To split your code between pages, use the fragment implementation of JavaScript frameworks such as jQuery Mobile, Sencha, and Dojo Mobile.

Fragments

Fragments are files that contain html tags that can be appended to a parent element. Fragments can include JavaScript files by adding a `<script>` tag and style sheets by adding a `<link>` tag.

For example:

```
<link href="css/page1.css" />
<script src="js/page1.js" type="text/javascript"></script>
<div id="page1div">
     This is page1
</div>
```

Inline JavaScript and CSS

Inline JavaScript and CSS are not supported in fragments. For example, the method `test` might not work in some environments:

```
<script type="text/javascript">
function test(){alert("test");}
</script>

<div id="one">
Hello Page 1!
</div>
```

The same problem might happen with inline style definitions:

```
<style type="text/css">
background-color:red;
</style>

<div id="one">
Hello Page 1!
</div>
```

Structure of <link> and <script> tags in Internet Explorer

In Internet Explorer, <link> and <script> tags are identified using regular expressions. They are therefore vulnerable to white space within the markup. Be sure to format these tags as in the example to ensure that they are recognized and handled at run time.

```
<link href="css/page1.css" />
<script src="js/page1.js" type="text/javascript"></script>
```

When testing on Internet Explorer, you can check in the logger to verify that all resources were loaded.

Support for Fragments for Windows Desktop Gadgets

Fragments are not supported on Windows 7 and Windows Vista Desktop Gadgets.

The use of Windows 7 and Vista gadgets is deprecated in Worklight version 5.0.5. Support might be removed in any future version.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

The onUnload Callback

When you use fragments and pages, you are responsible for deleting from the HTML DOM all objects that were created by scripts that are embedded in your fragment. To do so, you must implement the `onUnload` callback method and clean up your DOM there.

When you use fragments as pages, define all pages on a namespace object, which is possible because no two pages are loaded simultaneously, and clean that object in the `onUnload` method.

Example:

In the code that loads the page:

```
WL.Page.load("page1.fhtml", {
onComplete : function(){CurPage.pageFunction();},
onUnload: function(){CurPage.onUnload();}
});

// Define all JavaScript objects on a namespace object:
var CurPage = CurPage ? CurPage : {};

// Note that the onUnload callback is implemented once, not per fragment!
CurPage.onUnload = function() {
    for (var att in CurPage){
        delete CurPage[att];
    }
}
```

In the `page1.js` file:

```
// Implement the specific callbacks
CurPage.pageFunction = function() {
    WL.Logger.debug("pageFunction from page1.js called");
}
```

Note: The namespace (**`CurPage`** in the example) cannot be deleted, only the objects and methods that are defined in it.

WL.Fragment.load:

Deprecated. Replaces the content of a DOM element parent

Syntax

```
WL.Fragment.load(fragmentPath, parent, options)
```

Description

Replaces the content of the DOM element parent with the Fragment referred to by **`fragmentPath`**.

Parameters

Parameter	Description
`fragmentPath`	Mandatory. String. The path of the file that contains the HTML fragment, relative to the main HTML file.
`parent`	Mandatory. Object. The parent element which contains the fragment.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Parameter	Description
options	Mandatory. A hash object supporting the following option: • onComplete: Function. Callback function, called after the fragment is appended to the parent and all its JavaScript and CSS files are loaded. • onUnload: Function. Callback function, called when the fragment is unloaded. You are responsible to delete all objects that were created by the fragment. See The onUnload Callback for more details about this callback function.

Example

```
WL.Fragment.load ("./list.html", document.getElementById("listContainer"), {onComplete: onListLoaded
```

Class WL.Page:

Deprecated. A Page is similar to a Fragment, but has additional properties.

A page is a fragment that is appended to the HTML `<div>` element with the pagePort ID in the main HTML file. You can declare the pagePort element anywhere you want within the `content` element of the app.

`Page` supports back navigation, which is not supported by `Fragment`.

WL.Page.back:

Deprecated. Loads the previous page into the main HTML pagePort element.

Syntax

```
WL.Page.back(options)
```

Description

Loads the previous page into the main HTML pagePort element. Does nothing if the page history is empty. You can use the hasBack method to check whether the page history is empty.

Parameters

Table 65. `WL.Page.back` parameters

Parameter	Description
options	A hash object supporting the options of the WL.Page.load method, and in addition: pagesBack: Optional. Positive integer. The number of pages to go back. The default is 1. If the specified number exceeds the number of pages in the page stack, the first page in the stack is loaded.

Experimental IBM Worklight online user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

Example

```
WL.Page.back({onComplete: onPageLoaded, pagesBack: 2});
```

WL.Page.hasBack:

Deprecated. Returns `true` if the history stack contains at least one item.

Syntax

```
WL.Page.hasBack()
```

Parameters

None.

Returned Values

- `true` if the history stack contains at least one item
- `false` otherwise

Example

```
If (WL.Page.hasBack())
WL.Page.back({onComplete: onPageLoaded});
```

WL.Page.load:

Deprecated. Replaces the content of the DOM element pagePort

Syntax

```
WL.Page.load(pagePath, options)
```

Description

Replaces the content of the DOM element `pagePort` with the `fragment` referred to by `pagePath`.

Parameters

Table 66. WL.Page.load parameters

Parameter	Description
`pagePath`	String. The path of the file that contains the HTML fragment that implement the page, relative to the main HTML file.
`options`	A hash object supporting the following option: • `onComplete`: Function. Callback function, called after the page is appended to the parent and all its JavaScript and CSS files are loaded. • `onUnload`: Function. Callback function, called when the page is unloaded. You are responsible for deleting all objects that were created by the page. See the onUnload callback for more details about this callback function.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Example

```
WL.Page.load ("./page2.html", {onComplete: CurPage.onPageLoaded, onUnload: CurPage.onPageUnloaded});
```

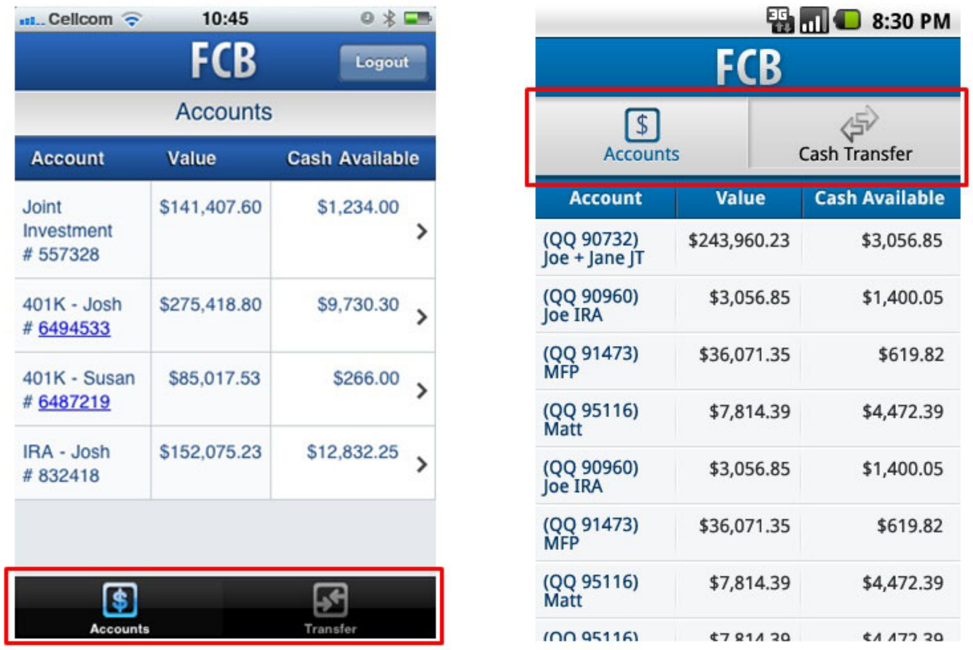
Tab Bar API

IBM Worklight provides an API for managing the tab bar on Android and iPhone.

This section applies to Android and iPhone only.

The Android and iPhone tab bars are graphical elements which look and work very much like the tab bars of regular web or desktop applications. IBM Worklight provides a client-side API for managing the tab bar. On iPhone, this API serves as a proxy to the native iPhone tab bar object; on Android, it is implemented as an HTML element.

The following example depicts an iPhone tab bar (on the left) and an Android tab bar (on the right) The tab bars are enclosed in red rectangles.



Account	Value	Cash Available
Joint Investment # 557328	$141,407.60	$1,234.00
401K - Josh # 6494533	$275,418.80	$9,730.30
401K - Susan # 6487219	$85,017.53	$266.00
IRA - Josh # 832418	$152,075.23	$12,832.25

Account	Value	Cash Available
(QQ 90732) Joe + Jane JT	$243,960.23	$3,056.85
(QQ 90960) Joe IRA	$3,056.85	$1,400.05
(QQ 91473) MFP	$36,071.35	$619.82
(QQ 95116) Matt	$7,814.39	$4,472.39
(QQ 90960) Joe IRA	$3,056.85	$1,400.05
(QQ 91473) MFP	$36,071.35	$619.82
(QQ 95116) Matt	$7,814.39	$4,472.39
(QQ 95116)	$7,814.39	$4,472.39

Figure 33. iPhone tab bar (left) and Android tab bar (right)

WL.TabBar.addItem:

Add an item to the tab bar.

Syntax

```
WL.TabBar.addItem(id, callback, title, options)
```

Description

Adds an item to the tab bar. Can be called only after the tab bar is initialized. Items are displayed in the tab bar according to the order in which they were added to the tab bar.

Parameters

Table 67. `WL.TabBar.addItem` parameters

Parameter	Description
`id`	Mandatory string. Identifies the tab.
`callback`	Mandatory function. The callback function that is invoked when the user touches the tab.
`title`	Mandatory string. The title of the tab. If `null` is passed, no title is displayed.
`options`	Options for customizing the tab item. • On iPhone: – `image`: String. File name or path relative to the application root directory, with a PNG image for the tab or an internal identifier for standard tabs. See the list of standard tabs in the next section. – `badge`: String. A string to display in the optional circular badge on the item; if null or unspecified, no badge is displayed. • On Android: – `image`: String. File name or path relative to the application root directory, with a PNG image for the tab in unselected mode. – `ImageSelected`: String. File name with an image for the tab in selected mode.

On iPhone, if the supplied image name is one of the labels in the following list, this method constructs a tab button by using the standard system buttons. If you use one of the system images, then the title you supply is ignored.

- `tabButton:More`
- `tabButton:Favorites`
- `tabButton:Featured`
- `tabButton:TopRated`
- `tabButton:Recents`
- `tabButton:Contacts`
- `tabButton:History`
- `tabButton:Bookmarks`
- `tabButton:Search`
- `tabButton:Downloads`
- `tabButton:MostRecent`
- `tabButton:MostViewed`

Experimental IBM Worklight offline user documentation
This document is provided as-is.
In case of issues refer to the online documentation.

Return Value

A WL.TabBarItem object.

iPhone

```
function selectCredit(){
alert("the CREDIT tab was selected!");
}
var creditTab = WL.TabBar.addItem("CREDIT", selectCredit, "Visa", {image:"images/credit.png", badge:
```

Android

```
var tabFeeds = WL.TabBar.addItem (
'tab2',
function(){worklightStarterApplication.selectTab('feedsWrapper'); },
"Engadget Feeds",
{image:"images/feed.png", imageSelected:"images/feed.png"});
```

WL.TabBar.init:

Initialize the tab bar.

Syntax

```
WL.TabBar.init ()
```

Description

Initializes the tab bar, enabling it, but keeping it invisible. Must be called before any other function, except setParentDivId on Android.

Parameters

None.

Return Value

None

```
WL.TabBar.init();
```

WL.TabBar.isVisible:

Returns whether the Android tab bar is visible.

Syntax

```
WL.TabBar.isVisible ()
```

Description

Returns whether the Android tab bar is visible. Can be called only after the tab bar is initialized.

Parameters

None.

WL.TabBar.setEnabled:

Enables or disables the tab bar.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Syntax

```
WL.TabBar.setEnabled(isEnabled)
```

Description

Enables or disables the tab bar. When the tab bar is disabled, it is still visible, but all its items are disabled. However, the selected item remains selected.

Parameters

Table 68. WL.TabBar.setEnabled parameters

Parameter	Description
isEnabled	Mandatory Boolean. • `true`: Enable the tab bar • `false`: Disable the tab bar

Return Value

None

```
WL.TabBar.setEnabled(false);
```

WL.TabBar.RemoveAllItems:

Remove all items from a tab bar

Syntax

```
WL.TabBar.removeAllItems()
```

Description

Removes all the previously added items from the tab bar. The effect is immediate.

Parameters

None.

Return Value

None.

WL.TabBar.setParentDivId:

Place the tab bar within another element.

Syntax

```
WL.TabBar.setParentDivId(parentId)
```

Description

This method applies to Android only.

By default the tab bar is added to the element with ID *content*. In the application template that is generated by the IBM Worklight Studio, the <body> element has this ID. Use this function to place the tab bar within an arbitrary element. This

Experimental IBM Worklight offline user documentation.
This document is provided "as is".
In case of issues, refer to the online user documentation.

function is useful when the location of the tab bar is not at the top of the application screen.

Parameters

Table 69. WL.TabBar.setParentDivId parameters

Parameter	Description
parentId	Mandatory. Identifies the division in which the tab bar is placed.

Return Value

None.

WL.TabBar.setSelectedItem:

Selects an item in the tab bar.

Syntax

```
WL.TabBar.setSelectedItem (id)
```

Description

Selects the specified item of the tab bar, deselecting any other item. If the ID does not specify an existing tab, nothing happens.

Parameters

Table 70. WL.TabBar.setSelectedItem parameters

Parameter	Description
id	Mandatory. The ID of the tab to be selected.

Return Value

Integer: the ID of the selected tab.

WL.TabBar.setVisible:

Makes the tab bar visible or invisible.

Syntax

```
WL.TabBar.setVisible (isVisible)
```

Description

Determines whether the tab bar is visible. Call this method after the tab bar is initialized and all necessary tabs are added.

Experimental IBM Worklight offline user documentation.
This document is provided "as is".
In case of issues, refer to the online user documentation.

Parameters

Table 71. WL.TabBar.setVisible parameters

Parameter	Description
isVisible	Mandatory Boolean. • `true`: Shows the tab bar • `false`: Hides the tab bar

Return Value

None

```
WL.TabBar.setVisible(true);
```

WL.TabBarItem:

Do not create a TabBarItem manually.

Objects of this type are returned by the WL.TabBar.addItem function and must not be created manually.

WL.TabBarItem.setEnabled:

Enables or disables a tab bar item.

Syntax

```
WL.TabBarItem.setEnabled(isEnabled)
```

Description

Enables or disables the tab bar item.

Parameters

Table 72. WL.TabBarItem.setEnabled parameters

Parameter	Description
isEnabled	Mandatory Boolean. • `true`: Enable the tab bar item • `false`: Disable the tab bar item

Return Value

None

For iPhone:

```
var creditTab = WL.TabBar.addItem("CREDIT", selectCredit, "Visa", {image:"images/credit.png", badg

creditTab.setEnabled(false);
```

For Android:

```
var tabFeeds = WL.TabBar.addItem (
'tab2',
function(){worklightStarterApplication.selectTab('feedsWrapper');},
```

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

```
"Engadget Feeds",
{image:"images/feed.png",imageSelected:"images/feed.png"});

tabFeeds.setEnabled(false);
```

WL.TabBarItem.updateBadge:

On iOS only, updates the badge value on a tab bar item.

Syntax

```
WL.TabBarItem.updateBadge(badge)
```

Description

This method applies only to iOS.

Updates the badge value that is displayed on the tab bar item.

Parameters

Table 73. WL.TabBarItem.updateBadge parameters

Parameter	Description
badge	Optional string. The badge value to display on the item. If `null` or not specified, no badge value is displayed.

Return Value

None

```
var creditTab = WL.TabBar.addItem("CREDIT", selectCredit, "Visa", {image:"images/credit.png", badge:

creditTab.updateBadge("3");

// using null will remove the badge from the TabBar Item
creditTab.updateBadge(null);
```

Fixing the Tab Bar on the Screen – Android 2.2 and Above:

Fix the position of the tab bar by updating HTML and CSS.

About this task

To fix the tab bar in one location on the screen on Android 2.2 and above, perform the following steps:

Procedure

1. Add the following meta tag to the HTML HEAD section:
   ```
   <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, minimum-
   ```
2. Update the Android CSS BODY tag to also apply to the HTML tag, as follows:
   ```
   html, body {
   height: auto;
   overflow: auto;
   }
   ```

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues refer to the online user documentation.

Objective-C client-side API for native iOS apps

You can use Objective-C API to develop native apps for the iOS environment.

Use the Objective-C client-side API for native iOS apps that IBM Worklight provides if you want to access IBM Worklight services from native iOS applications

You can find the description of this API in the following document: Objective-C client-side API for native iOS apps.

Java client-side API for native Android apps

You can use Java API to develop native apps for the Android environment.

Use the Java client-side API for native Android apps that IBM Worklight provides if you want to access IBM Worklight services from native Android applications.

You can find the description of this API in the following document: Java client-side API for native Android apps.

Java client-side API for Java ME apps

You can use Java API to develop Java Platform, Micro Edition (Java ME) apps.

Use the Java client-side API for Java Platform, Micro Edition (Java ME) that IBM Worklight provides if you want to access IBM Worklight services from native Java ME apps.

You can find the description of this API in the following document: Java client-side API for Java Platform, Micro Edition.

IBM Worklight server-side API

Use the server-side API that IBM Worklight defines to modify the behavior of the servers that your mobile applications rely on.

JavaScript server-side API

The IBM Worklight server-side JavaScript API comprises these methods and objects.

The following table lists the methods and objects you can use to perform necessary functions in server-side applications.

Table 74. JavaScript API methods and objects

Function	Description
Accessing a web service	WL.Server.invokeHttp WL.Server.signSoapMessage
Accessing a JDBC database	WL.Server.invokeSQLStoredProcedure WL.Server.createSQLStatement WL.Server.invokeSQLStatement
Accessing a JMS-enabled messaging provider	WL.Server.readSingleJMSMessage WL.Server.readAllJMSMessages WL.Server.writeJMSMessage WL.Server.requestReplyJMSMessage

Experimental IBM Worklight online user documentation.
This document is provided "as is".
In case of issues, refer to the online user documentation.

Table 74. JavaScript API methods and objects (continued)

Function	Description
Calling other procedures	WL.Server.invokeProcedure
Accessing an HttpServletRequest object	WL.Server.getClientRequest
Accessing user details	WL.Server.getActiveUser WL.Server.setActiveUser
Subscribing to push notifications	WL.Server.createEventSource WL.Server.createDefaultNotification WL.Server.getUserNotificationSubscription WL.Server.notifyAllDevices WL.Server.notifyDeviceToken WL.Server.notifyDeviceSubscription
Accessing Server configuration	WL.Server.configuration
Debugging	WL.Logger.debug, error, and log

Method WL.Server.invokeHttp

Call an HTTP service.

Syntax

`WL.Server.invokeHttp(invocationData)`

Description

The method can be used only inside a procedure declared within an HTTP adapter. It calls a back-end HTTP service and converts the results to JSON.

Parameters

The invokeHttp function accepts the following JSON block of parameters:

Table 75. JSON block properties

Property	Description
method	Mandatory. Defines the HTTP method. Valid values are `get`, `post`, `put`, and `delete`.

Experimental IBM Worklight offline user documentation
This document is provided "as-is" without warranty of any kind
In case of issues, refer to the online user documentation.

Table 75. JSON block properties (continued)

Property	Description
returnedContentType	Optional. Defines the type of the content that is returned by the called HTTP service, overriding the HTTP response's mime type. If this parameter is not provided, the IBM Worklight Server handles the response according to the mime type. If it is provided, the IBM Worklight Server handles the response according to the indicated value. The field can receive the following values: • `css`, `csv`, `html`, `javascript`, `json`, `plain`, and `xml` • Any mime type that includes one of these values (note that any response with mime type that contains`javascript` or `json` is considered to contain JSON objects).
returnedContentEncoding	Optional. Defines the encoding of the returned content. Default is `utf-8`.
path	Mandatory. Defines the path of the URL defining the HTTP service.
parameters	Optional. Defines the set of parameters that need to be passed to the HTTP service.
headers	Optional. Defines the headers for the HTTP request.
cookies	Optional. Defines cookies to be passed with the HTTP request.
body	Defines the content of the request body. • When the method is GET, this property is not allowed. • When the method is POST, this property is optional. **Note:** `body.content` must be a string. If you are explicitly creating a DOM object, such as in: `var request = <soap:Envelope ... </soap:Envelope>;`, you must convert the object to string before you assign it to body.content, for example: `request.toString();`
transformation	Optional. If defined, the response of the service undergoes the defined XSL transformation. If the service returns HTML, the IBM Worklight Server first converts the response to XHTML, and then runs the XSL transformation on the XHTML response.

Experimental IBM Worklight offline use documentation
This document is provided "as-is".
In case of issues, refer to the online documentation.

Note: In IBM Worklight V5.0.6, the path is no longer modified when you make the actual request. You might therefore face an issue if you use the **parameters** property in the `WL.Server.invokeHttp` options with a query parameter specified in the path. You might end up with two ? signs on the request. To avoid this, do not use query parameters in the path along with the **parameters** property in WL.Server.invokeHttp when using the method GET.

Returned Value

The method returns the response of the HTTP service after the following processing:

1. If the service returns HTML, the IBM Worklight Server converts the HTML response to XHTML. If the service returns XML, the IBM Worklight Server keeps it as is.
2. If an XSL transformation is defined in the transformation property, the IBM Worklight Server runs the transformation on the result of Step 1. The transformation should convert its XML input to JSON. If no transformation was defined, the IBM Worklight Server automatically converts the result of Step 1 to JSON.

Example

```
var response = WL.Server.invokeHttp(invocationData);
response.responseHeader; // responseHeader property contains HTTP response headers
response.statusCode; // statusCode property contains HTTP response status code
```

Method WL.Server.signSoapMessage

Sign a fragment of a SOAP envelope.

Syntax

```
WL.Server.signSoapMessage (envelope, wsId, keystoreAlias)
```

Description

The method can be used only inside a procedure that is declared within an HTTP adapter. It signs a fragment of the specified envelope with ID wsId, by using the key in the specified **keystoreAlias**, inserting the digital signature into the input document.

To use WL.Server.signSoapMessage() API when IBM Worklight runs on IBM WebSphere Application Server, you might need to add a JVM argument that instructs WebSphere to use a specific **SOAPMessageFactory** implementation instead of a default one. To do this, you must go to **Application servers** *{server_name}* > **Process definition** > **Java Virtual Machine** and provide the following argument under Generic JVM arguments, typing in the code phrase exactly as it is presented here:

```
-Djavax.xml.soap.MessageFactory=com.sun.xml.internal.messaging.saaj.soap.ver1_1.SOAPMessag
```

Then, you must restart the JVM.

Important: This workaround is only for IBM WebSphere.

Parameters

Table 76. WL.Server.signSoapMessage method parameters

Parameter	Description
envelope	Mandatory. The SOAP envelope that contains the fragment to sign.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

Table 76. WL.Server.signSoapMessage method parameters (continued)

Parameter	Description
wsId	Mandatory. The value of the `wsu:Id` attribute, within the envelope, which identifies the fragment to be signed
keystoreAlias	Mandatory. The alias of the certificate or key entry in the keystore that is to be used for the signature.

Worklight Server Configuration

This method relies on the following properties in the `worklight.properties` file:

Table 77. Properties in the `worklight.properties` file

Property	Description	Example
ssl.keystore.path	Path to the certificate that contains the key with which the envelope fragment must be encrypted	`/conf/default.keystore`
ssl.keystore.type	Type of the keystore	`jks`
ssl.keystore.password	Password of the keystore (can be encrypted)	`worklight`

Example

```
var myEnvelope =
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
<soapenv:Header>
...
</soapenv:Header>
<soapenv:Body wsu:Id="an-element-id" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-20040
...
</soapenv:Body>
</soapenv:Envelope>;

WL.Server.signSoapMessage(myEnvelope , "an-element-id", keystoreAlias);
```

Method WL.Server.invokeSQLStoredProcedure

Call a stored procedure on a database.

Syntax

```
WL.Server.invokeSQLStoredProcedure(options)
```

Description

The method can be used only inside a procedure that is declared within an SQL adapter.

It calls a stored procedure on a database.

Parameters

The invokeSQLStoredProcedure function accepts the following JSON block of parameters:

Experimental IBM Worklight online user documentation.
This document is provided "as is".
In case of issues, refer to the online user documentation.

```
{
  procedure: 'procedure-name',
  parameters: [value-1, value-2, ...]
}
```

The JSON block contains the following properties:

Table 78. JSON block properties

Property	Description
procedure	Mandatory. This string defines the name of the stored procedure to call.
parameters	Optional. An array of parameters to the stored procedure.

Returned Value

The method returns the result set of the SQL stored procedure. This returned value is formatted as a JSON array, in which each element corresponds to a row in the result set of the SQL stored procedure.

Method WL.Server.createSQLStatement

Create a prepared SQL statement.

Syntax

```
WL.Server.createSQLStatement(statement)
```

Description

The method can be used only inside a procedure declared within an SQL adapter. It must be used outside of the scope of any JavaScript function.

Creates a prepared SQL statement to be later invoked with `WL.Server.invokeSQLStatement`.

Parameters

The `invokeSQLStatement` method accepts the following parameter:

Table 79. invokeSQLStatement method parameter

Parameter	Description
statement	Mandatory string. An SQL statement with one of the following verbs: select, insert, delete, update. Use a question mark (?) as a parameter placeholder.

Returned Value

An object that represents the prepared statement.

Example

```
// Outside any function scope
var procedure1Statement = WL.Server.createPreparedStatement("select COLUMN1, COLUMN2 from TABLE1 whe
```

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Method WL.Server.invokeSQLStatement

Call a prepared SQL statement.

Syntax

```
WL.Server.invokeSQLStatement(options)
```

Description

The method can be used only inside a procedure that is declared within an SQL adapter.

It calls a prepared SQL statement that was created with WL.Server.createSQLStatement.

Parameters

```
{
  preparedStatement: prepared-statement-variable,
  parameters: [value-1, value-2, ...]
}
```

The JSON block contains the following properties:

Table 80. JSON block properties

Property	Description
preparedStatement	Mandatory. The prepared statement that was defined previously with createSQLStatement.
parameters	Optional. An array of parameters to the prepared statement.

Returned Value

The method returns the result set of the prepared statement. This returned value is formatted as a JSON array, in which each element corresponds to a row in the result set of the prepared statement.

```
// Outside of the function scope
var procedure1Statement = WL.Server.createPreparedStatement("select COLUMN1, COLUMN2 from TABLE1 w

function procedure1(param) {
  return WL.Server.invokePreparedStatement({
    preparedStatement : procedure1Statement,
    parameters : [param]}
  );
}
```

Method WL.Server.readSingleJMSMessage

Read a single message from the given destination.

Syntax

```
WL.Server.readSingleJMSMessage(options)
```

Description

The method attempts to read a single message from the given destination.

If the destination is a queue, this method also removes the message from the queue.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Parameters

The readSingleJMSMessage function accepts the following JSON block of parameters:

```
{
     destination : jndi-name-of-the-destination,
     timeout     : wait-timeout-in-milliseconds,
     filter      : jms-filter-string
}
```

The JSON block contains the following properties:

Table 81. JSON block properties

Property	Description
destination	Mandatory. The name of the administered destination object, held in the JNDI repository, that the message will be received from. For example: If the administered destination object is a JEE container managed object, the value may be `java:comp/env/....`
timeout	Optional. The time, in milliseconds, that the method will wait for a message, if a message is not immediately available. If **timeout** is not specified, the method will not wait for a message. **Special values for this parameter:** **0** wait indefinitely **<0** do not wait
filter	Optional. The JMS selector string applied to the wait call. The filter follows the standard JMS selector syntax rules.

Returned Value

The method returns the received message. If no message is immediately available on the destination, the method waits for the specified millisecond timeout. If no message is available after the specified timeout, the method returns successfully but with no message.

The message must be of type JMSText. If the message is not of type JMSText, it is read from the destination and written to the warnings element of the response using the javax.jms.Message.toString() method.

The returned object has the following structure:

```
{
   isSuccessful: Boolean,
   errors      : optional-error-messages,
   warnings    : optional-warnings-messages
   message     : { body : body of the message,
                   properties : properties: of the message
                 }
}
```

The invocation results object contains the following properties:

Experimental IBM Worklight offline documentation
This document is provided for convenience. In case of issues, refer to the online documentation.

Table 82. Invocation results object properties

Property	Description
`isSuccessful`	Identifies whether the method invocation succeeded or failed. Valid values are: **true** The method invocation succeeded. This is the default value. Also set to true if there is no message on the destination and the method returns without error. **false** The method invocation failed.
`errors`	Optional. Any errors during processing will appear here.
`warnings`	Optional. Any warnings during processing will appear here. This includes warnings about any messages not of a supported JMS Message Type.
`message`	Optional. The message, the message body, and the message properties are all optional. **body** The message body. If no message is returned, this will not be available. **properties** An array of message properties which follow the JMSMessage property rules. The following message properties can be returned: • JMS* properties. For example: JMSCorrelationID. • JMS_Provider_* properties. For example: JMS_IBM_Format. • user properties. For example: my_user_property.

Method WL.Server.readAllJMSMessages

Read all messages from the given destination.

Syntax

```
WL.Server.readAllJMSMessages(options)
```

Description

The method attempts to read all messages from the given destination.

If the destination is a queue, this method also removes the messages from the queue.

Parameters

The readAllJMSMessages function accepts the same set of parameters as readSingleJMSMessage.

Experimental IBM Worklight offline user documentation. This document is provided as-is. In case of issues, refer to the online user documentation.

Returned Value

The method returns all available messages on the destination. If no messages are immediately available on the destination, the method waits for the specified millisecond timeout.

Note: The timeout is applied per message. If a message is available within the timeout period, it is added to the list of received messages. The method then performs another wait with the same timeout for the next available message. If no messages are available after the specified timeout, the method returns successfully but with no messages.

The messages must be of type JMSText. If an individual message is not of type JMSText, it is read from the destination and added to the warnings element of the response. The method then continues to attempt to read messages from the destination. If there is an error processing the messages, an optional error parameter is returned in the result.

The returned object is a list of received messages. Each individual message holds the same body type and property list as readSingleJMSMessage.

Example of a returned object:

```
{
   isSuccessful: Boolean,
   messages    : [
                   {   body        : body of the message,
                       properties  : {properties: of the message,
                                       ....}
                   },
                   {   body        : body of the next message,
                       properties  : {properties: of the next message,
                                       ....}
                   }
                 ]
}
```

Method WL.Server.writeJMSMessage

Write a single JMSText message to the given destination.

Syntax

```
WL.Server.writeJMSMessage(options)
```

Description

The method writes a single JMSText message to the given destination.

The method options include write options, the message body, and message properties.

Parameters

The writeJMSMessage function accepts the following JSON block of parameters:

```
{
     destination : jndi-name-of-the-destination,
     message     : { body : message body,
                     properties : { message-properties }
                   },
     ttl         : time-to-live-in-milliseconds
}
```

Experimental IBM Worklight offline user documentation. This document is provided as is. In case of issues, refer to the online user documentation.

The JSON block contains the following properties:

Table 83. JSON block properties

Property	Description
destination	Mandatory. The name of the administered destination object, held in the JNDI repository, that the message will be written to. The administered object can be defined as a queue or a topic. For example: If the administered destination object is a JEE container managed object, the value might be `java:comp/env/....`
message	Optional. The message to write to the destination. The message, the message body, and the message properties are all optional. If there is no message body or message properties, an empty message will be sent. The properties follow the same property naming and setting rules as standard JMS messages.
ttl	Optional. The message time-to-live. The time is in milliseconds. If not specified, the time-to-live is set to infinite.

Returned Value

If the method is successful, the JMSMessageID of the sent message is returned.

The returned object has the following structure:

```
{
   isSuccessful: Boolean,
   errors      : optional-error-messages,
   JMSMessageID : ID:jms-message-id
}
```

The invocation results object contains the following properties:

Table 84. Invocation results object properties

Property	Description
isSuccessful	Identifies whether the method invocation succeeded or failed. Valid values are: **true** The method invocation succeeded. This is the default value. **false** The method invocation failed.
errors	Optional. Any errors during processing will appear here.
JMSMessageID	Optional. If the message was sent successfully, this is the message ID of the sent message. The message ID uses the standard of the underlying JMS Message provider. For example: JMSMessageID : ID:414d234e132a43c123d2b3c1e5a4a4b32132c.

Experimental IBM Worklight offline use documentation. This document is provided as a convenience. In case of issues, refer to the online user documentation.

Method WL.Server.requestReplyJMSMessage

Write a single JMSText message to the given destination and wait for the response.

Syntax

```
WL.Server.requestReplyJMSMessage(options)
```

Description

This method is designed for use when a service is called that uses the replyTo destination in the originating message to send the response to.

The IBM Worklight server creates a temporary JMS destination for the reply to be received on. The temporary JMS destination is deleted using the underlying JMS provider cleaning up rules. The temporary destination that is created is of the same type as the specified destination. For example: If the specified destination is a queue, then a temporary queue is created as the replyTo destination.

Parameters

The requestReplyJMSMessage function accepts the following JSON block of parameters:

```
{
     destination : jndi-name-of-the-destination,
     message     : { body : message body,
                     properties : { message-properties },
     timeout     : wait-timeout-in-milliseconds,
     ttl         : time-to-live-in-milliseconds
}
```

The JSON block contains the following properties:

Table 85. JSON block properties

Property	Description
destination	Mandatory. The name of the administered destination object, held in the JNDI repository, that the message is written to. The administered object can be defined as a queue or a topic. For example: If the administered destination object is a JEE container managed object, the value might be `java:comp/env/...`.
message	Optional. The message to write to the destination. The message, the message body, and the message properties are all optional. If there is no message body or message properties, an empty message is sent. The properties follow the same property naming and setting rules as standard JMS messages.

Experimental IBM Worklight online user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Table 85. JSON block properties (continued)

Property	Description
timeout	Optional. The time, in milliseconds, that the method waits for a message, if a message is not immediately available. If **timeout** is not specified, the method will not wait for a message. **Special values for this parameter:** **0** wait indefinitely; **<0** do not wait
ttl	Optional. The message time-to-live. The time is in milliseconds. If not specified, the time-to-live is set to infinite.

Returned Value

The requestReplyJMSMessage function follows the same syntax and rules as readSingleJMSMessage.

Method WL.Server.invokeProcedure

Invoke a procedure that is exposed by a Worklight adapter.

Syntax

```
WL.Server.invokeProcedure (invocationData)
```

Parameters

The invokeProcedure function accepts the following JSON block of parameters:

Table 86. JSON block properties

Property	Description
adapter	Mandatory. A string that contains the name of the adapter as specified when the adapter was defined.
procedure	Mandatory. A string that contains the name of the procedure as specified when the adapter was defined.
parameters	Optional. An array of parameter values that are passed to the back-end procedure. A parameter can be a scalar or an object.

Example of a JSON block of Parameters

```
{
adapter : "AcmeBank",
procedure : " getTransactions",
parameters : [accountId, fromDate, toDate],
};
```

Experimental IBM Worklight online user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

Returned Value

```
{
isSuccessful: Boolean,
errorMessages: ["Error Msg1", ...],
// Application object returned by procedure
}
```

The invocation results object contains the following properties:

Table 87. Invocation results object properties

Property	Description
`isSuccessful`	Optional. Identifies whether the procedure invocation succeeded or failed. Valid values are: **true** The procedure invocation succeeded. This is the default value. **false** The procedure invocation failed.
`errorMessages`	Optional. An array of strings that contain error messages. If no errors are provided, the returned array is empty.
`Application object`	Any object that is returned by the procedure.

Method WL.Server.getClientRequest

This method returns a reference to the Java HttpServletRequest object that was used to invoke an adapter procedure

Syntax

```
WL.Server.getClientRequest ()
```

Description

Returns a reference to the Java HttpServletRequest object that was used to invoke an adapter procedure. This method can be used in any adapter procedure.

Use this method to return headers or other information stored in an HttpServletRequest object.

Parameters

None.

Return Value

A reference to an HttpServletRequest object.

Example

```
var request = WL.Server.getClientRequest();
var userAgent = request.getHeader("User-Agent");
```

Method WL.Server.getActiveUser

Return an object that contains user identity properties.

Syntax

```
WL.Server.getActiveUser ()
```

Experimental IBM Worklight offline user documentation
This document is provided as is. In case of issues, refer to the online user documentation.

Description

Returns an object with the user identity properties, according to the following rules:

- If no realm is defined on the adapter, the method returns `null` (active user is unknown)
- If a realm is defined on the adapter:
  - If there is no strong identity associated with the user (the user was authenticated in this session or in a previous session), the method returns `null`.
  - If there is a strong identity associated with the user (from the current session or a previous one), the method returns the strong identity.

Parameters

None.

Return Value

An object that contains the user identity properties, as defined by the login module, with the following structure:

userId The login ID, mandatory

displayName
Optional

credentials
Optional. A string with the user's credentials, such as password

attributes
Optional. Custom user attributes. There are no constraints on the structure of the object.

Example

```
{
userId: "38017840288"
displayName: "John Doe",
attributes: {lastLogin: "2010-07-13 19:25:08.0 GMT"}
}
```

Method WL.Server.setActiveUser

Create a user identity in a specified realm.

Syntax

```
WL.Server.setActiveUser (realm, identity)
```

Description

Used in authenticator adapters at the end of the login sequence. Creates a user identity in the specified realm with the properties in the specified **`identity`** parameter. As a result of this method, the user's session is considered authenticated.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Parameters

Table 88. WL.Server.setActiveUser method parameters

Property	Description
`realm`	Mandatory. The realm to log in to (as defined in the `authenticationConfig.xml` file).
`identity`	Mandatory. A user identity object, as returned by WL.Server.getActiveUser, with the following structure: **userId** Mandatory. The login ID. **displayName** Optional. **credentials** Optional. A string with the user's credentials, such as password. **attributes** Optional. Custom user attributes,

Return Value

None.

Example

```
WL.Server.setActiveUser ("ACMERealm", {
userId: "38017840288",
displayName: "John Doe",
attributes: {
"firstName": "John",
"lastName": "Doe",
"lastLogin": "2010-07-13 19:25:08.0 GMT",
}
})
```

Method WL.Server.createEventSource

Create an event source.

Syntax

```
WL.Server.createEventSource(JSON-parameter-block)
```

Description

Creates an event source according to the parameters in the parameter block.

Parameters

The JSON block contains the following properties:

Table 89. JSON block properties

Property	Description
`name`	Mandatory. A string that contains the name of the event source.

Experimental IBM Worklight offline user documentation. This document is provided as is. In case of issues, refer to the online user documentation.

Table 89. JSON block properties (continued)

Property	Description
onUserSubscribe	Optional. The name of the JavaScript function (in the adapter file) that is called when the user subscribes to this event source for the first time, on first device subscription. The callback function receives the user subscription object as an input parameter.
onUserUnsubscribe	Optional. The name of the JavaScript function (in the adapter file) that is called when the user unsubscribes from this event source for the first time, on first device subscription. The callback function receives the user subscription object as an input parameter.
onDeviceUnsubscribe	Optional. The name of the JavaScript function that is called when the device subscription is removed by a client request or by the cleanup task. The callback function receives the device subscription as an input parameter.
onUserChange	Optional. The name of the JavaScript function that is called when a subscription from this device exists for a different user. The callback function receives the options sent to the createEventSource function. The callback function must return a JSON block that must contain at least an **isSuccessful** property, indicating whether the subscription should be created. The returned JSON block can contain other custom properties, and it is transferred back to the client app.
poll	Optional. If the method of getting the notification data from the back-end is polling, provide the following properties: **interval** Mandatory. The interval in seconds between the polls. **onPoll** Mandatory. The name of JavaScript function which is called on each poll.
securityTest	Mandatory. Declares the appropriate securityTest from `authenticationConfig.xml` to be used for this event source.

Experimental IBM Worklight offline user documentation.
This document is provided "as is".
In case of issues, refer to the online user documentation.

Example

```
WL.Server.createEventSource({
name: 'newCoupons',
onUserSubscribe: 'subscribeUser',
onUserUnsubscribe: 'unsubscribeUser',
onUserChange: 'onUserChange',
poll : {
```

```
interval: 2,
onPoll: 'produceNotifications'
}
});
```

Related links: Security Tests

Method WL.Server.createDefaultNotification

Return a default notification JSON block structure.

Syntax

```
var notification = WL.Server.createDefaultNotification(notificationText, badge, payload);
```

Description

The method creates and returns a notification JSON block for the supplied values, for all supported environments:

- Push notifications on iOS, Android, and Windows Phone 8
- SMS push notifications on iOS, Android, Windows Phone 7.5, Windows Phone 8, and BlackBerry devices that support SMS functions

Parameters

Table 90. WL.Server.createDefaultNotification(notificationText, badge, payload) method parameters

Property	Description
notificationText	Optional. The string that is displayed in the alert. On Windows Phone 8 the string is displayed in the application tile title.
badge	Optional. An integer value that is displayed in a badge on the application icon. On Windows Phone 8 the value is displayed as the application tile count.
payload	Optional. A JSON block that is transferred to the application. On iOS and Android, the payload is transferred to the application if the application is opened by the user when the notification is received, or if the application is already open. On Windows Phone 8, the payload is transferred to the application as a raw notification only if the application is already open.

Returned Value

The returned JSON block has the following structure:

```
  APNS: {
   badge: badge,
   alert: notificationText,
   payload: payload,
   sound: "",
   actionKey: null
   },
  GCM: {
   alert: notificationText,
   badge: badge,
   payload: payload
```

Experimental IBM Worklight offline user documentation
This document is provided "as-is".
In case of issues, refer to the online user documentation.

```
  },
 SMS: {
  text: notificationText
  },
 MPNS: {
  raw: {
   payload: payload
   },
  toast: null,
  tile: {
   title: notificationText,
   count: badge
   }
  }
```

Example

```
var notification = WL.Server.createDefaultNotification("You have " + numCoupons + " coupons.", num
```

Method WL.Server.getUserNotificationSubscription

Return a subscription object for a user.

Syntax

```
WL.Server.getUserNotificationSubscription(eventSourceName, userId);
```

Description

Returns a subscription object for the user with the specified ID to the specified event source.

Parameters

Table 91. WL.Server.getUserNotificationSubscription method parameters

Parameter	Description
eventSourceName	Mandatory. A string that contains the name of the event source.
userId	Mandatory. A string that contains the user ID, created during the login process. The user ID can be obtained by calling WL.Server.getActiveUser.

Return Value

The method returns a subscription object that contains the user ID and the mutable subscription state.

Note: All subscription object fields are read-only, except for the user subscription state. You can modify the user subscription state in your JavaScript code, and then must use the `save` method to save it to the IBM Worklight database.

Example

```
{userId: 'bjones', state: {numCoupons: 3}}
```

Method userSubscription.getDeviceSubscriptions

Return an array of device subscriptions.

Syntax

```
userSubscription.getDeviceSubscriptions();
```

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Description

Returns an array of device subscriptions for the user subscription object on which it is invoked.

Parameters

None.

Return Value

The method returns an array of the device subscriptions. The device subscriptions contain the device token, the application ID, the platform, and the options that were passed by the client in the subscribe call.

Example

```
[
{
alias: "myPush,
device: "123123123123....",
token: '53d8d76d0ec54b79552dd98dfeb4b4565c2b13cad53ff3898e7441d1f91b4574',
applicationId: 'HelloBookstore',
platform: 'Apple',
userAgent: 'Mozilla/5.0 (iPad; ...',
options: {foo: 'bar', alert: true, badge: true, sound: true}
}
]
```

Method userSubscription.save

Save the state of a user subscription.

Syntax

```
userSubscription.save();
```

Description

Saves the state of the user subscription. Should be called only after you explicitly change the **`state`** property of the user subscription object.

Parameters

None.

Return Value

None.

Method WL.Server.notifyAllDevices

Submit a notification to all a specified user's device subscriptions

Syntax

```
WL.Server.notifyAllDevices(userSubscription, notificationOptions)
```

Description

Submits a notification to all the device subscriptions of the specified user, according to the specified options.

Experimental IBM Worklight online user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Push notifications are supported on iOS, Android, and Windows Phone 8 devices. iOS apps use the Apple Push Notification Service (APNS), Android apps use Google Cloud Messaging (GCM), and Windows Phone 8 apps use the Microsoft Push Notification Service (MPNS).

Note: If a notification to a specific Windows Phone 8 device subscription fails, the notification is dismissed and not resubmitted.

SMS push notifications are supported on iOS, Android, Windows Phone 7.5, Windows Phone 8, and BlackBerry devices that support SMS functions.

For SMS notifications, the *text* property of the **notificationOptions** parameter contains the SMS text. A text message is sent as a single message if the text message length is less than or equal to 160 characters. If the text message length is greater than 160 characters, the message is either split into multiple messages of 160 characters or less, or it is rejected. The action that is taken depends on the configured SMS gateway. All other properties of **notificationOptions** are ignored.

Parameters

In IBM Worklight V5.0.6, the JSON block for the **notificationOptions** parameter has changed. The JSON block that was used in IBM Worklight V5.0.5 and earlier is deprecated in IBM Worklight V5.0.6. Support might be removed in any future version. See "Parameters for IBM Worklight V5.0.5 and earlier" on page 297 for details on the JSON block that was used in IBM Worklight V5.0.5 and earlier, and how it is used in IBM Worklight V5.0.6.

The **notificationOptions** parameter accepts the following JSON block:

```
APNS
          alert
          badge
          sound
          actionKey
          payload
GCM
          alert
          sound
          payload
          delayWhileIdle
          timeToLive
SMS
          text
MPNS
          raw
                    payload
          toast
                    text1
                    text2
                    param
          tile
                    id
                    count
                    title
                    backgroundImage
                    smallBackgroundImage
                    wideBackgroundImage
                    wideBackBackgroundImage
                    wideBackContent
                    backBackgroundImage
                    backTitle
                    backContent
```

Experimental IBM Worklight user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

```
smallIconImage
iconImage
wideContent1
wideContent2
wideContent3
backgroundColor
cycleImage1
cycleImage2
cycleImage3
cycleImage4
cycleImage5
cycleImage6
cycleImage7
cycleImage8
cycleImage9
```

Table 92. WL.Server.notifyAllDevices(userSubscription, notificationOptions) method parameters

Name	Description
`userSubscription`	Mandatory. A user subscription object, obtained by calling WL.Server.getUserNotificationSubscription

Experimental IBM Worklight offline user documentation
This document is provided "as-is".
In case of issues, refer to the online user documentation.

Table 92. WL.Server.notifyAllDevices(userSubscription, notificationOptions) method parameters (continued)

Name	Description
`notificationOptions`	Mandatory. The JSON block contains the following properties: **APNS** **alert** Optional. A string to be displayed in the alert. **badge** Mandatory. An integer value to be displayed in a badge on the application icon. **sound** Optional. The name of a file to play when the notification arrives. **actionKey** Optional. The label of the dialog box button that allows the user to open the app upon receiving the notification. **payload** Optional. A JSON block that is transferred to the application if the application is opened by the user when the notification is received, or if the application is already open. **GCM** **alert** Optional. A string to be displayed in the alert. **sound** Optional. The name of a file to play when the notification arrives. **payload** Optional. A JSON block that is transferred to the application if the application is opened by the user when the notification is received, or if the application is already open. **delayWhileIdle** Optional. A Boolean value that indicates that the message should not be sent if the device is idle. The server waits for the device to become active before the message is sent. Default value is `false`. **timeToLive** Optional. How long, in seconds, the message should be kept on GCM storage if the device is offline. Default value is 4 weeks, and must be set as a JSON number. **SMS** **text** Mandatory. A string to be displayed in the alert. **MPNS** **raw** **payload** Optional. A JSON block that is transferred to the

Experimental IBM Worklight offline user documentation
This document is provided "as-is".
In case of issues, refer to the online user documentation.

Windows Phone 8 Tile images can be local or remote. Store a local image, for example, `myImage.jpg`, as a web resource in the images folder. The URL of the image is then `www/default/images/myImage.jpg`. To use a remote image, you must declare the remote image domain in the `<allowedDomainsForRemoteImages>` subelement of `<windowsPhone8>` in the `application-descriptor.xml` file.

To clear a Windows Phone 8 Tile property, set it to an empty string for texts and URLs, or to zero for the count property, when you send a notification.

Note:
- If you submit a notification to MPNS and none of the appropriate properties are set, then the notification is not sent; for example, a raw notification is not sent if **`payload`** is not set; a toast notification is not sent if **`text1`**, **`text2`**, and **`param`** are not set.
- If you submit a notification to Windows Phone 8 with Tile properties that do not match the Tile template declared in the `WMAppManifest.xml` file, the Tile notification is ignored by the device OS.
- If you declare a Windows Phone 8 Tile as `cycle` in the `WMAppManifest.xml` file and the notification comprises only the title and count properties, then the notification is ignored by the device OS. As a workaround, add one of the **`notificationOptions`** cycle properties that are described in Table 92 on page 292.
- If you declare a Windows Phone 8 Tile as `iconic` in the `WMAppManifest.xml` file and the notification comprises only the title and count properties, then the notification is ignored by the device OS. As a workaround, add one of the **`notificationOptions`** iconic properties that are described in Table 92 on page 292.

See http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj662938%28v=vs.105%29.aspx for more details on Windows Phone 8 toast notifications.

See http://msdn.microsoft.com/en-us/library/windowsphone/develop/hh202948%28v=vs.105%29.aspx for more details on Windows Phone 8 Tiles.

Use the following process to create a notification that can be sent to all devices:
1. Call WL.Server.createDefaultNotification to create a default notification. See "Method WL.Server.createDefaultNotification" on page 288 for details on this method.
2. Individually set or change any property in the returned default notification.
3. Call WL.Server.notifyAllDevices with the updated notification.

Example

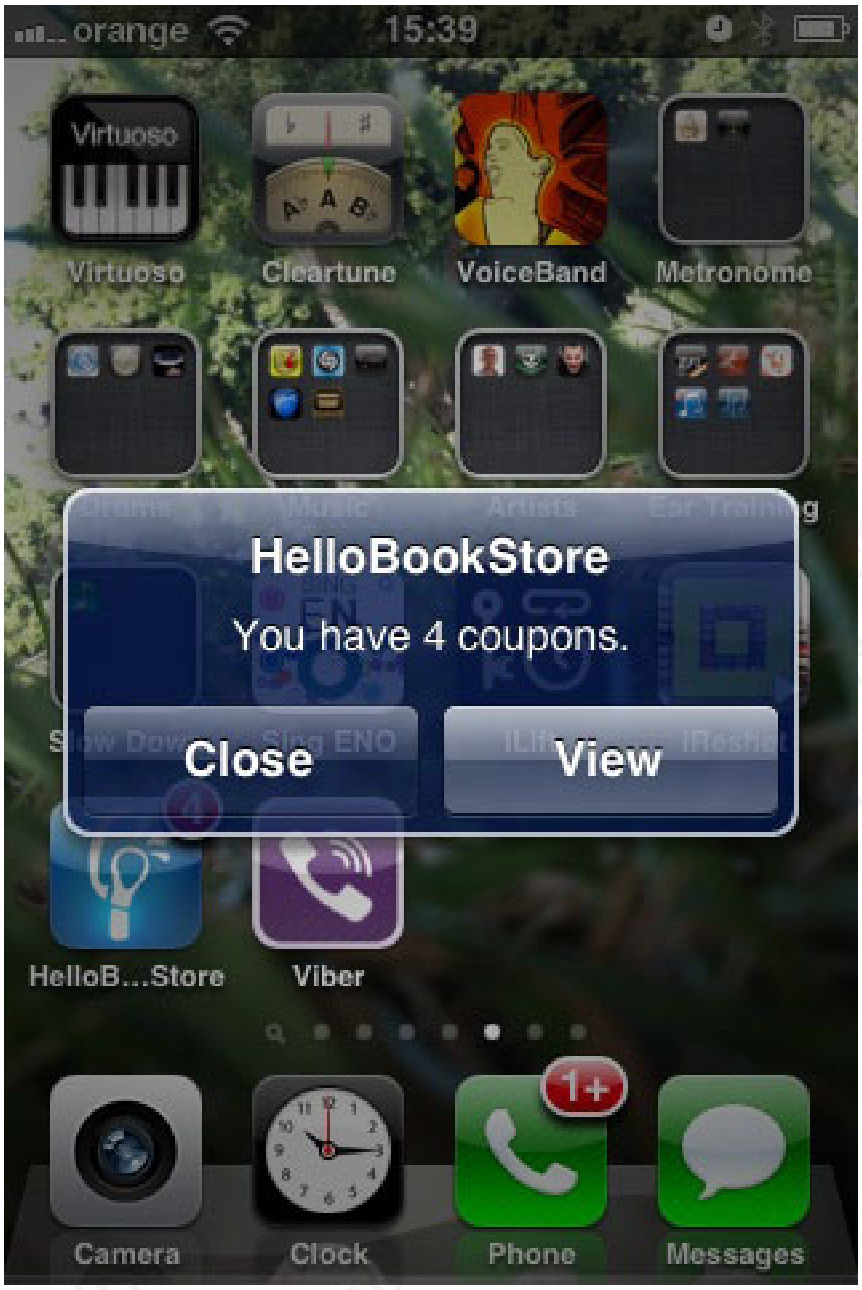
```
userSubscription = WL.Server.getUserNotificationSubscription ("MyEventSource", userID);

var notification = WL.Server.createDefaultNotification("You have " + numCoupons + " coupons.", numCou

// change the sound for APNS
notification.APNS.sound = mySound;
// change the alert for GCM
notification.GCM.alert = myAndroidAlert;
// change the payload for MPNS
notification.MPNS.raw.payload = myRawPayload;
// set toast notification properties for MPNS
notification.MPNS.toast = {};
notification.MPNS.toast.text1 = "Toast title";
```

Experimental IBM Worklight documentation.
This document is provided "as is". In case of issues, refer to the online user documentation.

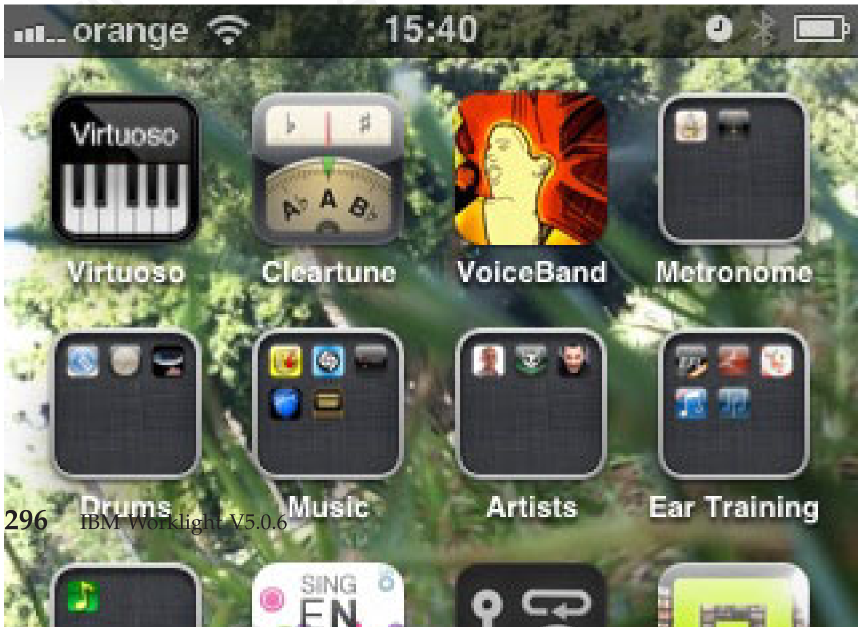
```
notification.MPNS.toast.text2 = "Toast content";
// set a local image for MPNS
notification.MPNS.tile.backgroundImage = "www/default/images/myImage.jpg";
// set a remote image for MPNS
notification.MPNS.tile.backBackgroundImage = "http://icons.aniboom.com/Animators/00e45896-68c6-446

WL.Server.notifyAllDevices(
userSubscription, notification
);
```

The following figure shows the notification dialog box, text, and badge numbers:

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.


HelloBookStore

You have 4 coupons.



Parameters for IBM Worklight V5.0.5 and earlier

The JSON block that was used in IBM Worklight V5.0.5 and earlier is deprecated in IBM Worklight V5.0.6. Support might be removed in any future version. If the deprecated JSON block structure is used in IBM Worklight V5.0.6, a deprecation warning message is printed to the console. A notification message is still submitted to all supported environments, however, the notification message that is sent to a Windows Phone 8 device is sent as two notifications:

- A tile message, which contains the badge, or count, and an alert, or title
- A raw message, which contains the payload

Table 93. WL.Server.notifyAllDevices(userSubscription, notificationOptions) method parameters

Name	Description
`userSubscription`	Mandatory. A user subscription object, obtained by calling WL.Server.getUserNotificationSubscription
`notificationOptions`	Mandatory. The JSON block contains the following properties: **badge** Mandatory. An integer value to be displayed in a badge on the application icon. **sound** Optional. The name of a file to play when the notification arrives. **alert** Optional. A string to be displayed in the alert. **activateButtonLabel** Optional. The label of the dialog box button that allows the user to open the app upon receiving the notification. **payload** Optional. A JSON block that is transferred to the application if the application is opened by the user when the notification is received, or if the application is already open.

Method WL.Server.notifyDevice

Submit a notification to a specified user and a specified device.

Syntax

```
WL.Server.notifyDevice(userSubscription, device, options)
```

Description

Submits a notification to the specified user with the specified device ID according to the specified options.

If the device ID does not exist, the server outputs an error to the log and returns.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

Push notifications are supported on iOS, Android, and Windows Phone 8 devices. iOS apps use the Apple Push Notification Service (APNS), Android apps use Google Cloud Messaging (GCM), and Windows Phone 8 apps use the Microsoft Push Notification Service (MPNS).

Note: If a notification to a specific Windows Phone 8 device subscription fails, the notification is dismissed and not resubmitted.

SMS push notifications are supported on iOS, Android, Windows Phone 7.5, Windows Phone 8, and BlackBerry devices that support SMS functions.

Useful when the notifications are generated by a back-end system, and directs them to specific device IDs.

For SMS notifications, the *text* property of the **`options`** parameter contains the SMS text. A text message is sent as a single message if the text message length is less than or equal to 160 characters. If the text message length is greater than 160 characters, the message is either split into multiple messages of 160 characters or less, or it is rejected. The action that is taken depends on the configured SMS gateway. All other properties of **`options`** are ignored.

Parameters

Table 94. WL.Server.notifyDevice method parameters

Name	Description
`userSubscription`	Mandatory. A user subscription object, obtained by calling WL.Server.getUserNotificationSubscription
`device`	Mandatory. The device ID that is used to identify the device by the Worklight Server.
`options`	Mandatory. See Method WL.Server.notifyAllDevices.

Use the following process to create a notification that can be sent to any device:
1. Call WL.Server.createDefaultNotification to create a default notification. See "Method WL.Server.createDefaultNotification" on page 288 for details on this method.
2. Individually set or change any property in the returned default notification.
3. Call WL.Server.notifyDevice with the updated notification.

Example

```
userSubscription = WL.Server.getUserNotificationSubscription ("MyEventSource", userID);

var notification = WL.Server.createDefaultNotification("You have " + numCoupons + " coupons.", numCou

WL.Server.notifyDevice(
userSubscription, userSubscription.getDeviceSubscriptions()[0].token, notification
);
```

Method WL.Server.notifyDeviceSubscription

Submit a notification to the specified device of a subscribed user.

Syntax

```
WL.Server.notifyDeviceSubscription (deviceSubscription, notificationOptions)
```

Experimental IBM Worklight offline user documentation.
This document is provided as-is.
In case of issues, refer to the online user documentation.

Description

This method replaces the deprecated method `WL.Server.submitNotification`.

Submits a notification to the specified device of a subscribed user, according to the specified options.

Push notifications are supported on iOS, Android, and Windows Phone 8 devices. iOS apps use the Apple Push Notification Service (APNS), Android apps use Google Cloud Messaging (GCM), and Windows Phone 8 apps use the Microsoft Push Notification Service (MPNS).

Note: If a notification to a specific Windows Phone 8 device subscription fails, the notification is dismissed and not resubmitted.

SMS push notifications are supported on iOS, Android, Windows Phone 7.5, Windows Phone 8, and BlackBerry devices that support SMS functions.

For SMS notifications, the *text* property of the **`notificationOptions`** parameter contains the SMS text. A text message is sent as a single message if the text message length is less than or equal to 160 characters. If the text message length is greater than 160 characters, the message is either split into multiple messages of 160 characters or less, or it is rejected. The action that is taken depends on the configured SMS gateway. All other properties of **`notificationOptions`** are ignored.

Parameters

Table 95. WL.Server.notifyDeviceSubscription method parameters

Name	Description
`deviceSubscription`	Mandatory. The device subscription, obtained by calling getDeviceSubscriptions on the user subscription object.
`notificationOptions`	Mandatory. See Method WL.Server.notifyAllDevices.

Use the following process to create a notification that can be sent to any device:
1. Call WL.Server.createDefaultNotification to create a default notification. See "Method WL.Server.createDefaultNotification" on page 288 for details on this method.
2. Individually set or change any property in the returned default notification.
3. Call WL.Server.notifyDeviceSubscription with the updated notification.

Example

```
userSubscription = WL.Server.getUserNotificationSubscription ("MyEventSource", userID);

var notification = WL.Server.createDefaultNotification("You have " + numCoupons + " coupons.", num

WL.Server.notifyDeviceSubscription(
userSubscription.getDeviceSubscriptions()[0], notification
);
```

Deprecated Method WL.Server.submitNotification

This method is deprecated.

Experimental IBM Worklight online user documentation. This document is provided as-is. In case of issues, refer to the online user documentation.

Syntax

```
WL.Server.submitNotification (deviceSubscription, notificationOptions)
```

Description

Note: This method is deprecated as of version 4.1.3, and should be replaced by **WL.Server.notifyDeviceSubscription**, which has the same signature.

Submits a notification to the specified device of a subscribed user, according to the specified options.

It is possible to submit notifications only to iOS and Android devices.

Parameters

Table 96. Deprecated Method WL.Server.submitNotification parameters

Name	Description
`deviceSubscription`	Mandatory. The device subscription obtained by calling getDeviceSubscriptions on the user subscription object.
`notificationOptions`	Mandatory. See Method WL.Server.notifyAllDevices.

Methods WL.Logger.debug, error, and log

Logger methods for server-side code.

Syntax

- `WL.Logger.debug (message)`
- `WL.Logger.error (message)`
- `WL.Logger.log (message)`

Description

Logger methods for server-side code. Each method writes a message to the *`Worklight Project Name`*`/server/log/server/server.log` file.

Note: For the development environment only, you can define the level of the traces that are logged by setting the appropriate values in the `development.logging.properties` file that is in the *`Worklight Project Name`*`/server/conf` folder. This property file contains default definitions that you can use as a starting point. By default, the messages that are of the debug level or higher are printed in the `server.log` file and the Worklight Studio console. If you are using a stand-alone server that has its specific logging level definitions, such as WebSphere Application Server or Tomcat, the `development.logging.properties` file is irrelevant. For more information, see “Logging and monitoring mechanisms” on page 400.

Parameters

Table 97. Methods WL.Logger.debug, error, and log parameters

Name	Description
`message`	Mandatory. A string that contains the message to be written to the log file.

Experimental IBM Worklight offline documentation. This document is provided "as-is". In case of issues, refer to the online user documentation.

Return Value

None.

Object WL.Server.configuration

A map that contains all the server properties that are defined in the file `worklight.properties`.

Syntax

```
WL.Server.configuration ["property-name"]
WL.Server.configuration.property-name
```

Both syntaxes are equivalent. When the property name contains a period ('.'), for example`local.IPAddress`, the array index syntax must be used.

Example

```
var addr = WL.Server.configuration["local.IPAddress"];
```

Java server-side API

The IBM Worklight server-side Java API comprises these interfaces.

Interface WorklightAuthenticator

The WorklightAuthenticator interface contains the methods necessary to write a custom authenticator in Java.

A custom authenticator class must implement this interface.

Method changeResponseOnSuccess:

This method is invoked after authentication success. It is used to add data to the response after the authentication is successful.

Syntax

```
Boolean changeResponseOnSuccess (HttpServletRequest request, HttpServletResponse response)
```

Parameters

`request`
Input.

`response`
Output.

Returns

`true`
The response is modified. This value is returned only if the method actually changed the response.

`false`
The response is not modified.

Method clone:

This method creates a deep copy of class members

Experimental IBM Worklight online user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Syntax

```
WorkLightAuthenticator clone()
```

Parameters

None.

Throws

`CloneNotSupportedException`

Method getAuthenticationData:

This method is used by a Login Module to get the credentials collected by an authenticator.

Syntax

```
Map<String, Object> getAuthenticationData()
```

Parameters

None.

Returns

Authentication data as a Map.

Method init:

This method is invoked when the authenticator instance is created.

Syntax

```
void init(Map<String, String> options)
```

Parameters

`Map`
An options map specified in a realm definition in the `authenticationConfig.xml` file.

Throws

MissingConfigurationOptionException

Example

Assume that you have the following realm definition in the `authenticationConfig.xml` file:

```
<realm name="realmForMyApp" loginModule="DatabaseLoginModule">
<className> com.worklight.core.auth.ext.BasicAuthenticator </className>
<parameter name="basic-realm-name" value="My App" >
</realm>
```

The `init` method is called with the options map populated with an entry: `"basic-realm-name"="My App"`.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Method processAuthenticationFailure:

This method is invoked if the Login Module returns a credentials validation failure.

Syntax

```
AuthenticationResult processAuthenticationFailure(HttpServletRequest request, HttpServletResponse
 String errorMessage)
```

Parameters

`request`
: Input.

`response`
: Input.

`Error message`
: Output.

Returns

`FAILURE`
: Authentication failed

`CLIENT_INTERACTION_REQUIRED`
: Additional information is required from the client

Method processRequestAlreadyAuthenticated:

This method is invoked for each request from an already authenticated session. It returns an AuthenticationResult for a request that is already authenticated.

Syntax

```
AuthenticationResult processRequestAlreadyAuthenticated(HttpServletRequest request, HttpServletRes
```

Parameters

`request`
: Input. The client request.

`response`
: Output. The response to the request.

Returns

`REQUEST_NOT_RECOGNIZED`
: The request should be ignored.

`CLIENT_INTERACTION_REQUIRED`
: The response should be returned to the client and the login context should wait for the client request.

`SEND_RESPONSE_TO_CLIENT_ONE_WAY`
: The response should be returned to the client and the login context should not wait for the client request.

`FAILURE`
: The client failed authentication

Experimental IBM Worklight online user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Throws

`java.io.IOException`

`javax.servlet.ServletException`

Method processRequest:

This method processes a request from an unauthenticated session, by sending an authentication request to a specific URL.

Syntax

```
AuthenticationResult processRequest(HttpServletRequest request, HttpServletResponse response, boolea
```

Parameters

`request`
: Input. The Client request.

`response`
: Input. Optionally output, may be modified by the authenticator.

`isAccessToProtectedResource`
: Input.

  `true`
  : The request is to a resource protected by this authenticator and it should not be ignored.

  `false`
  : The request is to a different resource than the one that is protected by this authenticator, and it should be ignored by returning REQUEST_NOT_RECOGNIZED.

Returns

`SUCCESS`
: All login data was received.

`FAILURE`
: The client cannot provide the login data.

`REQUEST_NOT_RECOGNIZED`
: The request does not belong to the login process.

`CLIENT_INTERACTION_REQUIRED`
: The method invocation modified the response.

Throws

`IOException`

`ServletException`

Deprecated method getRequestToProceed:

This method is invoked after the Login Module successfully validates the credentials that are collected by an authenticator.

Note: This method is deprecated since IBM Worklight V5.0.0.3.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

Syntax

```
HttpServletRequest getRequestToProceed(HttpServletRequest request, HttpServletResponse response,
 UserIdentity userIdentity, LoginExtension... loginExtension)
```

Interface WorkLightAuthLoginModule

The WorkLightAuthLoginModule interface contains the methods necessary to write a custom login module in Java.

A custom login module class must implement this interface.

Note: Use this WorkLightAuthLoginModule interface as a replacement of the "Deprecated interface WorkLightLoginModule" on page 307 interface, which is deprecated as of IBM Worklight V5.0.6.

This WorkLightAuthLoginModule interface defines the same methods as the "Deprecated interface WorkLightLoginModule" on page 307 interface, except for the "Method createIdentity" on page 306, which this interface defines as a replacement of the "Deprecated Method createIdenity" on page 308. Except for this specific method name change, the two interfaces offers the same sets of methods and have the same purpose. Use this WorkLightAuthLoginModule interface and its method as you used the "Deprecated interface WorkLightLoginModule" on page 307 interface in previous versions of IBM Worklight.

Method abort:

This method is used to clean up cached data, and is called when the login process is interrupted.

Syntax

```
void abort()
```

Parameters

None.

Method clone:

This method creates a deep copy of class members.

Syntax

```
WorkLightLoginModule clone()
```

Parameters

None.

Returns

WorklightLoginModule

Throws

`CloneNotSupportedException`

Experimental IBM Worklight offline documentation
This document is provided as-is.
In case of issues, refer to the online user documentation.

Method createIdentity:

This method is used to create an authenticated UserIdentity object after the credentials validation succeeds.

Note: Use this createIdentity method (defined in the "Interface WorkLightAuthLoginModule" on page 305) as a replacement of the "Deprecated Method createIdenity" on page 308 method that is defined in the "Deprecated interface WorkLightLoginModule" on page 307. The "Deprecated Method createIdenity" on page 308 and the "Deprecated interface WorkLightLoginModule" on page 307 are both deprecated as of IBM Worklight V5.0.6.

Syntax

```
UserIdentity createIdentity(String realm)
```

Parameters

`realm`

Returns

A UserIdentity object

Method init:

This method is invoked when the Login Module instance is created.

Syntax

```
void init(Map<String, String> options)
```

Parameters

`Map`
: An options map specified in a login module definition in the `authenticationConfig.xml` file.

Throws

MissingConfigurationOptionException

Example

Assume that you have the following realm definition in the `authenticationConfig.xml` file:

```
<realm name="realmForMyApp" loginModule="DatabaseLoginModule">
<className> com.worklight.core.auth.ext.BasicAuthenticator </className>
<parameter name="basic-realm-name" value="My App" >
</realm>
```

The `init` method is called with the options map populated with an entry: `"basic-realm-name"="My App"`.

Method login:

This method is used to validate the credentials that are collected by the authenticator.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

Syntax

```
boolean login(Map<String, Object> authenticationData)
```

Parameters

`authenticationData`

Returns

`true`
  Credential validation passed.

`false`
  Credential validation failed.

Throws

A runtime exception.

Method logout:

This method is used to clean up cached data after a logout.

Syntax

```
void logout()
```

Parameters

None.

Deprecated interface WorkLightLoginModule

The WorkLightLoginModule interface contains the methods necessary to write a custom login module in Java. This interface is deprecated as of IBM Worklight V5.0.6.

Deprecated. A custom login module class must implement this interface.

Note: As of IBM Worklight V5.0.6, this WorkLightLoginModule interface is deprecated and replaced with the "Interface WorkLightAuthLoginModule" on page 305.

Method abort:

This method is used to clean up cached data, and is called when the login process is interrupted.

Syntax

```
void abort()
```

Parameters

None.

Method clone:

This method creates a deep copy of class members.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Syntax

```
WorkLightLoginModule clone()
```

Parameters

None.

Returns

WorklightLoginModule

Throws

`CloneNotSupportedException`

Deprecated Method createIdenity:

This method is used to create an authenticated UserIdentity object after the credentials validation succeeds.

Note: As of IBM Worklight V5.0.6, this method createIdenity is deprecated. The "Deprecated interface WorkLightLoginModule" on page 307 where this method is defined is also deprecated. Instead of this method createIdenity, use the "Method createIdentity" on page 306 that is defined in the "Interface WorkLightAuthLoginModule" on page 305.

Syntax

```
UserIdentity createIdenity(String realm)
```

Parameters

`realm`

Returns

A UserIdentity object

Method init:

This method is invoked when the Login Module instance is created.

Syntax

```
void init(Map<String, String> options)
```

Parameters

`Map`
: An options map specified in a login module definition in the `authenticationConfig.xml` file.

Throws

MissingConfigurationOptionException

Experimental IBM Worklight online user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Example

Assume that you have the following realm definition in the `authenticationConfig.xml` file:

```
<realm name="realmForMyApp" loginModule="DatabaseLoginModule">
<className> com.worklight.core.auth.ext.BasicAuthenticator </className>
<parameter name="basic-realm-name" value="My App" >
</realm>
```

The `init` method is called with the options map populated with an entry: `"basic-realm-name"="My App"`.

Method login:

This method is used to validate the credentials that are collected by the authenticator.

Syntax

```
boolean login(Map<String, Object> authenticationData)
```

Parameters

`authenticationData`

Returns

`true`
Credential validation passed.

`false`
Credential validation failed.

Throws

A runtime exception.

Method logout:

This method is used to clean up cached data after a logout.

Syntax

```
void logout()
```

Parameters

None.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Chapter 6. IBM Worklight Server administration

This topic is intended for IT administrators who want to install and administer IBM Worklight. It describes the server-side components of the IBM Mobile Platform offering and requires administrative knowledge of WebSphere Application Server v7.0+, Apache Tomcat v7+, or Liberty profile v8.5+.

This topic describes the tasks required to install and maintain the IBM Worklight V5.0 production server. More precisely, it explains how to install, configure, optimize, and test the IBM Worklight Server.

It also contains information about installing and configuring database and application server software to support the IBM Worklight database.

For more information about how to size your system, see the following documents:
- Scalability and Hardware Sizing (PDF)
- Hardware Calculator (XLS)

.

Architecture and concepts

You can use the IBM Worklight framework for rapid development of mobile applications for the enterprise.

The framework consists of a development tool (IBM Worklight Studio) and a server (IBM Worklight Server). Each mobile application that is created by using the IBM Worklight Studio connects at run time to an IBM Worklight server.

To obtain maximum benefit from the framework and server administration information that is provided, any development organization must create the necessary files to be deployed on an IBM Worklight Server.

IBM Worklight server uses the following artifacts:

Platform
The Worklight server run time binary files

Project
A set of resources that are deployed on a Worklight server which define the behavior of applications and adapters. Not to be confused with an Eclipse Worklight Project, as used by Worklight Studio.

Application
Server-side metadata and web resources of a mobile application

Adapter
Metadata and code for server-side logic.

All but the Platform are created by using the Worklight Studio running on Eclipse.

The Worklight Server runs on top of an *Application Server*, such as Apache Tomcat or IBM WebSphere Application Server. *Deployment* is the process of installing

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

Worklight artifacts into an Application Server. All four of the Worklight artifacts must be deployed to the Application Server in order for Worklight client applications to connect and run.

IBM Worklight development lifecycle

When you use IBM Worklight Studio within the IBM Worklight framework to develop a mobile application, you produce client and server artifacts, and a `.adapter` file.

Client artifacts
A mobile binary file ready for deployment on a mobile device. For example, an Android `apk` file, or an iPhone `ipa` file. These are usually uploaded to an "App Store" such as the Apple Store or Google Play.

Server artifacts
A `.wlapp` file. Metadata and web resources of an IBM Worklight app deployed on the IBM Worklight Server. Used by the IBM Worklight Server to identify and service mobile Applications.

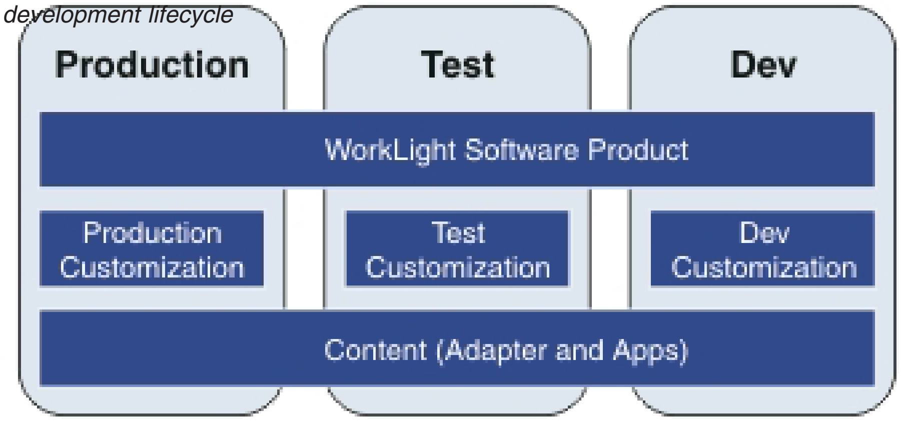
A `.adapter` file
An IBM Worklight adapter deployed on the IBM Worklight Server. This file contains server-side code written by the IBM Worklight developer (for example, retrieve data from a remote database). Adapter code is accessed by IBM Worklight apps via a simple invocation API.

`wlapp` and `adapter` files are referred to in this topic as *content*. These are typically identical between the organization's development, testing, and production environments.

A web archive (WAR) file to be deployed on each IBM Worklight server
This file is also called a `Customization` WAR file and it contains server-specific configurations such as security profiles, server properties, database connectivity, server hostname, and more. `wlapp` and `adapters` use these properties at various stages. The WAR content typically changes during the development lifecycle, that is, development, testing, and production.

The following diagram explains these three terms graphically.

Figure 35. IBM Worklight development lifecycle



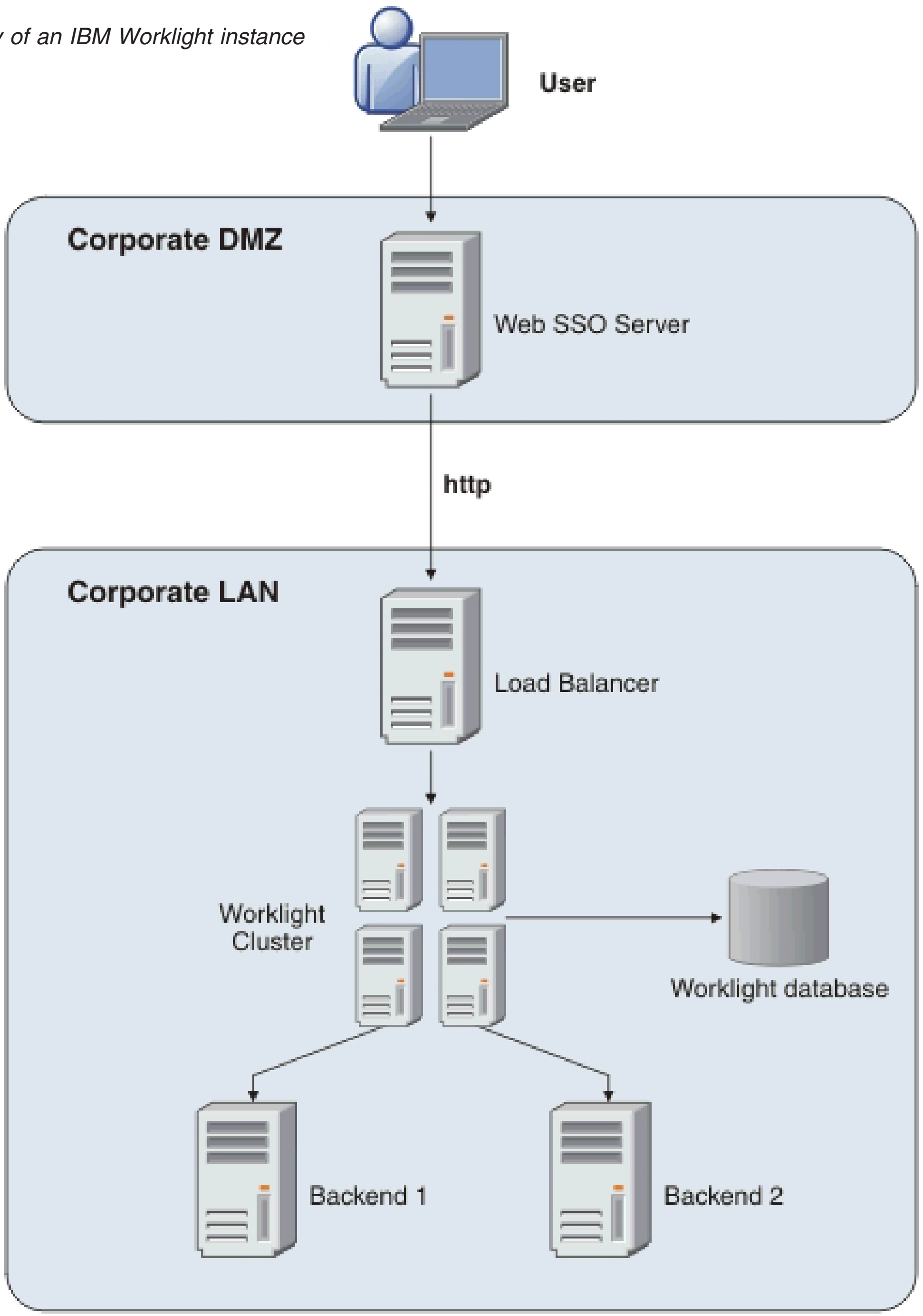
Experimental IBM Worklight offline user documentation. In case of discrepancies, the online documentation is provided as is. Refer to the online user documentation.

Typical topology of an IBM Worklight instance

An IBM Worklight instance uses a particular topology that is typical for organizations with an established extranet infrastructure.

The following figure depicts this topology.

Figure 36. Typical topology of an IBM Worklight instance



Such a topology is based on the following principles:

- IBM Worklight Server is installed in the organization LAN, connecting to various enterprise back-end systems.
- IBM Worklight Server can be clustered for high availability and scalability.

- IBM Worklight Server uses a database for storing push notification information, statistics for reporting and analytics, and storing metadata required by the server at run time. A single instance of the database is shared by all IBM Worklight servers.
- IBM Worklight Server is installed behind a web authentication infrastructure (Web SSO) acting as a reverse proxy and providing SSL.

IBM Worklight Server can be installed in different network configurations, which might include several DMZ layers, reverse proxies, NAT devices, firewalls, high availability components such as load balancers, IP sprayers, clustering, and alike. Some of these components are explained, though for the purpose of this document, a simpler configuration is assumed in which IBM Worklight Server is installed in the DMZ.

Installation

IBM installations are based on an IBM product called “IBM Installation Manager".

Install “IBM Installation Manager 1.5.2 or later" separately prior to installing IBM Worklight.

To ensure correct installation of IBM Worklight Server, please see installation prerequisites.

Installation prerequisites

For smooth installation of IBM Worklight Server, ensure that you have fulfilled all required environment setup and software prerequisites before attempting installation.

You can find a complete list of supported hardware as well as pre-requisite software under: IBM Worklight and IBM Mobile Foundation detailed system requirements.

Important: If you already have a version of IBM Worklight Server installed, you must uninstall it before you install a new version. Failure to do so can result in incomplete installation.

Download the IBM Mobile Platform foundation package from the IBM Passport Advantage site.

Ensure that you have the latest FixPacks for the IBM Worklight product. If you are connected to the Internet during the installation, IBM Installation manager can download the latest fixPacks for you.

The package contains an Install Wizard that guides you through the IBM Worklight Server installation.

The server installation wizard requires an application server and a database. You must choose which database management system and which application server to use. Database options include:
- IBM DB2®
- MySQL
- Oracle
- Apache Derby. Included in the installation image.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Note: Apache Derby is supplied for evaluation and testing purposes only and is not a production-grade database.

Application server options include:

- WebSphere Application Server Liberty Profile. Included in the installation image.
- WebSphere Application Server.
- Apache Tomcat.

The IBM Worklight installation automatically configures an application server and a database, resulting in a fully functional IBM Worklight server.

Creating the DB2 databases

During IBM Worklight installation, the installer can create the necessary databases for you.

About this task

The installer can create the databases for you if you enter the name and password of a user account on the database server that has the DB2 `SYSADM` or `SYSCTRL` privilege, and the account can be accessed through SSH. Otherwise, the database administrator can create the databases for you. For more information, see the DB2 Solution user documentation.

When you manually create the database instances, the IBM Worklight installation requires that the databases have specific names. You should replace the password in the script below with one of your choosing.

Important: You can name your user differently, or set a different password, but ensure that you enter the appropriate user name and password correctly across the DB2 database setup. DB2 has a user name and password length limit of 8 characters for Unix and Linux systems, and 30 characters for Windows.

Procedure

1. Create a system user `wluser` in a DB2 admin group such as `DB2USERS`, using the appropriate commands for your operating system. Give it the password `wluser`. For more information, see the `DB2` documentation and the documentation for your operating system.
2. Open a `DB2` command line processor, with a user that has `SYSADM` or `SYSCTRL` permissions:
   - On Windows systems, click **Start** > **IBM DB2** > **Command Line Processor**
   - On Linux or UNIX systems, navigate to `db2_install/sqllib/bin` and enter `./db2`.
   - Enter the following database manager and SQL statements to create the three databases:

     ```
     CREATE DATABASE WRKLGHT COLLATE USING SYSTEM PAGESIZE 32768
     CONNECT TO WRKLGHT
     GRANT CONNECT ON DATABASE TO USER wluser
     DISCONNECT WRKLGHT
     CREATE DATABASE WLREPORT COLLATE USING SYSTEM PAGESIZE 32768
     CONNECT TO WLREPORT
     GRANT CONNECT ON DATABASE TO USER wluser
     DISCONNECT WLREPORT
     CREATE DATABASE APPCNTR COLLATE USING SYSTEM PAGESIZE 32768
     ```

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

```
CONNECT TO APPCNTR
GRANT CONNECT ON DATABASE TO USER wluser
DISCONNECT APPCNTR
QUIT
```

Creating the MySQL databases

During the IBM® Worklight installation, the installer can create the necessary databases for you.

About this task

The installer can create the databases for you if you enter the name and password of the superuser account. For more information, see Securing the Initial MySQL Accounts on your MySQL database server. Your database administrator can also create the databases for you. When you manually create the database instances, the IBM Worklight installation requires that the databases have specific names. You should replace the password in the script below with one of your choosing.

Procedure

1. Start the MySQL command-line tool.
2. Enter the following commands:
```
CREATE DATABASE WRKLGHT CHARACTER SET utf8 COLLATE utf8_general_ci;
GRANT ALL privileges ON WRKLGHT.* TO 'worklight'@'Worklight-host' IDENTIFIED BY 'worklight';
GRANT ALL privileges ON WRKLGHT.* TO 'worklight'@'localhost' IDENTIFIED BY 'worklight';
Flush privileges;
CREATE DATABASE WLREPORT CHARACTER SET utf8 COLLATE utf8_general_ci;
GRANT ALL privileges ON WLREPORT.* TO 'worklight'@'Worklight-host' IDENTIFIED BY 'worklight';
GRANT ALL privileges ON WLREPORT.* TO 'worklight'@'localhost' IDENTIFIED BY 'worklight';
Flush privileges;
CREATE DATABASE APPCNTR CHARACTER SET utf8 COLLATE utf8_general_ci;
GRANT ALL privileges ON APPCNTR.* TO 'worklight'@'Worklight-host' IDENTIFIED BY 'worklight';
GRANT ALL privileges ON APPCNTR.* TO 'worklight'@'localhost' IDENTIFIED BY 'worklight';
Flush privileges;
```
3. Replace *Worklight-host* with the name of the host on which IBM Worklight runs.

Creating the Oracle databases

During the IBM® Worklight installation, the installer can create the necessary databases for you.

About this task

The installer can create the databases for you if you enter the name and password of the Oracle administrator on the database server, and the account can be accessed through SSH. Otherwise, the database administrator can create the databases for you. When you manually create the database instances, the IBM Worklight installation requires that the databases have specific names. You should replace the password in the script below with one of your choosing.

Procedure

1. Use the Oracle Database Configuration Assistant (DBCA) and follow the steps in the wizard to create a new general-purpose database named `WRKLGHT`:
   a. Use global database name `WRKLGHT_your_domain`, and system identifier (SID) `WRKLGHT`.
   b. On the **Custom Scripts** tab of the step **Database Content**, do not run the SQL scripts, because you must first create a user account.

Experimental IBM Worklight online user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

c. On the **Character Sets** tab of the step **Initialization Parameters**, select **Use Unicode (AL32UTF8) character set and UTF8 - Unicode 3.0 UTF-8 national character set**.

d. Complete the procedure, accepting the default values.

2. Repeat step 1 to create the IBM Worklight reports database. Use global database name `WLREPORT_`*`your_domain`* and SID `WLREPORT`.
3. Repeat step 1 to create the IBM Application Center database. Use global database name `APPCNTR_`*`your_domain`* and SID`APPCNTR`.
4. Create the users either by using `Oracle Database Control`, or by using the `Oracle SQLPlus` command-line interpreter. To create the user for the `WRKLGHT` database by using `Oracle Database Control`:
   a. Connect as `SYSDBA`.
   b. Go to the **Users** page:
      - Click **Server**, then **Users** in the **Security** section.
      - Create a user named `worklight` with the following attributes:
        – Profile: **`DEFAULT`**
        – Authentication: **`password`**
        – Default table space: **`USERS`**
        – Temporary table space: **`TEMP`**
        – Status: **`UNLOCK`**
        – Add role: **`CONNECT`**
        – Add role: **`RESOURCE`**
        – Add system privilege: **`CREATE VIEW`**
        – Add system privilege: **`UNLIMITED TABLESPACE`**

   c. Repeat the previous step to create the user `worklight` for the IBM Worklight report database, `WLREPORT`.
   d. Repeat the previous step to create the user `worklight` for the IBM Application Center database, `APPCNTR`.

   To create the user for all three databases with `Oracle SQLPlus`, enter the following commands:

```
CONNECT system/<system_password>@WRKLGHT
CREATE USER worklight IDENTIFIED BY worklight;
GRANT CONNECT TO worklight;
GRANT RESOURCE TO worklight;
GRANT CREATE VIEW TO worklight;
DISCONNECT;

CONNECT system/<system_password>@WLREPORT
CREATE USER worklight IDENTIFIED BY worklight;
GRANT CONNECT TO worklight;
GRANT RESOURCE TO worklight;
GRANT CREATE VIEW TO worklight;
DISCONNECT;

CONNECT system/<system_password>@APPCNTR
CREATE USER worklight IDENTIFIED BY worklight;
GRANT CONNECT TO worklight;
GRANT RESOURCE TO worklight;
GRANT CREATE VIEW TO worklight;
DISCONNECT;
```

Experimental IBM Worklight offline user documentation
This document is provided "as is"
In case of issue, refer to the online user documentation.

Running IBM Installation Manager

IBM Installation Manager is a tool that you can use to install and maintain your software packages.

IBM Installation Manager helps you install, update, modify, roll back, and uninstall packages on your computer. You can use IBM Installation Manager to install IBM Worklight in several different modes, including single-user and multi-user installation modes.

You can also use silent installations to deploy IBM Worklight to multiple systems, or systems without a GUI interface.

For more information about Installation Manager, see the IBM Installation Manager Information Center at http://pic.dhe.ibm.com/infocenter/install/v1r5/index.jsp.

Single-user versus multi-user installations

You can install IBM Worklight in several different modes.

Administrator installation
: It is an administrator installation when Installation Manager is installed through the `install` command. In this case, it requires administrator privileges to run, and it produces multi-user installations of products.

Single-user installation
: It is a single-user installation when Installation Manager is installed through the `userinst` command. In this case, only the user who installed this copy of Installation Manager can use it.

When, during the installation of IBM Worklight, you choose to install WebSphere Application Server Liberty Profile, this server is either a multi-user or single-user installation, depending on how you installed IBM Installation Manager.

Note: In multi-user mode, different users can run the server in sequence, but not at the same time. There can only be one server process running, because the server is configured to attach to a specific network port.

When, during the installation of IBM Worklight, you choose to use a pre-installed application server, the following constraints regarding user accounts on UNIX apply:

- If the pre-installed application server is owned by a non-root user, you can install IBM Worklight in either of two ways:
  - Through a single-user installation of IBM Installation Manager as the same non-root user.
  - Through an administrator installation of IBM Installation Manager, as root, and afterwards change the owner of all files and directories added or modified during the installation to that user. The result is a single-user installation.
- If the pre-installed application server is owned by root, you can install IBM Worklight only through an administrator installation of IBM Installation Manager; a single-user installation of IBM Installation Manager does not work, because it lacks the necessary privileges.

Silent installation

You can use IBM® Installation Manager to perform silent installation of IBM Worklight on multiple machines or on machines where a GUI interface is not available.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

About this task

Silent installation uses predetermined answers to wizard questions, rather than presenting a GUI that asks the questions and records the answers. Silent installation is useful when:

- You want to install IBM Worklight on a set of machines that are configured in the same way.
- You want to install IBM Worklight on a machine where a GUI is not readily available, for example a production server behind a firewall that prevents the use of `VNC`, `RDP`, `remote X11`, and `ssh -X`.

Important: Silent installation support is one of the major features in the IBM Worklight Server 5.0.5 installer. It applies only to IBM Worklight Server, not to IBM Worklight Studio.

Silent installations are defined by an XML file called a response file. This file contains the necessary data to complete installation operations silently. Silent installations are launched from the command line or a batch file.

You can use IBM Installation Manager to record preferences and installation actions for your response file in user interface mode. Alternatively, you can create a response file manually by using the documented list of response file commands and preferences.

You can use one response file to install, update, or uninstall multiple products.

Using a response file, you can perform almost any action that you can perform by using Installation Manager in wizard mode. For example, with a response file you can specify the location of the repository that contains the package, the package to install, and the features to install for that package. You can also use a response file to modify installed packages, to apply updates, and to apply a license.

Silent installation is described in the IBM Installation Manager documentation, see Working in silent mode.

Procedure

1. Record a response file, by running IBM Installation Manager in wizard mode and with option `record responseFile` on a machine where a GUI is available. For more details, see Record a response file with Installation Manager. The following code example shows a recorded response file:

```
<?xml version="1.0" encoding="UTF-8"?>
<agent-input acceptLicense='true'>
  <server>
    <repository location='http://packages.example.com/ibm/worklight-5.0.5/'/>
  </server>
  <profile id='Worklight' installLocation='/opt/IBM/Worklight'>
    <data key='eclipseLocation' value='/opt/IBM/Worklight'/>
    <data key='user.import.profile' value='false'/>
    <data key='cic.selector.os' value='linux'/>
    <data key='cic.selector.ws' value='gtk'/>
    <data key='cic.selector.arch' value='x86'/>
    <data key='cic.selector.nl' value='en'/>
    <data key='user.writable.data.group' value='admin'/>
    <data key='user.database.db2.port' value='50000'/>
    <data key='user.database.preinstalled' value='true'/>
    <data key='user.database.selection' value='db2'/>
    <data key='user.database.db2.host' value='db2-101.example.com'/>
    <data key='user.database.db2.username' value='wl5test'/>
```

Experimental IBM Worklight offline user documentation.
This document is provided "as-is".
In case of issues refer to the online user documentation.

```
    <data key='user.database.db2.password' value='{xyzzy}7284OFD1KRHW8AC13S'/>
    <data key='user.database.db2.driver' value='/n/databases/drivers/db2-10.1/db2jcc4.jar'/>
    <data key='user.appserver.was85liberty.preinstalled' value='false'/>
    <data key='user.appserver.selection' value='was85liberty'/>
  </profile>
  <install modify='false'>
    <offering id='com.ibm.imp.mfee' version='5.0.5.20121018_0636' profile='Worklight' features='m
  </install>
  <preference name='com.ibm.cic.common.core.preferences.eclipseCache' value='/n/java/rational/SDP
  <preference name='com.ibm.cic.common.core.preferences.connectTimeout' value='30'/>
  <preference name='com.ibm.cic.common.core.preferences.readTimeout' value='45'/>
  <preference name='com.ibm.cic.common.core.preferences.downloadAutoRetryCount' value='0'/>
  <preference name='offering.service.repositories.areUsed' value='true'/>
  <preference name='com.ibm.cic.common.core.preferences.ssl.nonsecureMode' value='false'/>
  <preference name='com.ibm.cic.common.core.preferences.http.disablePreemptiveAuthentication' val
  <preference name='http.ntlm.auth.kind' value='NTLM'/>
  <preference name='http.ntlm.auth.enableIntegrated.win32' value='true'/>
  <preference name='com.ibm.cic.common.core.preferences.preserveDownloadedArtifacts' value='true'
  <preference name='com.ibm.cic.common.core.preferences.keepFetchedFiles' value='false'/>
  <preference name='PassportAdvantageIsEnabled' value='false'/>
  <preference name='com.ibm.cic.common.core.preferences.searchForUpdates' value='false'/>
  <preference name='com.ibm.cic.common.core.preferences.import.enabled' value='true'/>
  <preference name='com.ibm.cic.agent.ui.displayInternalVersion' value='false'/>
  <preference name='com.ibm.cic.common.sharedUI.showErrorLog' value='true'/>
  <preference name='com.ibm.cic.common.sharedUI.showWarningLog' value='true'/>
  <preference name='com.ibm.cic.common.sharedUI.showNoteLog' value='true'/>
</agent-input>
```

2. Modify the response file to take into account differences between the machine on which the response file was created and the target machine. The following code example shows the same response file, edited so that it can be used in step 3.

   Note: This is an example file based on the file in step 1. It might not be suitable for your environment. It is important that you record your own response file, so that it contains the correct parameters for your requirements.

```
<?xml version="1.0" encoding="UTF-8"?>
<agent-input acceptLicense='true'>

  <server>
    <!-- The repositories where Installation Manager can find offerings.
         URLs and absolute file names are accepted; they should point to
         directories that contain a repository.config file. -->
    <repository location='http://packages.example.com/ibm/worklight-5.0.5/'/>
  </server>

  <!-- The declaration of the Installation Manager profile.
       Make sure that the installLocation, if it exists, is empty. -->
  <profile id='Worklight' installLocation='/opt/IBM/Worklight'>

    <!-- The eclipseLocation is not relevant for Worklight Server. -->
    <data key='eclipseLocation' value='/opt/IBM/Worklight'/>
    <data key='user.import.profile' value='false'/>

    <!-- Characteristics of the target machine.
         For the possible values, refer to
         http://help.eclipse.org/indigo/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Fguide%2
    <data key='cic.selector.os' value='linux'/>
    <data key='cic.selector.ws' value='gtk'/>
    <data key='cic.selector.arch' value='x86'/>
    <data key='cic.selector.nl' value='en'/>

    <!-- Database choice. Possible values are derby, db2, mysql, oracle. -->
    <data key='user.database.selection' value='db2'/>
    <data key='user.database.preinstalled' value='true'/>
```

Experimental IBM Worklight offline documentation
This document is provided
In case of issues, refer to the online documentation

```
  <!-- Settings for the database.
       The database user password is obfuscated.
       Make sure that the database driver jar file (including the accompanying
       license file, in the case of DB2) exists on the target machine. -->
  <data key='user.database.db2.host' value='db2-101.example.com'/>
  <data key='user.database.db2.port' value='50000'/>
  <data key='user.database.db2.username' value='wl5test'/>
  <data key='user.database.db2.password' value='{xyzzy}7284OFD1KRHW8AC13S'/>
  <data key='user.database.db2.driver' value='/n/databases/drivers/db2-10.1/db2jcc4.jar'/>

  <!-- Application server choice. -->
  <data key='user.appserver.selection' value='was85liberty'/>
  <data key='user.appserver.was85liberty.preinstalled' value='false'/>

  <!-- Operating system group that shall be allowed to start the server. -->
  <data key='user.writable.data.group' value='admin'/>

</profile>

<!-- Define what Installation Manager should install. -->
<install modify='false'>
  <!-- You can omit the 'version' and 'installFixes' attributes. -->
  <offering id='com.ibm.imp.mfee' version='5.0.5.20121018_0636' profile='Worklight' features
</install>

<!-- The Installation Manager preferences don't need to be transferred to the
     target machine. -->

</agent-input>
```

3. Install IBM Worklight using the response file on the target machine, as described inInstall a package silently by using a response file.

Completing the installation

When installation is complete, you must restart the web application server in certain cases.

You must restart the web application server in the following circumstances:

- If you are using WebSphere Application Server, with DB2 as database type.
- If you are using WebSphere Application Server Liberty Profile or Apache Tomcat.
- After you upgraded from a previous version of IBM Worklight Server.

If you are using WebSphere Application Server Network Deployment and chose an installation through the deployment manager:

- You must restart the servers that were running during the installation and on which the Worklight Server applications were installed.

  To restart these servers with the deployment manager console, select **Applications** > **Application Types** > **WebSphere enterprise applications** > **IBM_Worklight_Console** > **Target specific application status**.
- You do not have to restart the deployment manager or the node agents.

Manually configuring the databases

In some cases, you may want to re-configure IBM Worklight Server so that it uses a different database from the one that was specified during installation of IBM Worklight Server. The way you do this depends on the type of database and on the kind of application server, as explained in the following topics.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

Configuring the DB2 databases manually

You configure the DB2 databases manually by creating the databases, creating the database tables, and then configuring the relevant application server to use this database setup.

Procedure

1. Create the databases. This step is described in “Creating the DB2 databases” on page 315
2. Create the tables in the databases. This step is described in “Setting up your DB2 databases manually”
3. Configure the application server to use this database setup. Go to one of the following topics:
   - “Configuring Liberty Profile for DB2 manually” on page 324
   - “Configuring WebSphere Application Server for DB2 manually” on page 325
   - “Configuring Apache Tomcat for DB2 manually” on page 326

Setting up your DB2 databases manually:

IBM Worklight V5.0.5. automatically configures your databases. However, if you are using an older version of IBM Worklight, or if you are experiencing problems with the automatic configuration process, you must manually set up and configure your DB2 database.

About this task

Complete the following procedure to set up your DB2 database.

Procedure

1. Create the database schema:
   a. Create a system user, **`worklight`**, in a DB2 admin group such as **`DB2USERS`**, using the appropriate commands for your operating system. Give it the password **`worklight`**. For more information, see the DB2 documentation and the documentation for your operating system.

      Important: You can name your user differently, or set a different password, but ensure that you enter the appropriate user name and password correctly across the DB2 database setup. DB2 has a user name and password length limit of 8 characters for Unix and Linux systems, and 30 characters for Windows.

   b. Open a DB2 command line processor, with a user that has **`SYSADM`** or **`SYSCTRL`** permissions:

      On Windows systems, click **Start** > **IBM DB2** > **Command Line Processor**.

      On Linux or UNIX systems, navigate to `db2_install/sqllib/bin` and enter `./db2`.

   c. Enter the following database manager and SQL statements to create a database called **`WRKLGHT`**:

      ```
      CREATE DATABASE WRKLGHT COLLATE USING SYSTEM PAGESIZE 32768
      CONNECT TO WRKLGHT
      GRANT CONNECT ON DATABASE TO USER worklight
      QUIT
      ```

      Where **`worklight`** is the name of the system user that you have previously created. If you have defined a different user name, replace **`worklight`** accordingly.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

d. Invoke **db2** with the following commands to create the **WRKLGHT** tables:

```
db2 CONNECT TO WRKLGHT USER worklight USING worklight
db2 -vf <worklight_install_dir>/WorklightServer/databases/create-worklight-db2.sql -t
```

Where **worklight** after **USER** is the name of the system user with "CONNECT" access to the **WRKLGHT** database that you have previously created, and **worklight** after **USING** is this user's password. If you have defined either a different user name, or a different password, or both, replace **worklight** accordingly.

DB2 has a user name and password length limit of 8 characters for Unix and Linux systems, and 30 characters for Windows.

Important: If you do not specify the user name and password, DB2 assumes that the user is the current user, and creates the tables by using this current user's schema. If the current user differs from the settings in **Worklight**, then the current user is denied access to the tables in the database.

e. Enter the following database manager and SQL statements to create a database called **WLREPORT**:

```
CREATE DATABASE WLREPORT COLLATE USING SYSTEM PAGESIZE 32768
CONNECT TO WLREPORT
GRANT CONNECT ON DATABASE TO USER worklight
QUIT
```

f. Invoke DB2 with the following commands to create the **WLREPORT** tables:

```
db2 CONNECT TO WLREPORT
db2 -vf
<worklight_install_dir>/WorklightServer/databases/create-worklightreports-db2.sql -t
```

g. Enter the following database manager and SQL statements to create a database called **APPCNTR**:

```
CREATE DATABASE APPCNTR COLLATE USING SYSTEM PAGESIZE 32768
CONNECT TO APPCNTR
GRANT CONNECT ON DATABASE TO USER worklight
QUIT
```

h. Invoke DB2 with the following commands to create the **APPCNTR** tables:

```
db2 CONNECT TO APPCNTR
db2 -vf
<worklight_install_dir>/ApplicationCenter/databases/create-appcenter-db2.sql -t
```

2. Create a `worklight.properties` file. Give the file the following contents, depending on whether you are using JDBC or JNDI.

a. JDBC version:

```
wl.db.jndi.name=
wl.db.type=DB2
wl.db.url=jdbc:db2://server:50000/WRKLGHT
wl.reports.db.type=DB2
wl.reports.db.url=jdbc:db2://server:50000/WLREPORT
wl.db.username=worklight
wl.db.password=worklight
reports.exportRawData=true
```

Where **worklight** after **wl.db.username=** is the name of the system user with "CONNECT" access to the **WRKLGHT** database that you have previously created, and **worklight** after **wl.db.password=** is this user's password. If you have defined either a different user name, or a different password, or both, replace **worklight** accordingly.

DB2 has a user name and password length limit of 8 characters for Unix and Linux systems, and 30 characters for Windows.

b. JNDI version

```
wl.db.jndi.name=jdbc/WorklightDS wl.reports.db.jndi.name=jdbc/WorklightReportsDS wl.db.type
```

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues refer to the online user documentation.

Where **worklight** after **wl.db.username=** is the name of the system user with "CONNECT" access to the **WRKLGHT** database that you have previously created, and **worklight** after **wl.db.password=** is this user's password. If you have defined either a different user name, or a different password, or both, replace **worklight** accordingly.

DB2 has a user name and password length limit of 8 characters for Unix and Linux systems, and 30 characters for Windows.

3. Replace the `WEB-INF/classes/conf.worklight.properties` file in `worklight.war` with the file you created in the previous step.

Configuring Liberty Profile for DB2 manually:

IBM Worklight V5.0.5. automatically configures your databases. However, if you are using an older version of IBM Worklight, or if you are experiencing problems with the automatic configuration process for the Liberty Application Server, you must manually set up and configure your DB2 database and then the Liberty Application Server for DB2.

About this task

Complete the DB2 Database Setup procedure before continuing.

Procedure

1. Add the DB2 JDBC driver JAR file (download it from DB2 JDBC Driver Versions, or fetch it from the directory `DB2_INSTALL_DIR/java` on the DB2 server) to `$LIBERTY_HOME/wlp/usr/shared/resources/db2`. If that directory does not exist, create it.
2. Configure the data source in the `$LIBERTY_HOME/wlp/usr/servers/worklightServer/server.xml` file as follows:

```
<!-- Declare the jar files for DB2 access through JDBC. -->
<library id="DB2Lib">
    <fileset dir="${shared.resource.dir}/db2" includes="*.jar"/>
</library>

<!-- Declare the IBM Worklight Server database. Used through property wl.db.jndi.name.
     If you change this declaration to refer to a different kind of database,
     you have to update the property wl.db.type in the file worklight.properties
     inside the file worklight.war. -->
<dataSource id="WorklightDS" jndiName="jdbc/WorklightDS">
    <jdbcDriver libraryRef="DB2Lib"/>
    <properties.db2.jcc databaseName="WRKLGHT"
        serverName="db2server" portNumber="50000"
        user="worklight" password="worklight"/>
</dataSource>

<!-- Declare the IBM Worklight Server Reports database. Used through property wl.reports.db.jndi.
     If you change this declaration to refer to a different kind of database,
     you have to update the property wl.reports.db.type in the file worklight.properties
     inside the file worklight.war. -->
<dataSource id="WorklightReportsDS" jndiName="jdbc/WorklightReportsDS">
    <jdbcDriver libraryRef="DB2Lib"/>
    <properties.db2.jcc databaseName="WLREPORT"
        serverName="db2server" portNumber="50000"
        user="worklight" password="worklight"/>
</dataSource>

<!-- Declare the IBM Application Center database. -->
<dataSource id="AppCenterDS" jndiName="jdbc/AppCenterDS">
    <jdbcDriver libraryRef="DB2Lib"/>
```

Experimental IBM Worklight offline user documentation.
This document is provided "as is".
In case of issues refer to the online user documentation.

```
    <properties.db2.jcc databaseName="APPCNTR"
        serverName="db2server" portNumber="50000"
        user="worklight" password="worklight"/>
</dataSource>
```

where **worklight** after **user=** is the name of the system user with "CONNECT" access to the **WRKLGHT** database that you have previously created, and **worklight** after **password=** is this user's password. If you have defined either a different user name, or a different password, or both, replace **worklight** accordingly. Also, replace **db2server** with the host name of your DB2 server (for example, localhost, if it is on the same machine).

DB2 has a user name and password length limit of 8 characters for Unix and Linux systems, and 30 characters for Windows.

Configuring WebSphere Application Server for DB2 manually:

IBM Worklight V5.0.5. automatically configures your databases. However, if you are using an older version of IBM Worklight, or if you are experiencing problems with the automatic configuration process for the WebSphere Application Server, you must manually set up and configure your DB2 database and then the WebSphere Application Server for DB2.

About this task

Complete the DB2 Database Setup procedure before continuing.

Procedure

1. Add the DB2 JDBC driver JAR file (download it from DB2 JDBC Driver Versions, or fetch it from the directory DB2_INSTALL_DIR/java on the DB2 server) to WAS_INSTALL_DIR/optionalLibraries/IBM/Worklight/5.0/db2. If that directory does not exist, create it.
2. Set up the JDBC provider:
   a. In the WebSphere Application Server console, click **Resources** > **JDBC** > **JDBC Providers** > **New**.
   b. Set the scope of the JDBC connection to **Node level**.
   c. Set **Database type** to DB2.
   d. Set **Provider type** to DB2 Universal JDBC Driver Provider.
   e. Set **Implementation Type** to Connection pool data source.
   f. Set **Name** to DB2 Universal JDBC Driver Provider.
   g. Click **Next**.
   h. Set the class path to the set of jar files in the directory WAS_INSTALL_DIR/optionalLibraries/IBM/Worklight/5.0/db2, one per line.
   i. Do not set **Native library path**.
   j. Click **Next**.
   k. Click **Finish**.
   l. The JDBC provider is created.
   m. Click **Save**.
3. Create a data source for the IBM Worklight database:
   a. Select the new JDBC provider and click **Data Source**.
   b. Click **New** to create a data source.
   c. Set the **Data source name** to Worklight Database.
   d. Set **JNDI Name** to jdbc/WorklightDS.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

   e. Click **Next**.
   f. Enter properties for the datasource: For example, **Driver type**: `4`, **Database Name**: `WRKLGHT`, **Server name**: `localhost`, **Port number**: `50000` (default). Leave "Use this data source in (CMP)" checked;
   g. Click **Next**.
   h. Create `JAAS-J2C` authentication data, specifying the `DB2` user name and password for **Container Connection**.
   i. Select the component-managed authentication alias that you created.
   j. Click **Next** and **Finish**.
   k. Click **Save**.
4. Create a data source for the IBM Worklight reports database:
   a. Select the new `JDBC` provider and click **Data Source**.
   b. Click **New** to create a data source.
   c. Set the **Data source name** to `Worklight Reports Database`.
   d. Set `JNDI Name` to `jdbc/WorklightReportsDS`.
   e. Click **Next**.
   f. Select the component-managed authentication alias that you created.
   g. Click **Next** and **Finish**.
5. Create a data source for the IBM Application Center database:
   a. Select the new `JDBC` provider and click **Data Source**.
   b. Click **New** to create a data source.
   c. Set the **Data source name** to `Application Center Database`.
   d. Set `JNDI Name` to `jdbc/AppCenterDS`.
   e. Click **Next**.
   f. Select the component-managed authentication alias that you created.
   g. Click **Next** and **Finish**.
6. Test the data source connection by selecting each **Data Source** and clicking **Test Connection**.

Configuring Apache Tomcat for DB2 manually:

IBM Worklight V5.0.5. automatically configures your databases. However, if you are using an older version of IBM Worklight, or if you are experiencing problems with the automatic configuration process for the Apache Tomcat Server, you must manually set up and configure your DB2 database and then the Apache Tomcat Server for DB2.

About this task

Complete the DB2 Database Setup procedure before continuing.

Procedure

1. Add the `DB2 JDBC` driver `JAR` file (download it from DB2 JDBC Driver Versions, or fetch it from the directory `DB2_INSTALL_DIR/java` on the DB2 server) to `$TOMCAT_HOME/lib`.
2. Update the `$TOMCAT_HOME/conf/context.xml` file as follows:

```
<Context>
    ...
    <Resource auth="Container"
        driverClassName="com.ibm.db2.jcc.DB2Driver"
```

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues refer to the online user documentation.

```
            name="jdbc/WorklightDS"
            password="worklight"
            username="worklight"
            type="javax.sql.DataSource"
            url="jdbc:db2://server:50000/WRKLGHT"/>
        <Resource auth="Container"
            driverClassName="com.ibm.db2.jcc.DB2Driver"
            name="jdbc/WorklightReportsDS"
            password="worklight"
            username="worklight"
            type="javax.sql.DataSource"
            url="jdbc:db2://server:50000/WLREPORT"/>
        <Resource auth="Container"
            driverClassName="com.ibm.db2.jcc.DB2Driver"
            name="jdbc/AppCenterDS"
            password="worklight"
            username="worklight"
            type="javax.sql.DataSource"
            url="jdbc:db2://server:50000/APPCNTR"/>
        ...
    </Context>
```

Where **worklight** after **user=** is the name of the system user with "CONNECT" access to the **WRKLGHT** database that you have previously created, and **worklight** after **password=** is this user's password. If you have defined either a different user name, or a different password, or both, replace **worklight** accordingly.

DB2 has a user name and password length limit of 8 characters for Unix and Linux systems, and 30 characters for Windows.

3. Add references to the datasources:
   a. For the JDBC Version, modify the `Worklight.properties` file as follows:

      ```
      wl.db.url=jdbc:db2://<server>:50000/WRKLGHT
      wl.db.type=DB2
      wl.reports.db.type=DB2
      wl.reports.db.url=jdbc:db2://<server>:50000/WLREPORT
      ```

   b. For the JNDI Version, update the `$TOMCAT_HOME/conf/web.xml` file as follows:

      ```
      <web-app>
          ....
          ....
          <resource-ref>
              <res-ref-name>jdbc/WorklightDS</res-ref-name>
              <res-type>javax.sql.DataSource</res-type>
              <res-auth>Container</res-auth>
          </resource-ref>
          <resource-ref>
              <res-ref-name>jdbc/WorklightReportsDS</res-ref-name>
              <res-type>javax.sql.DataSource</res-type>
              <res-auth>Container</res-auth>
          </resource-ref>
      </web-app>
      ```

4. Modify the `worklight.properties` file as follows:

   ```
   wl.db.jndi.name=java:comp/env/jdbc/WorklightDS
   wl.db.type=DB2
   wl.reports.db.jndi.name=java:comp/env/jdbc/WorklightReportsDS
   ```

Configuring the Apache Derby databases manually

You configure the Apache Derby databases manually by creating the databases and database tables, and then configuring the relevant application server to use this database setup.

Experimental IBM Worklight online user documentation
This document is provided "as is".
In case of issues refer to the online user documentation.

Procedure

1. Create the databases and the tables within them. This step is described in "Setting up your Apache Derby databases manually"
2. Configure the application server to use this database setup. Go to one of the following topics:
   - "Configuring Liberty Profile for Derby manually" on page 329
   - "Configuring WebSphere Application Server for Derby manually" on page 329
   - "Configuring Apache Tomcat for Derby manually" on page 330

Setting up your Apache Derby databases manually:

IBM Worklight V5.0.5. automatically configures your databases. However, if you are using an older version of IBM Worklight, or if you are experiencing problems with the automatic configuration process, you must manually set up and configure your Apache Derby database.

About this task

Complete the following procedure to set up your `Apache Derby` database.

Procedure

1. Create the database schema:
   a. In the location where you want the database to be created, run `ij.bat` on Windows systems or `ij.sh` on UNIX and Linux systems. The script displays `ij` version 10.4.

      Note: The `ij` program is part of Apache Derby. If you do not already have it installed, you can download it from Apache Derby: Downloads.
   b. At the command prompt, enter the following commands:
```
connect 'jdbc:derby:WRKLGHT;user=WORKLIGHT;create=true';
run '<worklight_install_dir>/WorklightServer/databases/create-worklight-derby.sql';
connect 'jdbc:derby:WLREPORT;user=WORKLIGHT;create=true';
run '<worklight_install_dir>/WorklightServer/databases/create-worklightreports-derby.sql';
connect 'jdbc:derby:APPCNTR;user=APPCENTER;create=true';
run '<worklight_install_dir>/ApplicationCenter/databases/create-appcenter-derby.sql';
quit;
```
2. Create a `worklight.properties` file. Give the file the following contents, depending on whether you are using `JDBC` or `JNDI`.
   a. JDBC version:
```
wl.db.jndi.name=
wl.db.type=DERBY
wl.db.url=path_to_db/WRKLGHT
wl.reports.db.type=DERBY
wl.reports.db.url=path_to_db/WLREPORT
wl.db.username=WORKLIGHT
wl.db.password=
reports.exportRawData=true
```
   b. JNDI version:
```
wl.db.jndi.name=jdbc/WorklightDS
wl.db.type=DERBY
wl.reports.db.jndi.name=jdbc/WorklightReportsDS
wl.reports.db.type=DERBY
wl.db.username=WORKLIGHT
wl.db.password=
```

Experimental IBM Worklight offline user documentation.
This document is provided "as is".
In case of issues, refer to the online user documentation.

3. Replace the `WEB-INF/classes/conf/worklight.properties` file in `worklight.war` with the file you created in the previous step.

Configuring Liberty Profile for Derby manually:

IBM Worklight V5.0.5. automatically configures your databases. However, if you are using an older version of IBM Worklight, or if you are experiencing problems with the automatic configuration process for Liberty Profile on Derby, you must manually set up and configure your Apache Derby database and then the Liberty Profile for Derby.

About this task

Complete the Apache Derby database setup procedure before continuing.

Procedure

Configure the data source in the `$LIBERTY_HOME/wlp/usr/servers/worklightServer/server.xml` file as follows:

```
<library id="derbyLib">
    <fileset dir="C:/Drivers/derby" includes="derby.jar" />
</library>
<dataSource id="WorklightDS" jndiName="jdbc/WorklightDS" statementCacheSize="10">
    <jdbcDriver libraryRef="DerbyLib"
        javax.sql.ConnectionPoolDataSource="org.apache.derby.jdbc.EmbeddedConnectionPoolDataSource
    <properties.derby.embedded databaseName="DERBY_DATABASES_DIR/WRKLGHT" user="WORKLIGHT"
        shutdownDatabase="false" connectionAttributes="upgrade=true"/>
    <connectionManager connectionTimeout="180"
        maxPoolSize="10" minPoolSize="1"
        reapTime="180" maxIdleTime="1800"
        agedTimeout="7200" purgePolicy="EntirePool" />
</dataSource>
<dataSource id="WorklightReportsDS" jndiName="jdbc/WorklightReportsDS" statementCacheSize="10">
    <jdbcDriver libraryRef="DerbyLib"
        javax.sql.ConnectionPoolDataSource="org.apache.derby.jdbc.EmbeddedConnectionPoolDataSource
    <properties.derby.embedded databaseName="DERBY_DATABASES_DIR/WLREPORT" user="WORKLIGHT"
        shutdownDatabase="false" connectionAttributes="upgrade=true"/>
    <connectionManager connectionTimeout="180"
        maxPoolSize="10" minPoolSize="1"
        reapTime="180" maxIdleTime="1800"
        agedTimeout="7200" purgePolicy="EntirePool" />
</dataSource>
<dataSource id="AppCenterDS" jndiName="jdbc/AppCenterDS" statementCacheSize="10">
    <jdbcDriver libraryRef="DerbyLib"
        javax.sql.ConnectionPoolDataSource="org.apache.derby.jdbc.EmbeddedConnectionPoolDataSource
    <properties.derby.embedded databaseName="DERBY_DATABASES_DIR/APPCNTR" user="APPCENTER"
        shutdownDatabase="false" connectionAttributes="upgrade=true"/>
    <connectionManager connectionTimeout="180"
        maxPoolSize="10" minPoolSize="1"
        reapTime="180" maxIdleTime="1800"
        agedTimeout="7200" purgePolicy="EntirePool"/>
</dataSource>
```

Configuring WebSphere Application Server for Derby manually:

IBM Worklight V5.0.5. automatically configures your databases. However, if you are using an older version of IBM Worklight, or if you are experiencing problems with the automatic configuration process for the WebSphere Application Server, you must manually set up and configure your Apache Derby database and then the WebSphere Application Server for Derby.

About this task

Complete the Apache Derby database setup procedure before continuing.

Procedure

1. Add the Derby JAR file from `WORKLIGHT_INSTALL_DIR/ApplicationCenter/tools/lib/derby.jar` to `WAS_INSTALL_DIR/optionalLibraries/IBM/Worklight/5.0/derby`. If that directory does not exist, create it.
2. Set up the JDBC provider.
   a. In the WebSphere Application Server console, click **Resources** > **JDBC** > **JDBC Providers** and select the **Derby JDBC Provider**.
   b. Set Name to **Derby JDBC Provider**.
   c. Set **Implementation class name** to `org.apache.derby.jdbc.EmbeddedConnectionPoolDataSource`.
   d. Set the Class path to `WAS_INSTALL_DIR/optionalLibraries/IBM/Worklight/5.0/derby/derby.jar`.
3. Set up the data source for the IBM Worklight database.
   a. In the WebSphere Application Server console, click **Resources** > **JDBC** > **Data sources** and select the **Derby JDBC Driver datasource**.
   b. Set **JNDI name** to `jdbc/WorklightDS`.
   c. In the "Common and required data source properties" pane, set **Database name** to `WRKLGHT`.
   d. Click **Resources** > **JDBC** > **Data sources** > **Driver DataSource** > **Custom properties**.
   e. Set **user = WORKLIGHT**.
4. Set up the data source for the IBM Worklight report database.
   a. In the WebSphere Application Server console, click **Resources** > **JDBC** > **Data sources** and select the **Derby JDBC Driver datasource**.
   b. Set **JNDI name** to `jdbc/WorklightReportsDS`.
   c. In the "Common and required data source properties" pane, set **Database name** to `WLREPORT`.
   d. Click **Resources** > **JDBC** > **Data sources** > **Driver DataSource** > **Custom properties**.
   e. Set **user = WORKLIGHT**.
5. Set up the data source for the IBM Application Center database.
   a. In the WebSphere Application Server console, click **Resources** > **JDBC** > **Data sources** and select the **Derby JDBC Driver datasource**.
   b. Set **JNDI name** to `jdbc/AppCenterDS`.
   c. In the "Common and required data source properties" pane, set **Database name** to `APPCNTR`.
   d. Click **Resources** > **JDBC** > **Data sources** > **Driver DataSource** > **Custom properties**.
   e. Set **user = APPCENTER**.

Configuring Apache Tomcat for Derby manually:

IBM Worklight V5.0.5. automatically configures your databases. However, if you are using an older version of IBM Worklight, or if you are experiencing problems

Experimental IBM Worklight offline user documentation. This document is provided as is; In case of issues, refer to the online user documentation.

with the automatic configuration process for the Apache Tomcat Server, you must manually set up and configure your Apache Derby database and then the Apache Tomcat Server for Derby.

About this task

Complete the Apache Derby database setup procedure before continuing.

Procedure

1. Add the Derby JAR file from WORKLIGHT_INSTALL_DIR/ApplicationCenter/tools/lib/derby.jar to the directory $TOMCAT_HOME/lib.
2. Update the $TOMCAT_HOME/conf/context.xml file as follows:

```
<Context>
    ...
    <Resource auth="Container"
        driverClassName="org.apache.derby.jdbc.EmbeddedDriver"
        name="jdbc/WorklightDS"
        password="WORKLIGHT"
        username=""
        type="javax.sql.DataSource"
        url="jdbc:derby:DERBY_DATABASES_DIR/WRKLGHT"/>
    <Resource auth="Container"
        driverClassName="org.apache.derby.jdbc.EmbeddedDriver"
        name="jdbc/WorklightReportsDS"
        password="WORKLIGHT"
        username=""
        type="javax.sql.DataSource"
        url="jdbc:derby:DERBY_DATABASES_DIR/WLREPORT"/>
    <Resource auth="Container"
        driverClassName="org.apache.derby.jdbc.EmbeddedDriver"
        name="jdbc/AppCenterDS"
        password="APPCENTER"
        username=""
        type="javax.sql.DataSource"
        url="jdbc:derby:DERBY_DATABASES_DIR/APPCNTR"/>
    ...
</Context>
```

3. Add references to the datasources.
   a. For JDBC, modify the worklight.properties file as follows:

```
wl.db.type=DERBY
wl.db.url=path_to_db/WRKLGHT
wl.reports.db.type=DERBY
wl.reports.db.url=path_to_db/WLREPORT
wl.db.username=worklight
wl.db.password=worklight
reports.exportRawData=true
```

   b. For JNDI, update the $TOMCAT_HOME/conf/web.xml file as follows:

```
<web-app>
    ....
    ....
    <resource-ref>
        <res-ref-name>jdbc/WorklightDS</res-ref-name>
        <res-type>javax.sql.DataSource</res-type>
        <res-auth>Container</res-auth>
    </resource-ref>
    <resource-ref>
        <res-ref-name>jdbc/WorklightReportsDS</res-ref-name>
        <res-type>javax.sql.DataSource</res-type>
        <res-auth>Container</res-auth>
    </resource-ref>
</web-app>
```

Experimental IBM Worklight online user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

4. In the `worklight.properties` file, update `wl.db.jndi.name` as follows:

```
wl.db.jndi.name=java:comp/env/jdbc/WorklightDS
wl.reports.db.jndi.name=java:comp/env/jdbc/WorklightReportsDS
```

Configuring the MySQL databases manually

You configure the MySQL databases manually by creating the databases, creating the database tables, and then configuring the relevant application server to use this database setup.

Procedure

1. Create the databases. This step is described in "Creating the MySQL databases" on page 316
2. Create the tables in the databases. This step is described in "Setting up your MySQL databases manually"
3. Configure the application server to use this database setup. Go to one of the following topics:
   - "Configuring Liberty Profile for MySQL manually" on page 333
   - "Configuring WebSphere Application Server for MySQL manually" on page 334
   - "Configuring Apache Tomcat for MySQL manually" on page 335

Setting up your MySQL databases manually:

IBM Worklight V5.0.5. automatically configures your databases. However, if you are using an older version of IBM Worklight, or if you are experiencing problems with the automatic configuration process, you must manually set up and configure your MySQL databases.

About this task

Complete the following procedure to set up your `MySQL` databases.

Procedure

1. Create the database schema.
   a. Run a `MySQL` command-line client with options `-u root`.
   b. Enter the following commands:

```
CREATE DATABASE WRKLGHT CHARACTER SET utf8 COLLATE utf8_general_ci;
GRANT ALL privileges ON WRKLGHT.* TO 'worklight'@'Worklight-host' IDENTIFIED BY 'worklight';
GRANT ALL privileges ON WRKLGHT.* TO 'worklight'@'localhost' IDENTIFIED BY 'worklight';
Flush privileges;
CREATE DATABASE WLREPORT CHARACTER SET utf8 COLLATE utf8_general_ci;
GRANT ALL privileges ON WLREPORT.* TO 'worklight'@'Worklight-host'IDENTIFIED BY 'worklight';
GRANT ALL privileges ON WLREPORT.* TO 'worklight'@'localhost' IDENTIFIED BY 'worklight';
Flush privileges;
CREATE DATABASE APPCNTR CHARACTER SET utf8 COLLATE utf8_general_ci;
GRANT ALL privileges ON APPCNTR.* TO 'worklight'@'Worklight-host'IDENTIFIED BY 'worklight';
GRANT ALL privileges ON APPCNTR.* TO 'worklight'@'localhost' IDENTIFIED BY 'worklight';
Flush privileges;

USE WRKLGHT;
SOURCE <worklight_install_dir>/WorklightServer/databases/create-worklight-mysql.sql;

USE WLREPORT;
SOURCE <worklight_install_dir>/WorklightServer/databases/create-worklightreports-mysql.sql;

USE WLREPORT;
SOURCE <worklight_install_dir>/ApplicationCenter/databases/create-appcenter-mysql.sql;
```

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

Where **worklight** before the @ sign is the user name, **worklight** after **IDENTIFIED BY** is its password, and **Worklight-host** is the name of the host on which IBM Worklight runs.

2. Create a `worklight.properties` file. Give the file the following contents, depending on whether you are using JDBC or JNDI:
   a. JDBC version:
   ```
   wl.db.type=MYSQL
   wl.db.url=jdbc:mysql://localhost:3306/WRKLGHT
   wl.reports.db.type=MYSQL
   wl.reports.db.url=jdbc:mysql://localhost:3306/WLREPORT
   wl.db.username=worklight
   wl.db.password=worklight
   reports.exportRawData=true
   ```
   b. JNDI version:
   ```
   wl.db.jndi.name=jdbc/WorklightDS
   wl.db.type=MYSQL
   wl.reports.db.jndi.name=jdbc/WorklightReportsDS
   wl.reports.db.type=MYSQL
   reports.exportRawData=true
   ```
3. Add the following property to your MySQL option file: `max_allowed_packet=16M`

   For more information about option files, see theMySQL documentation at MySQL.
4. Replace the `WEB-INF/classes/conf.worklight.properties` file in `worklight.war` with the file you created in the previous step.

   If you do not already have the MySQL driver Connector/J, download it from Download Connector/J. The driver is supplied as a compressed file. Extract the `.jar` file from it.

Configuring Liberty Profile for MySQL manually:

IBM Worklight V5.0.5. automatically configures your databases. However, if you are using an older version of IBM Worklight, or if you are experiencing problems with the automatic configuration process for Liberty Profile on MySQL, you must manually set up and configure your MySQL database and then the Liberty Profile for MySQL.

About this task

Complete the MySQL database setup procedure before continuing.

Procedure

1. Add the MySQL JDBC driver JAR file to `$LIBERTY_HOME/wlp/usr/shared/resources/mysql`. If that directory does not exist, create it.
2. Configure the data source in the `$LIBERTY_HOME/wlp/usr/servers/worklightServer/server.xml` file as follows:
```
<!-- Declare the jar files for MySQL access through JDBC. -->
<library id="MySQLLib">
    <fileset dir="${shared.resource.dir}/mysql" includes="*.jar"/>
</library>

<!-- Declare the IBM Worklight Server database. Used through property wl.db.jndi.name.
     If you change this declaration to refer to a different kind of data base,
     you have to update the property wl.db.type in the file worklight.properties
     inside the file worklight.war. -->
<dataSource id="WorklightDS" jndiName="jdbc/WorklightDS">
    <jdbcDriver libraryRef="MySQLLib"/>
    <properties databaseName="WRKLGHT"
```

Experimental IBM Worklight online user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

```
        serverName="mysqlserver" portNumber="3306"
        user="worklight" password="worklight"/>
</dataSource>

<!-- Declare the IBM Worklight Server Reports database. Used through property wl.reports.db.jndi.
     If you change this declaration to refer to a different kind of data base,
     you have to update the property wl.reports.db.type in the file worklight.properties
     inside the file worklight.war. -->
<dataSource id="WorklightReportsDS" jndiName="jdbc/WorklightReportsDS">
    <jdbcDriver libraryRef="MySQLLib"/>
    <properties databaseName="WLREPORT"
        serverName="mysqlserver" portNumber="3306"
        user="worklight" password="worklight"/>
</dataSource>

<!-- Declare the IBM Application Center database. -->
<dataSource id="AppCenterDS" jndiName="jdbc/AppCenterDS">
    <jdbcDriver libraryRef="MySQLLib"/>
    <properties databaseName="APPCNTR"
        serverName="mysqlserver" portNumber="3306"
        user="worklight" password="worklight"/>
</dataSource>
```

where **worklight** after **user=** is the user name, **worklight** after **password=** is this user's password, and **mysqlserver** is the host name of your MySQL server (for example, localhost, if it is on the same machine).

Configuring WebSphere Application Server for MySQL manually:

IBM Worklight V5.0.5. automatically configures your databases. However, if you are using an older version of IBM Worklight, or if you are experiencing problems with the automatic configuration process for the WebSphere Application Server, you must manually set up and configure your MySQL database and then the WebSphere Application Server for MySQL.

About this task

Complete the MySQL database setup procedure before continuing.

Procedure

1. Set up the JDBC provider:
   a. In the WebSphere Application Server console, click **Resources** > **JDBC** > **JDBC Providers**.
   b. Create a **JDBC provider** named **MySQL**.
   c. Set **Database type** to **User defined**.
   d. Set **Scope** to **Cell**.
   e. Set **Implementation class** to **com.mysql.jdbc.jdbc2.optional.MysqlConnectionPoolDataSource**.
   f. Set **Database classpath** to the location of the `MySQL JDBC` connector `.jar` file.
   g. Save your changes.
2. If you are using `JNDI` configuration, rather than `URI` configuration, create a data source for the IBM Worklight database:
   a. Click **Resources** > **JDBC** > **Data sources**.
   b. Click **New** to create a data source.
   c. Type any name (for example, `Worklight Database`).
   d. Set **JNDI Name** to `jdbc/WorklightDS`.
   e. Use the existing **JDBC Provider MySQL**, defined in the previous step.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

f. Set Scope to **New**.

g. On the **Configuration** tab, select the **Non-transactional data source** check box.

h. Click **Next** a number of times, leaving all other settings as defaults.

i. Save your changes.

3. If you are using JNDI configuration, rather than URI configuration, create a data source for the IBM Worklight reports database:
   a. Click **New** to create a data source.
   b. Type any name (for example, `Worklight Report Database`).
   c. Set **JNDI Name** to `jdbc/WorklightReportsDS` or any other name defined in the `worklight.properties` file to refer to the IBM Worklight report database.
   d. Use the existing **JDBC Provider MySQL**, defined in the previous step.
   e. Set Scope to **New**.
   f. On the **Configuration** tab, select the **Non-transactional data source** check box. **New**.
   g. Click **Next** a number of times, leaving all other settings as defaults.
   h. Save your changes.
4. Create a data source for the IBM Application Center database:
   a. Click **New** to create a data source.
   b. Type any name (for example, `Application Center Database`).
   c. Set **JNDI Name** to `jdbc/AppCenterDS`.
   d. Use the existing **JDBC Provider MySQL**, defined in the previous step.
   e. Set Scope to **New**.
   f. On the **Configuration** tab, select the **Non-transactional data source** check box. **New**.
   g. Click **Next** a number of times, leaving all other settings as defaults.
   h. Save your changes.
5. Set the custom properties of each new data source.
   a. Select the new data source.
   b. Click **Custom properties**.
   c. Set the following properties:

```
portNumber = 3306
relaxAutoCommit=true
databaseName = WRKLGHT or WLREPORT or APPCNTR, respectively
serverName = the host name of the MySQL server
user = the user name of the MySQL server
password = the password associated with the user name
```

Configuring Apache Tomcat for MySQL manually:

IBM Worklight V5.0.5. automatically configures your databases. However, if you are using an older version of IBM Worklight, or if you are experiencing problems with the automatic configuration process for the Apache Tomcat Server, you must manually set up and configure your MySQL database and then the Apache Tomcat Server for MySQL.

About this task

Complete the MySQL database setup procedure before continuing.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issue, refer to the online user documentation.

Procedure

1. Add the MySQL Connector/J JAR file to the $TOMCAT_HOME/lib directory.
2. Update the $TOMCAT_HOME/conf/context.xml file as follows:

```
<Context>
    ...
    <Resource name="jdbc/WorklightDS"
        auth="Container"
        type="javax.sql.DataSource"
        maxActive="100"
        maxIdle="30"
        maxWait="10000"
        username="worklight"
        password="worklight"
        driverClassName="com.mysql.jdbc.Driver"
        url="jdbc:mysql://server:3306/WRKLGHT"/>
    <Resource name="jdbc/WorklightReportsDS"
        auth="Container"
        type="javax.sql.DataSource"
        maxActive="100"
        maxIdle="30"
        maxWait="10000"
        username="worklight"
        password="worklight"
        driverClassName="com.mysql.jdbc.Driver"
        url="jdbc:mysql://server:3306/WLREPORT"/>
    <Resource name="jdbc/AppCenterDS"
        auth="Container"
        type="javax.sql.DataSource"
        maxActive="100"
        maxIdle="30"
        maxWait="10000"
        username="worklight"
        password="worklight"
        driverClassName="com.mysql.jdbc.Driver"
        url="jdbc:mysql://server:3306/APPCNTR"/>
    ...
</Context>
```

3. Add references to the datasources:

   For JDBC, modify worklight.properties as follows:

```
wl.db.type=MYSQL
wl.db.url= jdbc:mysql://localhost:3306/WRKLGHT
wl.reports.db.type=MYSQL
wl.reports.db.url=jdbc:mysql://localhost:3306/WLREPORT
wl.db.username=worklight
wl.db.password=worklight
reports.exportRawData=true
```

   For JNDI, edit the $TOMCAT_HOME/conf/web.xml file as follows:

```
<web-app>
    ....
    ....
    <resource-ref>
        <res-ref-name>jdbc/WorklightDS</res-ref-name>
        <res-type>javax.sql.DataSource</res-type>
        <res-auth>Container</res-auth>
    </resource-ref>
    <resource-ref>
        <res-ref-name>jdbc/WorklightReportsDS</res-ref-name>
        <res-type>javax.sql.DataSource</res-type>
        <res-auth>Container</res-auth>
    </resource-ref>
</web-app>
```

4. In the worklight.properties file, update the following properties:

Experimental IBM Worklight official user documentation
In case of issue, refer to the online user documentation.
This document is provided "as is".

```
wl.db.jndi.name=java:comp/env/jdbc/WorklightDS
wl.db.reports.jndi.name=java:comp/env/jdbc/WorklightReportsDS
```

The `worklight.properties` file is in `worklight.war`.

5. Copy the `worklight.war` file to `$TOMCAT_HOME/webapps`.
6. Copy the `worklight-jee-library.jar` file to `$TOMCAT_HOME/lib`.

Configuring the Oracle databases manually

You configure the Oracle databases manually by creating the databases, creating the database tables, and then configuring the relevant application server to use this database setup.

Procedure

1. Create the databases. This step is described in "Creating the Oracle databases" on page 316
2. Create the tables in the databases. This step is described in "Setting up your Oracle databases manually"
3. Configure the application server to use this database setup. Go to one of the following topics:
   - "Configuring Liberty Profile for Oracle manually" on page 339
   - "Configuring WebSphere Application Server for Oracle manually" on page 340
   - "Configuring Apache Tomcat for Oracle manually" on page 341

Setting up your Oracle databases manually:

IBM Worklight V5.0.5. automatically configures your databases. However, if you are using an older version of IBM Worklight, or if you are experiencing problems with the automatic configuration process, you must manually set up and configure your Oracle databases.

About this task

Complete the following procedure to set up your `Oracle` databases.

Procedure

1. Create the database and users. Using the Oracle Database Configuration Assistant (DBCA), follow the steps in the wizard to create a new general-purpose database named `WRKLGHT`:
   a. Use global database name `WRKLGHT_your_domain`, and system identifier (SID) `WRKLGHT`.
   b. On the **Custom Scripts** tab of the step **Database Content**, do not run the `SQL` scripts, because you must first create a user account.
   c. On the **Character Sets** tab of the step **Initialization Parameters**, select **Use Unicode (AL32UTF8) character set and UTF8 - Unicode 3.0 UTF-8 national character set**.
   d. Complete the wizard, accepting the default values.

   Repeat the previous step to create the IBM Worklight report database.
   a. Use global database name `WLREPORT_your_domain`, and SID `WLREPORT`.

   Repeat the previous step to create the IBM Application Center database.
   a. Use global database name `APPCNTR_your_domain`, and SID `APPCNTR`.
2. Create the user `worklight`, either by using Oracle Database Control, or by using the Oracle SQLPlus command-line interpreter.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues refer to the online user documentation.

Create the user for the `WRKLGHT` database, by using Oracle Database Control:

a. Connect as `SYSDBA`.

b. Go to the Users page.

c. Click **Server**, then **Users** in the Security section.

d. Create a user named `worklight` with the following attributes:

```
Profile: DEFAULT
Authentication: password
Default tablespace: USERS
Temporary tablespace: TEMP
Status: UNLOCK
Add role: CONNECT
Add role: RESOURCE
Add system privilege: CREATE VIEW
Add system privilege: UNLIMITED TABLESPACE
```

Repeat the previous step to create the user `worklight` for the IBM Worklight report database, `WLREPORT` and the IBM Application Center database, `APPCNTR`.

To create the user for all three databases by using Oracle SQLPlus, enter the following commands:

```
CONNECT system/<system_password>@WRKLGHT
CREATE USER worklight IDENTIFIED BY worklight;
GRANT CONNECT TO worklight;
GRANT RESOURCE TO worklight;
GRANT CREATE VIEW TO worklight;
DISCONNECT;
CONNECT system/<system_password>@WLREPORT
CREATE USER worklight IDENTIFIED BY worklight;
GRANT CONNECT TO worklight;
GRANT RESOURCE TO worklight;
GRANT CREATE VIEW TO worklight;
DISCONNECT;
CONNECT system/<system_password>@APPCNTR
CREATE USER worklight IDENTIFIED BY worklight;
GRANT CONNECT TO worklight;
GRANT RESOURCE TO worklight;
GRANT CREATE VIEW TO worklight;
DISCONNECT;
```

3. Create the database tables for the IBM Worklight database and IBM Worklight reports database:

a. Using the Oracle SQLPlus command-line interpreter, create the required tables for the IBM Worklight database (`WRKLGHT`) by executing the `create-worklight-oracle.sql` file:

```
CONNECT system/<system_password>@WRKLGHT
@<worklight_install_dir>/WorklightServer/databases/create-worklight-oracle.sql
DISCONNECT;
```

b. Using the Oracle SQLPlus command-line interpreter, create the required tables for the IBM Worklight report database (`WLREPORT`) by executing the `create-worklightreports-oracle.sql` file:

```
CONNECT system/<system_password>@WLREPORT
@<worklight_install_dir>/WorklightServer/databases/create-worklightreports-oracle.sql
DISCONNECT;
```

c. Using the Oracle SQLPlus command-line interpreter, create the required tables for the IBM Application Center database (APPCNTR) by executing the `create-appcenter-oracle.sql` file:

```
CONNECT system/<system_password>@APPCNTR
@<worklight_install_dir>/ApplicationCenter/databases/create-appcenter-oracle.sql
DISCONNECT;
```

Experimental IBM Worklight online user documentation
This document is provided "as is".
In case of issue, refer to the online user documentation.

4. Create a `worklight.properties` file. Give the file the following contents, depending on whether you are using JDBC or JNDI. For JDBC:

   ```
   wl.db.type=ORACLE
   wl.db.url=jdbc:oracle:thin:@<HOST IP>:1521/WRKLGHT
   wl.reports.db.type=ORACLE
   wl.reports.db.url=jdbc:oracle:thin:@<HOST IP>:1521/WLREPORT
   wl.db.username=worklight
   wl.db.password=worklight
   reports.exportRawData=true
   ```

   For JNDI:

   ```
   wl.db.jndi.name=jdbc/WorklightDS
   wl.reports.db.jndi.name=jdbc/WorklightDS
   wl.db.type=ORACLE
   ```

   If you choose to use the `worklight.properties` file supplied as part of `worklight.war`, remove all properties related to Apache Derby.
5. Replace the `WEB-INF/classes/conf.worklight.properties` file in `worklight.war` with the file you created in the previous step.
6. Download and configure the Oracle JDBC driver:
   a. Download the JDBC driver from the Oracle website at Oracle: JDBC, SQLJ, Oracle JPublisher and Universal Connection Pool (UCP):
   b. Ensure that the Oracle JDBC driver is in the system path. The driver file is `ojdbc6.jar`.

Configuring Liberty Profile for Oracle manually:

IBM Worklight V5.0.5. automatically configures your databases. However, if you are using an older version of IBM Worklight, or if you are experiencing problems with the automatic configuration process for Liberty Profile on Oracle, you must manually set up and configure your Oracle database and then the Liberty Profile for Oracle.

About this task

Complete the Oracle database setup procedure before continuing.

Procedure

1. Add the Oracle JDBC Driver JAR file to `$LIBERTY_HOME/wlp/usr/shared/resources/oracle`. If that directory does not exist, create it.
2. If you are using JNDI, configure the data sources in the `$LIBERTY_HOME/wlp/usr/servers/worklightServer/server.xml` file as shown in the following JNDI code example:

   ```
   <!-- Declare the jar files for Oracle access through JDBC. -->
   <library id="OracleLib">
       <fileset dir="${shared.resource.dir}/oracle" includes="*.jar"/>
   </library>

   <!-- Declare the IBM Worklight Server database. Used through property wl.db.jndi.name.
        If you change this declaration to refer to a different kind of data base,
        you have to update the property wl.db.type in the file worklight.properties
        inside the file worklight.war. -->
   <dataSource id="WorklightDS" jndiName="jdbc/WorklightDS">
       <jdbcDriver libraryRef="OracleLib"/>
       <properties.oracle driverType="thin" databaseName="WRKLGHT"
           serverName="oserver" portNumber="1521"
           user='"worklight"' password="worklight"/>
   </dataSource>

   <!-- Declare the IBM Worklight Server Reports database. Used through property wl.reports.db.jn
   ```

Experimental IBM Worklight offline user documentation
This document is provided as is.
In case of issues refer to the online user documentation.

```
         If you change this declaration to refer to a different kind of data base,
         you have to update the property wl.reports.db.type in the file worklight.properties
         inside the file worklight.war. -->
    <dataSource id="WorklightReportsDS" jndiName="jdbc/WorklightReportsDS">
        <jdbcDriver libraryRef="OracleLib"/>
        <properties.oracle driverType="thin" databaseName="WLREPORT"
           serverName="oserver" portNumber="1521"
           user='"worklight"' password="worklight"/>
    </dataSource>

    <!-- Declare the IBM Application Center database. -->
    <dataSource id="AppCenterDS" jndiName="jdbc/AppCenterDS">
        <jdbcDriver libraryRef="OracleLib"/>
        <properties.oracle driverType="thin" databaseName="APPCNTR"
           serverName="oserver" portNumber="1521"
           user='"worklight"' password="worklight"/>
    </dataSource>
```

where **`worklight`** after **`user=`** is the user name, **`worklight`** after **`password=`** is this user's password, and **`oserver`** is the host name of your Oracle server (for example, localhost, if it is on the same machine).

Configuring WebSphere Application Server for Oracle manually:

IBM Worklight V5.0.5. automatically configures your databases. However, if you are using an older version of IBM Worklight, or if you are experiencing problems with the automatic configuration process for the WebSphere Application Server, you must manually set up and configure your Oracle database and then the WebSphere Application Server for Oracle.

About this task

Complete the Oracle database setup procedure before continuing.

Procedure

1. Set up the `JDBC` provider:
   a. In the WebSphere Application Server console, click **Resources** > **JDBC** > **JDBC Providers** > **New**.
   b. Set the scope of the `JDBC` connection to **Node**.
   c. Complete the `JDBC Provider` fields as indicated in the following table:

Table 98. JDBC Provider field values

Field	**Value**
Database type	**Oracle**
Provider type	**Oracle JDBC Driver**
Implementation type	**Connection pool data source**
Name	**Oracle JDBC Driver**

   d. Click **Next**.
   e. Set the class path for the `ojdbc6.jar` file, for example `/home/Oracle-jar/ojdbc6.jar`.
   f. Click **Next**.
      The JDBC provider is created.
2. Create a data source for the IBM Worklight database:
   a. Click **Resources** > **JDBC** > **Data sources** > **New**.
   b. Set **Data source name** to **Oracle JDBC Driver DataSource**.

Experimental IBM Worklight online user documentation
This document is provided "as is".
In case of issues refer to the online user documentation.

c. Set **JNDI name** to `jdbc/WorklightDS`.
d. Click **Next**.
e. Click **Select an existing JDBC provider** and select **Oracle JDBC driver** from the list.
f. Click **Next**.
g. Set the URL value to **jdbc:oracle:thin:@oserver:1521/WRKLGHT**, where **`oserver`** is the host name of your Oracle server (for example, localhost, if it is on the same machine).
h. Click **Next** twice.
i. Click **Resources** > **JDBC** > **Data sources** > **Oracle JDBC Driver DataSource** > **Custom properties**.
j. Set **oracleLogPackageName** to **oracle.jdbc.driver**.
k. Set **user = worklight**.
l. Set **password = worklight**.

3. Create a data source for the IBM Worklight reports database, following the instructions in step 2, but using the JNDI name `jdbc/WorklightReportsDS` and the URL value **jdbc:oracle:thin:@oserver:1521/WLREPORT**.
4. Create a data source for the IBM Application Center database, following the instructions in step 2, but using the JNDI name `jdbc/AppCenterDS` and the URL value **jdbc:oracle:thin:@oserver:1521/APPCNTR**.

Configuring Apache Tomcat for Oracle manually:

IBM Worklight V5.0.5. automatically configures your databases. However, if you are using an older version of IBM Worklight, or if you are experiencing problems with the automatic configuration process for the Apache Tomcat Server, you must manually set up and configure your Oracle database and then the Apache Tomcat Server for Oracle.

About this task

Complete the Oracle database setup procedure before continuing.

Procedure

1. Add the Oracle JDBC driver JAR file to the directory `$TOMCAT_HOME/lib`.
2. Update the `$TOMCAT_HOME/conf/context.xml` file as follows:

```
<Context>
    ...
    <Resource name="jdbc/WorklightDS"
        auth="Container"
        type="javax.sql.DataSource"
        username="worklight"
        password="worklight"
        driverClassName="oracle.jdbc.driver.OracleDriver"
        url="jdbc:oracle:thin:@oserver:1521/WRKLGHT"/>
    <Resource name="jdbc/WorklightReportsDS"
        auth="Container"
        type="javax.sql.DataSource"
        username="worklight"
        password="worklight"
        driverClassName="oracle.jdbc.driver.OracleDriver"
        url="jdbc:oracle:thin:@oserver:1521/WLREPORT"/>
    <Resource name="jdbc/AppCenterDS"
        auth="Container"
        type="javax.sql.DataSource"
        username="worklight"
```

Experimental IBM Worklight offline user documentation
This document is provided "as-is".
In case of issues, refer to the online user documentation.

```
        password="worklight"
        driverClassName="oracle.jdbc.driver.OracleDriver"
        url="jdbc:oracle:thin:@oserver:1521/APPCNTR"/>
    ...
</Context>
```

Where **oserver** is the host name of your Oracle server (for example, localhost, if it is on the same machine).

3. Add references to the datasources:
   a. For JDBC, modify `worklight.properties` as follows:
   ```
   wl.db.type=ORACLE
   wl.db.url= jdbc:oracle:thin:@ host_ip_address:1521/WRKLGHT
   wl.reports.db.type=ORACLE
   wl.reports.db.url=jdbc:oracle:thin:@ host_ip_address:1521/WLREPORT
   wl.db.username=worklight
   wl.db.password=worklight
   reports.exportRawData=true
   ```
   b. For JNDI, update the `$TOMCAT_HOME/conf/web.xml` file as follows:
   ```
   <web-app>
       ....
       ....
       <resource-ref>
           <res-ref-name>jdbc/WorklightDS</res-ref-name>
           <res-type>javax.sql.DataSource</res-type>
           <res-auth>Container</res-auth>
       </resource-ref>
       <resource-ref>
           <res-ref-name>jdbc/WorklightDS</res-ref-name>
           <res-type>javax.sql.DataSource</res-type>
           <res-auth>Container</res-auth>
       </resource-ref>
   </web-app>
   ```
   Where **oserver** is the host name of your Oracle server (for example, localhost, if it is on the same machine).

Manually configuring the application server

In some cases, you may want to re-configure IBM Worklight Server so that it uses a different application server from the one you specified originally when installing IBM Worklight Server. The procedure depends on the type of application server being configured.

These manual instructions assume that you are familiar with your application server.

Note: Installing by using IBM Installation Manager is more reliable than installing and configuring manually, and should be used whenever possible.

Configuring the WebSphere Liberty Profile manually

To configure the Websphere Liberty Profile Application Server manually, you must modify the `server.xml` file.

About this task

In addition to modifications for the databases that are described in "Manually configuring the databases" on page 321, you must make the following modifications to the `server.xml` file.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

Procedure

1. Ensure that the `<featureManager>` element contains at least the following `<feature>` elements:

   ```
   <feature>ssl-1.0</feature>
   <feature>servlet-3.0</feature>
   <feature>jdbc-4.0</feature>
   <feature>appSecurity-1.0</feature>
   ```

2. Add the following declarations in the `<server>` element, for the Worklight runtime and the Worklight Console:

   ```
   <!-- Declare the IBM Worklight Server application. -->
   <application id="worklight" name="worklight" location="worklight.war" type="war">
       <classloader delegation="parentLast">
           <commonLibrary>
               <fileset dir="${shared.resource.dir}/lib" includes="worklight-jee-library.jar"/>
           </commonLibrary>
       </classloader>
   </application>

   <!-- Declare web container custom properties for the IBM Worklight Server application. -->
   <webContainer invokeFlushAfterService="false"/>
   ```

3. Add the following declarations for the Application Center:

   ```
   <!-- Declare the IBM Application Center Console application. -->
   <application id="appcenterconsole" name="appcenterconsole" location="appcenterconsole.war" typ
       <application-bnd>
           <security-role name="appcenteradmin">
               <group name="appcentergroup"/>
           </security-role>
       </application-bnd>
   </application>

   <!-- Declare the IBM Application Center Services application. -->
   <application id="applicationcenter" name="applicationcenter" location="applicationcenter.war"
       <application-bnd>
           <security-role name="appcenteradmin">
               <group name="appcentergroup"/>
           </security-role>
       </application-bnd>
   </application>

   <!-- Declare the user registry for the IBM Application Center. -->
   <basicRegistry id="applicationcenter-registry" realm="ApplicationCenter">
       <!-- The users defined here are members of group "appcentergroup",
            thus have role "appcenteradmin", and can therefore perform
            administrative tasks through the Application Center Console. -->
       <user name="appcenteradmin" password="admin"/>
       <user name="demo" password="demo"/>
       <group name="appcentergroup">
           <member name="appcenteradmin"/>
           <member name="demo"/>
       </group>
   </basicRegistry>
   ```

What to do next

For more steps to configure the IBM Application Center, see "Configuring WebSphere Application Server Liberty Profile" on page 433.

Configuring WebSphere Application Server manually

You need to configure variables, custom properties, and class loader policies.

Experimental IBM Worklight user documentation.
This document is provided "as is".
In case of issues refer to the online user documentation.

Before you begin

These instructions assume that you already have a stand-alone profile created with an application server named Worklight and that the server is using the default ports.

Procedure

1. Log on to the WebSphere Application Server administration console for your IBM Worklight server. The address is of the form `http://server.com:9060/ibm/console`, where *server* is the name of the server.
2. Create the **`WORKLIGHT_INSTALL_DIR`** variable:
   a. Click **Environment** > **WebSphere Variables**.
   b. From the **Scope** list, select **Worklight server**.
   c. Click **New**. The Configuration pane is displayed.
   d. In the **Name** field, type `WORKLIGHT_INSTALL_DIR`.
   e. In the **Value** field, type `/opt/IBM/Worklight`.
   f. (Optional) In the **Description** field, type a description of the variable.
   g. Click **OK**.
   h. Save the changes.
3. Set the **`worklight.home`** Java Virtual Machine (JVM) custom property:
   a. Click **Servers** > **Server Types** > **Application Servers** > **Worklight**.
   b. In the Server Infrastructure section, select **Java**, or **Java and Process Management**, depending on your version of WebSphere Application Server.
   c. Depending on your version of WebSphere Application Server, click one of the following options:
      - **Process Management** > **Process Definition** > **Java Virtual Machine** > **Custom Properties**
      - **Process Definition** > **Java Virtual Machine** > **Custom Properties**
   d. Click **New**. The Configuration pane is displayed.
   e. In the **Name** field, type `worklight.home`.
   f. In the **Value** field, type `${USER_INSTALL_ROOT}/worklightHome`. This assumes that the environment variable **`USER_INSTALL_ROOT`** defines the profile home directory for this server. If you have more than one application server hosting IBM Worklight in the same profile, you must specify a unique `worklight.home` value for each server.
   g. (Optional) In the **Description** field, type a description of the variable.
   h. Click **OK**.
   i. Save the changes.
4. Create the IBM Worklight shared library definition:
   a. Click **Environment** > **Shared libraries**.
   b. From the **Scope** list, select **Worklight server**.
   c. Click **New**. The Configuration pane is displayed.
   d. In the **Name** field, type `WL_PLATFORM_LIB`.
   e. (Optional) In the **Description** field, type a description of the library.
   f. In the **Classpath** field, type `${WORKLIGHT_INSTALL_DIR}/WorklightServer/worklight-jee-library.jar`.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

5. Create the IBM Worklight JDBC data source and provider. See the instructions for the appropriate DBMS in “Manually configuring the databases” on page 321
6. Add a specific web container custom property.
   a. Click **Servers** > **Server Types** > **Application Servers**, and select the server used for Worklight.
   b. Click **Web Container Settings** > **Web container**.
   c. Click **Custom properties**.
   d. Click **New**.
   e. Enter the property values listed in the following table:

Table 99. Web container custom property values

Property	Value
Name	`com.ibm.ws.webcontainer.invokeFlushAfterService`
Value	`false`
Description	See `http://www-01.ibm.com/support/docview.wss?uid=swg1PM50111`

   f. Click **OK**.
   g. Click **Save**.
7. Install an IBM Worklight customization WAR file:
   a. Depending on your version of WebSphere Application Server, click one of the following options:
      - **Applications** > **New** > **New Enterprise Application**
      - **Applications** > **New Application** > **New Enterprise Application**
   b. Navigate to the IBM Worklight Server installation directory `WL_INSTALL_DIR/WorklightServer`.
   c. Select `worklight.war`, and then click **Next**.
   d. On the "How do you want to install the application?" page, select **Detailed**, and then click **Next**.
   e. On the Application Security Warnings page, click **Continue**.
   f. Click **Continue** repeatedly until you reach Step 4 of the wizard: Map Shared Libraries.
   g. Select the **Select** check box for `worklight_war` and click **Reference shared libraries**.
   h. From the Available list, select `WL_PLATFORM_LIB` and click the **>** button.
   i. Click **OK**.
   j. Click **Next** until you reach the “Map context roots for web modules” page.
   k. In the **Context Root** field, type `/worklight`.
   l. Click **Next**.
   m. Click **Finish**.
8. (Optional). As an alternative to step 6, you can map the shared libraries as follows:
   a. Click **Applications** > **Application Types** > **WebSphere enterprise applications** > **worklight_war**.
   b. In the References section, click **Shared library references**.
   c. Select the **Select** check box for `worklight_war` and click **Reference shared libraries**.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issue, refer to the online user documentation.

   d. From the Available list, select `WL_PLATFORM_LIB` and click the **>** button.
   e. Click **OK** twice to return to the worklight_war configuration page.
   f. Click the **Save** link.
9. Configure the class loader policies and then start the application:
   a. Click the **Manage Applications** link, or click **Applications** > **WebSphere Enterprise Applications**.
   b. From the list of applications, click **worklight_war**.
   c. In the "Detail Properties" section, click the **Class loading and update detection** link.
   d. In the "Class loader order" pane, click **Classes loaded with local class loader first (parent last)**.
   e. Click **OK**.
   f. In the Modules section, click **Manage Modules**.
   g. From the list of modules, click the Worklight module.
   h. In the "Class loader order" pane, click **Classes loaded with local class loader first (parent last)**.
   i. Click **OK** twice.
   j. Click **Save**.
   k. Select the **Select** check box for **worklight_war**and click **Start**.
10. Repeat steps 7 through 9 for `appserverconsole.war` and `applicationcenter.war`.
11. Configure the server to use the single class loader policy:
    a. Click **Servers** > **Server Types** > **Application Servers** > **Worklight**
    b. Change the class loader policy from **Multiple** to **Single**.
    c. Change the class loading mode to **Classes loaded with local class loader first (parent last)**.

Results

You can now access IBM Worklight Console at `http://<server>:<port>/worklight/console`, where *server* is the host name of your server and *port* is the port number (default 9080).

What to do next

For additional steps to configure the IBM Application Center, see "Configuring WebSphere Application Server full profile" on page 431.

Configuring Apache Tomcat manually

You need to copy JAR and WAR files to Tomcat, add database drivers, edit the server.xml file, and then start Tomcat.

Procedure

1. Copy the Worklight Platform `JAR` file to the Tomcat lib directory:
   - On UNIX and Linux systems: `cp WL_INSTALL_DIR/WorklightServer/worklight-jee-library.jar TOMCAT_HOME/lib`
   - On Windows systems: `copy /B WL_INSTALL_DIR\WorklightServer\worklight-jee-library.jar TOMCAT_HOME\lib\worklight-jee-library.jar`
2. Add the database drivers to the Tomcat lib directory. See the instructions for the appropriate DBMS in "Manually configuring the databases" on page 321.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues refer to the online user documentation.

3. Copy the Worklight Customization War file to Tomcat:
   - On UNIX and Linux systems: cp WL_INSTALL_DIR/WorklightServer/ worklight.war TOMCAT_HOME/webapps
   - On Windows systems: copy /B WL_INSTALL_DIR\WorklightServer\ worklight.war TOMCAT_HOME\webapps\worklight.war
4. Edit TOMCAT_HOME/conf/server.xml.
   a. Uncomment the following element, which is initially commented out: <Valve className="org.apache.catalina.authenticator.SingleSignOn" />.
   b. Declare the context and properties of the Worklight application:

```
<!-- Declare the IBM Worklight Console application. -->
<Context path="/worklight" docBase="worklight"/>

<!-- Declare the IBM Application Center application. -->
<Context path="/applicationcenter" docBase="applicationcenter"/>

<!-- Declare the user registry for the IBM Application Center.
      The MemoryRealm recognizes the users defined in conf/tomcat-users.xml.
      For other choices, see Apache Tomcat's "Realm Configuration HOW-TO"
      http://tomcat.apache.org/tomcat-7.0-doc/realm-howto.html . -->
<Realm className="org.apache.catalina.realm.MemoryRealm"/>
```

5. Copy the IBM Application Center WAR file to Tomcat:
   - On UNIX and Linux systems: cp WL_INSTALL_DIR/ApplicationCenter/ console/*.war TOMCAT_HOME/webapps
   - On Windows systems:

```
copy /B WL_INSTALL_DIR\ApplicationCenter\console\appcenterconsole.war TOMCAT_HOME\webapps\ap
copy /B WL_INSTALL_DIR\ApplicationCenter\console\applicationcenter.war TOMCAT_HOME\webapps\a
```

6. Start Tomcat.

What to do next

For additional steps to configure the IBM Application Center, see "Configuring Apache Tomcat" on page 435.

Starting IBM Worklight Server with Liberty Profile

You can start IBM Worklight Server with Liberty Profile by using the appropriate commands for your specific operating system.

About this task

Use the appropriate commands for UNIX, Windows 7, and Windows XP systems to start IBM Worklight Server with Liberty Profile:

Procedure

1. Ensure that you have a Java 6 implementation in your *PATH*.
2. Issue the commands on the command line:

   For UNIX systems:

   cd /opt/IBM/Worklight (or your installation location, if different).

   cd server/wlp/bin

   ./server start worklightServer

   For Windows systems:

   cd C:\Program Files (x86)\IBM\Worklight (or your installation location, if different).

Experimental IBM Worklight online documentation. This document is provided as user documentation. In case of issues, refer to the online user documentation.

```
cd server\wlp\bin

server.bat start worklightServer
```

On Windows 7 systems, if you installed IBM Worklight as a multi-user installation for the group "Administrators", you must run the server with administrator privileges. To this effect, launch the command window or the desktop shortcut **Start server** with the **Run as Administrator** action.

Results

Worklight Server starts.

If you experience problems, inspect the server log files in the following directories:

On UNIX systems:
`<INSTALL_DIR>/server/wlp/usr/servers/worklightServer`

On Windows 7 systems:
`C:\ProgramData\IBM\Worklight\WAS85liberty-server\wlp\usr\servers\worklightServer`

On Windows XP systems:
`C:\Documents and Settings\All Users\Application Data\IBM\Worklight\WAS85liberty-server\wlp\usr\servers\worklightServer`

Starting IBM Worklight Server with WebSphere Application Server

You can start IBM Worklight Server with WebSphere Application Server by using the appropriate commands for your specific operating system.

See the WebSphere Application Server user documentation.

Note: Ensure that your **PATH** system variable contains the directory `was\bin`.

Starting IBM Worklight Server with Apache Tomcat

You can start IBM Worklight Server with Apache Tomcat by using the appropriate commands for your specific operating system.

About this task

Use the appropriate commands for Unix and Linux, or Windows systems to start IBM Worklight Server with Apache Tomcat:

Procedure

Start Tomcat:

On UNIX and Linux systems:
`TOMCAT_HOME/bin/startup.sh`

On Windows systems:
`TOMCAT_HOME\bin\startup.bat`

Verifying IBM Worklight Server startup

You can check that IBM Worklight Server is correctly installed.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

About this task

You can easily verify correct installation of IBM Worklight Server, by opening a web browser and pointing it to your application server.

Procedure

1. Open a web browser.
2. Point it to your application server `http://<server>:<port>/<publicWorkLightContext>/console`. The port number might vary by application server. The URL is referred to as the *IBM Worklight Console URL*. The installation sets *PublicWorkLightContext* to `worklight`. This context changes, as you work with different IBM Worklight projects.

   An IBM Worklight console GUI with no errors indicates that the installation is valid.

Applying environment-specific customization

You can apply environment-specific customization by creating a different environment-specific customization `war` file for each environment, for example testing, production, and so on.

About this task

The IBM Worklight installation creates a temporary customization `war` file that causes an initial IBM Worklight server to be started (thus validating an initial install).

You must now create a project-specific customization `WAR` file, which will be one of several. A project-specific customization `WAR` file is required for each environment.

Project-specific
: A customization `WAR` file contains properties, libraries, web applications, and security settings that are specific to your mobile project.

Environment-specific
: A different `WAR` file is required for each environment, testing, production, and so on. For example, the public access URL, access passwords to databases, and URL to back-end systems change between environments.

A `WAR` file is created from the IBM Worklight Project source code generated by a developer from your organization. Specific customization `WAR` files can be created in two ways:

- By working with the developer to re-build a variant of the mobile project using the IBM Worklight Development Studio.
- By automating the process using an Ant task to generate the `WAR` file from the project source code (usually stored in a source control system). The Ant task is documented in "Ant tasks for building and deploying" on page 134.

Note: It is important to match the context root to which the `WAR` file is deployed to the IBM Worklight property **`publicWorkLightContext`** in the `worklight.properties` file. On most application servers, the default context root name is the `WAR` file name.

Note: You cannot deploy more than one IBM Worklight `WAR` file per server. The deployment might succeed, but might result in some unexpected runtime behavior.

Experimental IBM Worklight offline user documentation
This document is provided "as-is".
In case of issues, refer to the online user documentation.

Perform the following steps:

Procedure

1. Create a WAR file using one of the above methods. Prior to generating the file, correct the `worklight.properties` file (part of the source) to include accurate database connection properties. These properties can be:
   - Created using the authorized `worklight.properties` values given in "IBM Worklight properties" on page 406.
   - Retrieved from the automatically generated WAR file created using the IBM Worklight installation by viewing the file via a zip utility.
2. If the WAR file already exists, undeploy the automatically-created WAR file created by the installation from the application server.
3. Deploy the new customization WAR to the IBM Worklight Server. Each application server type (Tomcat, WebSphere, and Liberty) has its own way of deploying a WAR file.
4. Repeat the verification process above using the new `publicWorkLightContext` in the console URL.

Deploying content: applications and adapters

You can deploy customer-specific content (apps and adapters) only after the customization is set and the server is started.

About this task

Customer-specific content includes applications that must be served by the IBM Worklight Server and their underlying integration adapters. You can create apps and adapters by building them in IBM Worklight Studio, or by using the Ant tasks provided with the IBM Worklight platform to build them. The result of the build action is files with extension `.wlapp` and `.adapter` respectively.

There are two ways to deploy applications and adapters to IBM Worklight Console:

- Use Ant tasks provided with the IBM Worklight platform, and described in "Ant tasks for building and deploying" on page 134.
- Use the IBM Worklight Console to manually deploy apps and adapters as described below.

The IBM Worklight Console opens in a "Catalog" page that enables you to work with apps and adapters.

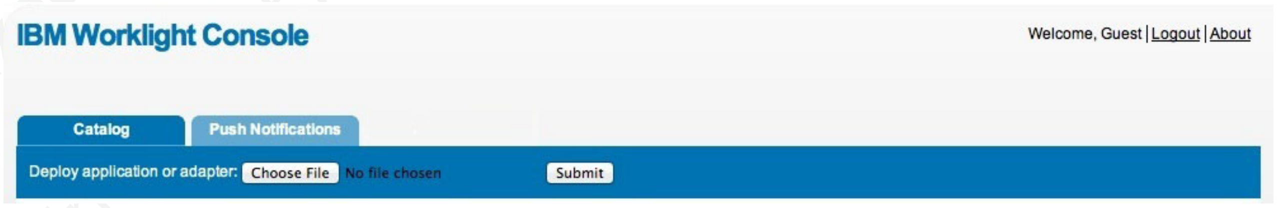


Figure 37. IBM Worklight Catalog

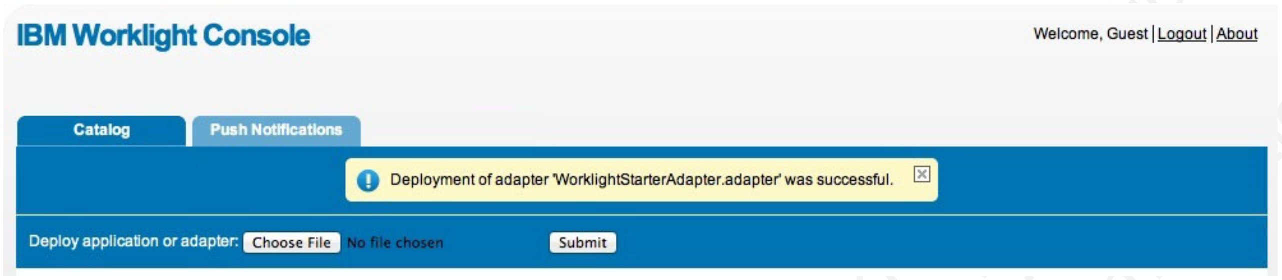
To deploy an adapter:

Experimental IBM Worklight offline user documentation. This document is provided "as is", refer to the online user documentation in case of issues.

Procedure

1. Click **Browse**, then navigate to the `.adapter` file and select it.
2. Click Submit.

   A message is displayed indicating whether the deployment action succeeded or failed.



IBM Worklight Console

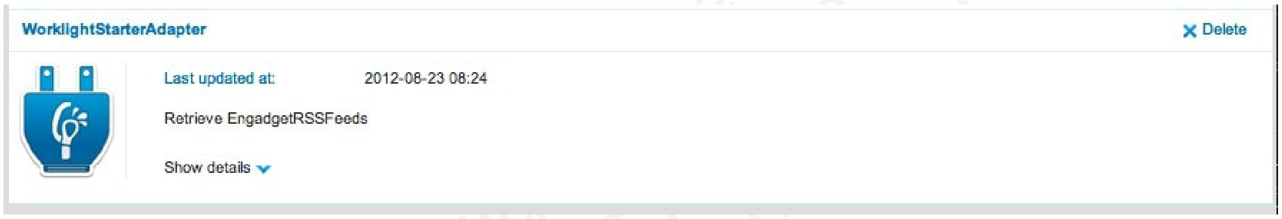
Welcome, Guest | Logout | About

Deployment of adapter 'WorklightStarterAdapter.adapter' was successful.

Deploy application or adapter: Choose File No file chosen Submit

Figure 38. IBM Worklight adapter deployment success or failure message

As a result, the details of the deployed adapter are added to the catalog:



WorklightStarterAdapter ✕ Delete

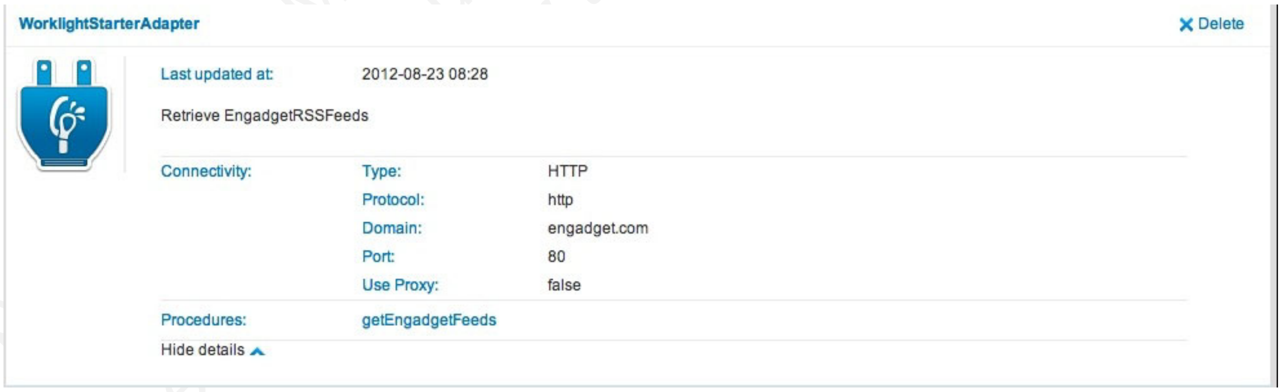
Last updated at: 2012-08-23 08:24

Retrieve EngadgetRSSFeeds

Show details

Figure 39. Deployed adapter details

3. Click **Show details** to view connectivity details for the adapter and the list of procedures it exposes.



WorklightStarterAdapter ✕ Delete

Last updated at: 2012-08-23 08:28

Retrieve EngadgetRSSFeeds

Connectivity:	Type:	HTTP
	Protocol:	http
	Domain:	engadget.com
	Port:	80
	Use Proxy:	false
Procedures:	getEngadgetFeeds	

Hide details

Figure 40. Adapter connectivity details

4. Repeat steps 1 – 3 for each adapter.

   To deploy an application:
5. In the catalog page, click **Browse**, then navigate to the `.wlapp` file and select it.

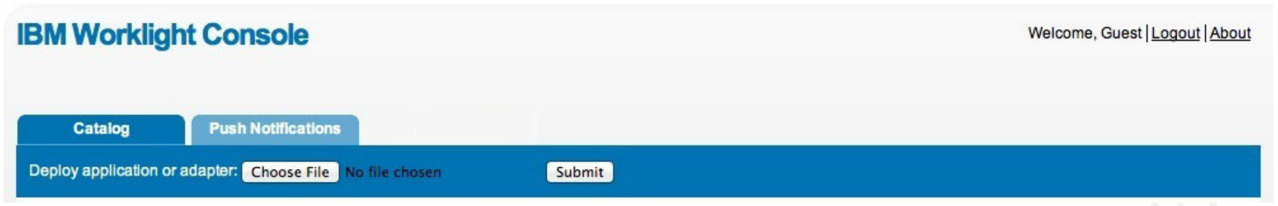
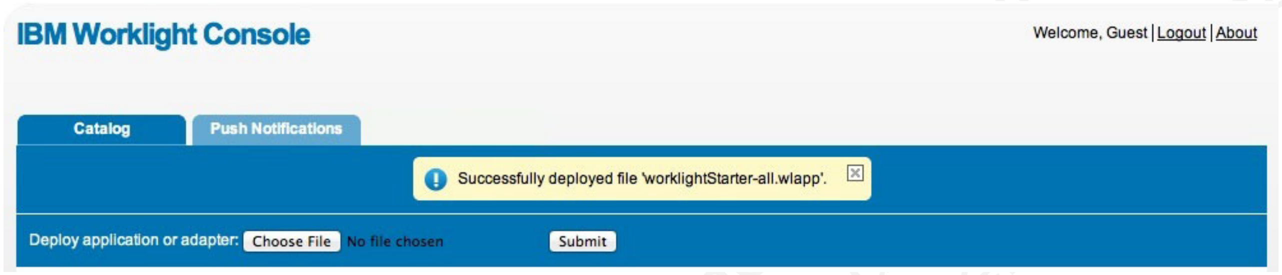


Figure 41. Browsing the catalog page to find the `.wlapp` file

6. Click **Submit**. A message is displayed indicating whether the deployment action succeeded or failed.

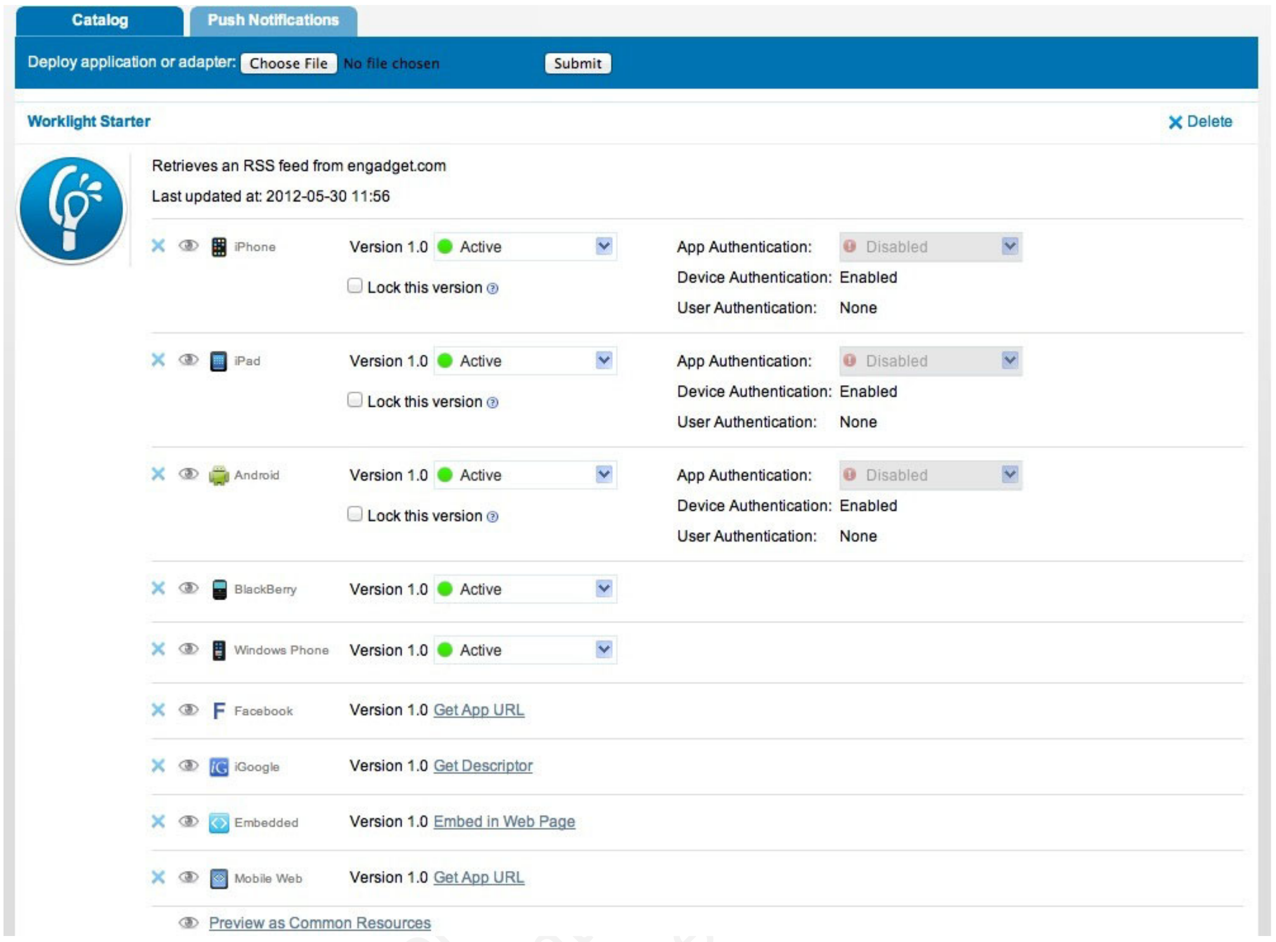


Successfully deployed file 'worklightStarter-all.wlapp'.

Figure 42. Deployment success or failure message

As a result, the details of the deployed application are added to the catalog.

Experimental IBM Worklight offline documentation. This document is provided for offline use. In case of issues, refer to the online documentation.


Catalog | Push Notifications

Deploy application or adapter: Choose File No file chosen Submit

Worklight Starter ✕ Delete

Retrieves an RSS feed from engadget.com

Last updated at: 2012-05-30 11:56

iPhone	Version 1.0 ● Active ☐ Lock this version	App Authentication:	Disabled
		Device Authentication:	Enabled
		User Authentication:	None
iPad	Version 1.0 ● Active ☐ Lock this version	App Authentication:	Disabled
		Device Authentication:	Enabled
		User Authentication:	None
Android	Version 1.0 ● Active ☐ Lock this version	App Authentication:	Disabled
		Device Authentication:	Enabled
		User Authentication:	None
BlackBerry	Version 1.0 ● Active		
Windows Phone	Version 1.0 ● Active		
Facebook	Version 1.0 Get App URL		
iGoogle	Version 1.0 Get Descriptor		
Embedded	Version 1.0 Embed in Web Page		
Mobile Web	Version 1.0 Get App URL		

Preview as Common Resources

Figure 43. Details of the deployed application

7. Repeat steps 1 and 2 for each app.

Database and certificate security passwords

When you configure an IBM Worklight server, you must typically configure database and certificate passwords for security.

Configuration of an IBM Worklight server typically includes the following credentials:

- User name and password to the IBM Worklight database
- User name and password to other custom databases
- User name and password to certificates that enable the stamping of apps

All credentials are stored in the `worklight.properties` file. See "IBM Worklight properties" on page 406 for information about individual properties stored in this configuration file.

You can encrypt any or all of these passwords. For more information, see "Storing properties in encrypted format" on page 411.

Apache Tomcat security options

An optimal Apache Tomcat security balances ease of use and access with strengthening of security and hardening of access.

You must harden the Tomcat Server according to your company policy. Information on how to harden Apache Tomcat is available on the Internet. IBM Worklight usage of Apache Tomcat is limited to a single web app and a single shared library. All other out-of-the-box services provided by Apache Tomcat are unnecessary and can be removed.

WebSphere Application Server security options

The WebSphere Application Server provides Java Platform, Enterprise Edition container and server security by supporting a variety of user registries and a common mechanism to secure EAR/WAR/services.

IBM Worklight provides an extensible authentication model as part of its core function. Follow the instructions to use WebSphere Application Server security to protect the application and adapters hosted on the IBM Worklight runtime environment.

There are three major phases to show how a device uses a typical IBM Worklight app running on WebSphere Application Server shown in the following diagram.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

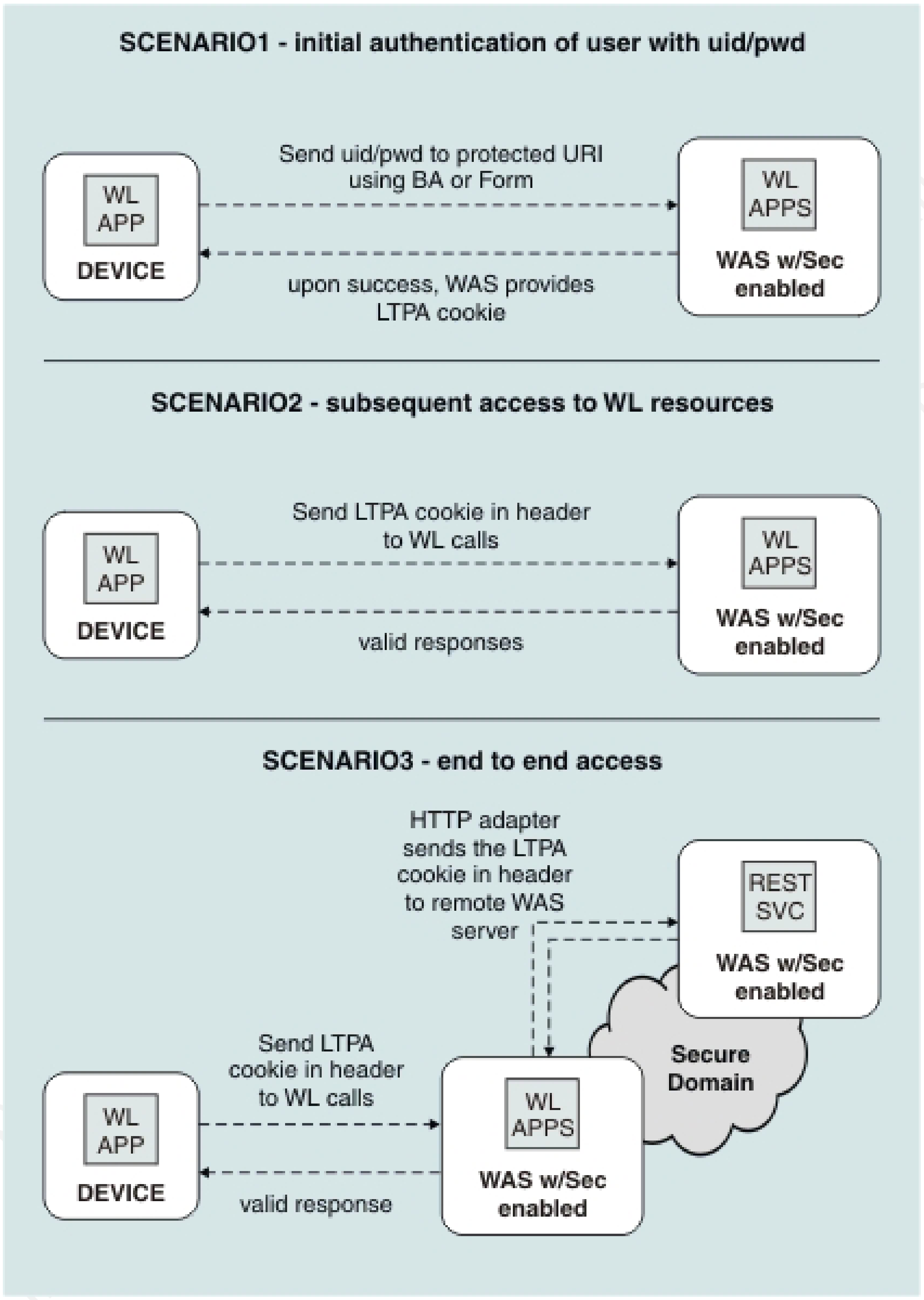

SCENARIO1 - initial authentication of user with uid/pwd
WL APP
DEVICE
Send uid/pwd to protected URI using BA or Form
upon success, WAS provides LTPA cookie
WL APPS
WAS w/Sec enabled
SCENARIO2 - subsequent access to WL resources
WL APP
DEVICE
Send LTPA cookie in header to WL calls
valid responses
WL APPS
WAS w/Sec enabled
SCENARIO3 - end to end access
HTTP adapter sends the LTPA cookie in header to remote WAS server
REST SVC
WAS w/Sec enabled
Secure Domain
WL APP
DEVICE
Send LTPA cookie in header to WL calls
valid response
WL APPS
WAS w/Sec enabled

Figure 44. Phases in securing an IBM Worklight app

When the user first accesses the app, the app issues an authentication challenge, requiring the user to enter their credentials. If authentication is successful, an LTPA token is obtained. This token is used in subsequent calls.

The LTPA token can also be transmitted to back-end WebSphere applications or services that are in the same security domain as IBM Worklight Server.

You can secure IBM Worklight in a typical WebSphere Application Server runtime environment in either of two ways:

- Option 1: Securing WebSphere Application Server using application security and securing the IBM Worklight WAR file.
- Option 2: Securing WebSphere Application Server using application security but not securing the IBM Worklight WAR file.

Each option has advantages and disadvantages. Both options use underlying WebSphere Application Security configuration, but in different ways. Choose an option based on your specific requirements.

Table 100. WebSphere Application Security Options

	Option 1	**Option 2**
BENEFITS	Uses the traditional WebSphere Application Server authentication and trust model. The container enforces all security, so it can use existing third-party SSO products to secure the Java Platform, Enterprise Edition container.	Uses the traditional WebSphere Application Server authentication and trust model without the impact of modifying the IBM Worklight Project WAR. The container enforces all security, so it can use existing third-party SSO products to secure the Java Platform, Enterprise Edition container. The layered authentication of device, application, application instance, and user functions as intended. Flexibility in configuring specific security settings that are specific to the IBM Worklight runtime environment without being hindered by the underlying container security.
USAGE	Suitable for scenarios where the devices can be trusted and access for rogue applications is restricted.	Suitable for scenarios where the devices or the apps on the devices cannot be trusted. The multi-step authenticity checking built into IBM Worklight platform ensures denial of service to devices subjected to unauthorized modifications, rogue applications, and unauthorized users.

WebSphere Application Server security option 1 procedure

To secure WebSphere Application Server, you can choose between two different configurations. The security option 1 procedure secures the IBM Worklight WAR file.

Experimental IBM Worklight online user documentation.
This document is provided "as is".
In case of issues, refer to the online user documentation.

About this task

Complete the following steps to perform the WebSphere Application Server security option 1 procedure and secure the IBM Worklight WAR file.

Procedure

1. Ensure that IBM Worklight is correctly installed on a WebSphere Application Server instance. The IBM Worklight instance contains all the necessary libraries to support WebSphere Application Server security.
2. When installation of the IBM Worklight Server application on WebSphere Application Server is complete, open your WebSphere Application Server integrated solutions console.
3. Ensure that application security is enabled and configured to your enterprise user.

   The IBM Worklight project uses the existing login page and login error page and preconfigured realms as part of the IBM Worklight Server installation on WebSphere Application Server. The IBM Worklight Server application is secured by default using a generic role and using a login form and error page. The following code snippet shows the `web.xml` file of the IBM Worklight Server WAR that is generated for WebSphere Application Server.

```
<security-constraint id="SecurityConstraint_1">
    <web-resource-collection id="WebResourceCollection_1">
    <web-resource-name>Protected URI</web-resource-name>
      <description>Protection area for stuff we want to protect, of course.</description>
      <url-pattern>/*</url-pattern>
      <http-method>GET</http-method>
      <http-method>POST</http-method>
  </web-resource-collection>
  <auth-constraint id="AuthConstraint_1">
    <description>All Authenticated users for our protected stuff.</description>
    <role-name>Role 3</role-name>
  </auth-constraint>
  <user-data-constraint id="UserDataConstraint_1">
    <transport-guarantee>NONE</transport-guarantee>
  </user-data-constraint>
</security-constraint>
<security-role id="SecurityRole_1">
  <description>All Authenticated Users Role.</description>
  <role-name>Role 3</role-name>
</security-role>
```

WebSphere Application Server security option 2 procedure

To secure WebSphere Application Server, you can choose between two different configurations. The security option 2 procedure disables the security at the IBM Worklight Server WAR file level and authenticates users within the IBM Worklight Server runtime environment.

About this task

Complete the following steps to perform the WebSphere Application Server security option 2 procedure, which disables the security at the IBM Worklight Server WAR file level and authenticates users within the IBM Worklight Server runtime environment.

Procedure

1. Complete the same steps as for the Security Option 1, but do not secure the WAR file.

Experimental IBM Worklight user documentation. This is a preview of the online user documentation. In case of issues, use the online user documentation.

To secure WebSphere Application Server and secure the IBM Worklight Server application:

2. Do not enable security-constraint on the `web.xml` file.
3. Configure `applicationDescriptor.xml`.
4. Complete the remaining steps.

Running IBM Worklight in WebSphere Application Server with Java 2 security enabled

You can run IBM Worklight in WebSphere Application Server with Java 2 security enabled.

About this task

To run IBM Worklight in WebSphere Application Server with Java 2 security enabled, complete the following procedure to modify the `app.policy` file and then restart WebSphere Application Server for the modification to take effect.

Procedure

1. Install IBM Worklight Server on a WebSphere Application Server instance. The IBM Worklight instance contains all the necessary libraries to support WebSphere Application Server security.
2. Enable Java 2 security in WebSphere Application Server.
   a. In the WebSphere Application Server console, click **Security** > **Global security**
   b. Select the **Use Java 2 security to restrict application access to local resources** check box.
3. Modify the `app.policy` file, `<ws.install.root>/profiles/<server_name>/config/cells/<cell_name>/node/<node_name>/app.policy`.

   The `app.policy` file is a default policy file that is shared by all of the WebSphere Application Server enterprise applications. For more information, see "*`app.policy` file permissions*" in the WebSphere Application Server documentation.

   In order to run IBM Worklight in WebSphere Application Server with Java 2 security enabled, add the following content into the `app.policy` file.

```
grant codeBase "file:${was.install.root}/worklight-jee-library-xxx.jar"{
  permission java.security.AllPermission;
};

// The war file is your WL server war.
grant codeBase "file:worklight.war"{
  //permission java.security.AllPermission;
  //You can use all permission for simplicity, however, it might
  // cause security problems.
  permission java.lang.RuntimePermission "*";
  permission java.io.FilePermission "${was.install.root}${/}-", "read,write,delete";
  permission java.io.FilePermission "C:/Windows/TEMP/${/}-", "read,write,delete";// In Linux need
  permission java.util.PropertyPermission "*", "read, write";
  permission java.lang.reflect.ReflectPermission "suppressAccessChecks";
  permission com.ibm.tools.attach.AttachPermission "createAttachProvider";
  permission com.ibm.tools.attach.AttachPermission "attachVirtualMachine";
  permission com.sun.tools.attach.AttachPermission "createAttachProvider";
  permission com.sun.tools.attach.AttachPermission "attachVirtualMachine";
  permission java.net.SocketPermission "*", "accept,resolve";
};
```

Experimental IBM Worklight offline user documentation
This document is provided "as is"
In case of issues refer to the online user documentation.

4. Restart WebSphere Application Server for the modification of the `app.policy` file to take effect.

Changing the IBM Worklight Server working directory

You can change the IBM Worklight server working directory according to your application server.

IBM Worklight requires a working directory that stores logs, temporary files, and so on. The default directory varies according to your application server:

- For WebSphere Application Server: *WAS_INSTALL_DIR*`/profiles/profile/servers/server/worklight.home`
- For WebSphere Application Server Liberty Profile:`wlp/usr/servers/server/worklight.home`
- For Tomcat: *TOMCAT_INSTALL_DIR*`/bin/worklight.home`

To change this directory, you must set either the JVM system property **`worklight.home`** or the environment variable **`WORKLIGHT_HOME`** to a new directory. The directory can be any folder to which the IBM Worklight Server has write permissions.

Worklight properties file

The IBM `worklight.properties` file contains configuration properties for the IBM Worklight Server. The file is packed into the customization `WAR`. Review the file on the server location and ensure it is tuned for production, see "IBM Worklight properties" on page 406 for further details.

Administering IBM Worklight applications

You can administer IBM Worklight applications through the Worklight console, by implementing direct updates to mobile devices and desktop apps, by locking apps or denying access, or by displaying notification messages.

Use the Worklight Console to manage your applications. You can use the console to see all applications that are installed and all the device platforms that are supported. You can use the console to disable specific application versions on specific platforms and to force users to upgrade the application before they continue to use them. Additionally, you can use the console to send out notifications to application users, and to manage push notifications from defined event sources to applications. You can also use the Worklight Console to install and manage adapters that are used by applications, and to inspect aggregated usage statistics from the Worklight Server.

When you implement direct updates to mobile devices and desktop apps, software updates are pushed directly to application web resources or users' desktops.

You can lock apps to prevent them from being mistakenly updated and to prevent the redeployment of web resources for a particular application.

You can display a notification message on app startup to give information to users, but which does not cause the application to exit.

You can also control authenticity testing for an application.

Experimental IBM Worklight offline user documentation
This document is provided "as-is".
In case of issues, refer to the online user documentation.

Direct updates of app versions to mobile devices

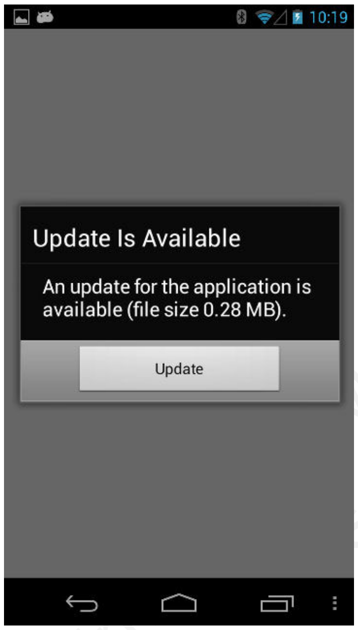
The IBM Worklight Server can directly push updated web resources to deployed applications.

Subject to the terms and conditions of the target platform, organizations are not required to upload new app versions to the app store or market. This option is available for iPhone, iPad, and Android apps.

When you redeploy an app to the IBM Worklight Server without changing its version, the IBM Worklight Server directly pushes the web resources (HTML, JavaScript, and CSS) of the newly deployed app to the device. When an app with an older version of these resources connects to the server. It does not push updated native code.

Direct Update is enabled by default. To update the web resources of an app on a certain environment, redeploy the app.

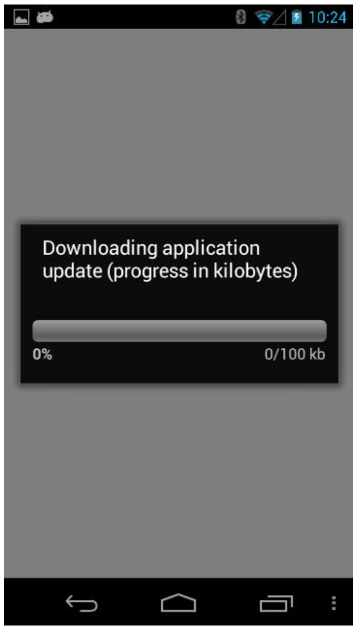
When the app connects to the IBM Worklight Server, it starts downloading the newly deployed resources, as shown in the following figures.


Update Is Available

An update for the application is available (file size 0.28 MB).

Figure 45. Update notice from Android

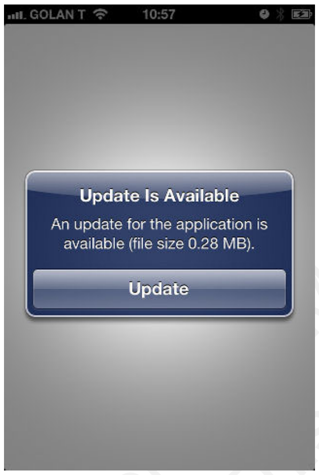
Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.


Downloading application update (progress in kilobytes)

0%

0/100 kb

Figure 46. Downloading newly deployed resources to Android



Update Is Available

An update for the application is available (file size 0.28 MB).

Update

Figure 47. Update notice from iOS

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

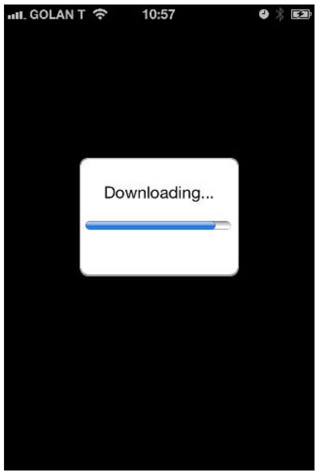


Figure 48. Downloading newly deployed resources to iOS

Direct updates of app versions to desktop apps

A direct updates mechanism is available for desktop apps as well as for mobile devices.

The use of Windows 7 and Vista gadgets and OS X Dashboard widgets is deprecated in Worklight V5.0.5. Support might be removed in any future version.

When you redeploy a desktop app with a new version, the IBM Worklight Server automatically pushes the app to the user's desktop. When the desktop app connects to the IBM Worklight Server and an update is available, it displays a dialog box for the user, asking the user to accept a new version. If the user accepts the new version, it is automatically downloaded to the user's desktop. The user must then open the downloaded app to install it on the desktop.

This option is available for Adobe AIR applications, Windows 7 and Vista gadgets, and Mac OS X Dashboard widgets.

Locking an application

You can prevent developers or administrators from mistakenly updating an application, by locking it in IBM Worklight Console.

About this task

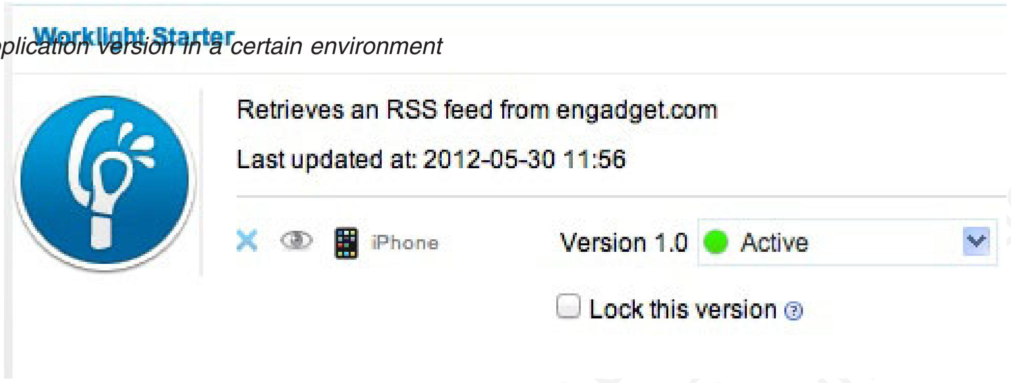
You can lock applications for iPhone, iPad, and Android.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Procedure

To lock an application version in a certain environment, check the **Lock this version** check box for that application version in the required environment.

Figure 49. Locking an application version in a certain environment



Retrieves an RSS feed from engadget.com

Last updated at: 2012-05-30 11:56

Remotely disabling application connectivity

You can use the Remote Disable procedure to deny a user's access to a certain application version due to phase-out policy or due to security issues encountered in the application.

About this task

Using the IBM Worklight Console, you can disable access to a specific version of a specific application on a specific mobile operating system and provide a custom message to the user.

Experimental IBM Worklight offline user documentation. This document is provided as is. In case of issues, refer to the online user documentation.

Procedure

To use this Remote Disable feature, change the status of the application version that must be disabled from **Active** to **Access Disabled**, and add a custom message:

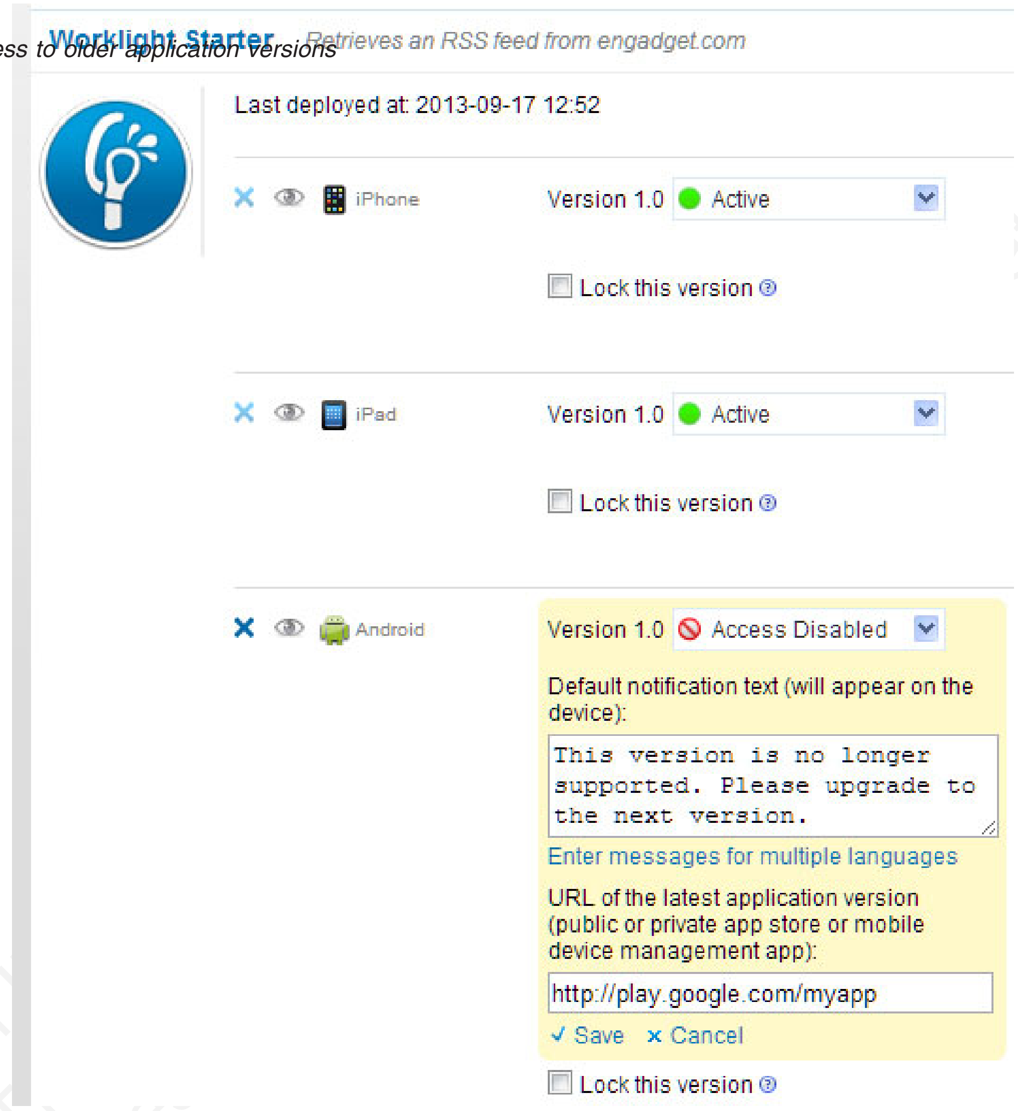


Figure 50. Denying access to older application versions

Worklight Starter Retrieves an RSS feed from engadget.com

Last deployed at: 2013-09-17 12:52

iPhone Version 1.0 Active

Lock this version

iPad Version 1.0 Active

Lock this version

Android Version 1.0 Access Disabled

Default notification text (will appear on the device):

```
This version is no longer
supported. Please upgrade to
the next version.
```

Enter messages for multiple languages

URL of the latest application version (public or private app store or mobile device management app):

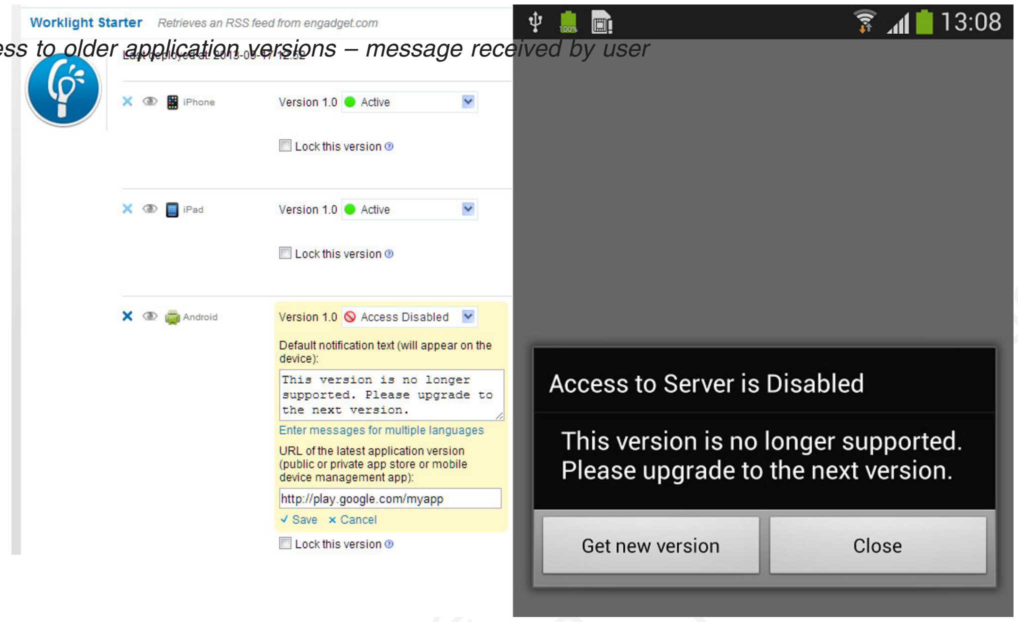
http://play.google.com/myapp

✓ Save × Cancel

Lock this version

You can also specify a URL for the new version of the application (usually in the appropriate app store or market).

When users run an application after it has been Remotely Disabled, they receive a text message about the access denial and can either close the dialog and continue working offline (that is, without access to the Worklight Server), or they can upgrade to the latest version of the application. Closing the dialog keeps the application running, but any application interaction that requires server connectivity causes the dialog to be displayed again.


Worklight Starter Retrieves an RSS feed from engadget.com

Last deployed at: 2013-05-17 12:52

iPhone Version 1.0 ● Active

Lock this version

iPad Version 1.0 ● Active

Lock this version

Android Version 1.0 Access Disabled

Default notification text (will appear on the device):

This version is no longer supported. Please upgrade to the next version.

Enter messages for multiple languages

URL of the latest application version (public or private app store or mobile device management app):

http://play.google.com/myapp

✓ Save × Cancel

Lock this version

13:08

Access to Server is Disabled

This version is no longer supported. Please upgrade to the next version.

Get new version Close

Figure 51. Denying access to older application versions – message received by user

Modifying the behavior of the Remote Disable operation

As noted above, the *default* dialog that is displayed to a user when an application is remotely disabled contains two buttons, **Get new version**, and **Close**. Clicking **Close** closes the dialog, but allows the user to continue working offline, with no connection to the Worklight Server.

Note: The actual text on the two buttons is customizable, and can be overridden in the `message.properties` file.

In older versions of IBM Worklight, when you disabled an application using the Worklight Console, the default behavior was to completely disable it, such that the application would not function, even in offline mode.

There is a way to modify the default behavior of the Remote Disable feature to completely disable an application if there is a need to do so (such as a severe security flaw).

- Add a new Boolean attribute to your `initOptions.js` file, named **showCloseOnRemoteDisableDenial**.
- If this attribute is missing or is set to `true`, the Remote Disable notification displays the default behavior described earlier.
- If this attribute is set to `false` (that is, "Do not show the **Close** button on the dialog"), the behavior is as follows:
  - If you disable the application on the Worklight Console and specify a link to the new version, the dialog displays only a single button, the **Get new version** button. The **Close** button is not shown. The user has no choice but to update the application, and this preserves the older behavior of forcing the user to exit the application.
  - If you disable the application and do not specify a link to the new version, the dialog again displays only a single button, but in this case the **Close** button.

Related tasks:

"Defining administrator messages from the IBM Worklight Console in multiple languages" on page 366

You can set the deny and notification messages from the IBM Worklight Console in multiple languages. The messages are sent based on the locale of the device, and must comply with the ISO 639-1 and ISO 3166-2 standards.

Displaying a notification message on application startup

You can set a notification message that is displayed for the user when the application starts, but does not cause the application to exit.

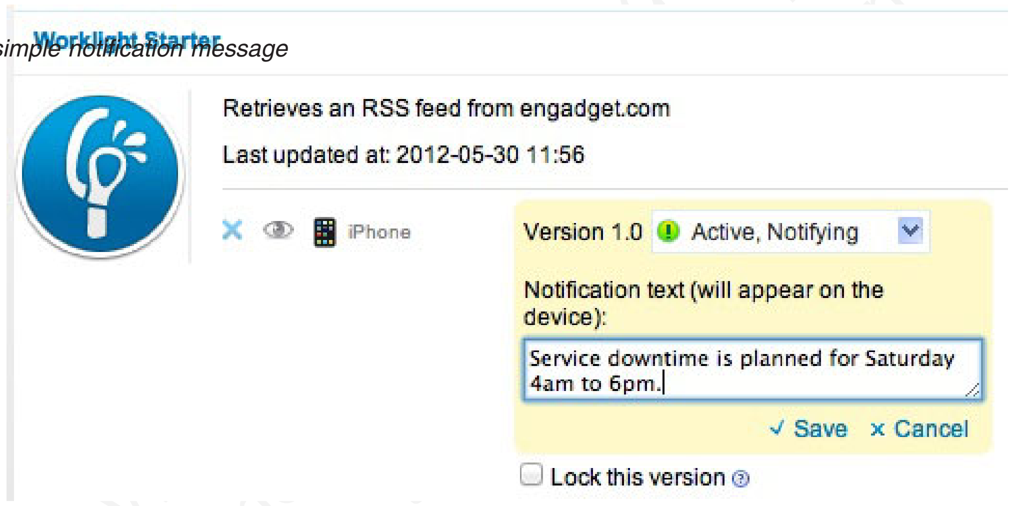
About this task

You can use this type of message to notify application users of temporary situations, such as planned service downtime.

Procedure

For the relevant application, change the status of the application version from **Active** to **Active, Notifying**, and add a custom message:

Figure 52. Displaying a simple notification message



Retrieves an RSS feed from engadget.com

Last updated at: 2012-05-30 11:56

Notification text (will appear on the device):

Service downtime is planned for Saturday 4am to 6pm.

Results

The message is displayed the next time that the app is started or resumed. The message is displayed only once.

Related tasks:

"Defining administrator messages from the IBM Worklight Console in multiple languages"

You can set the deny and notification messages from the IBM Worklight Console in multiple languages. The messages are sent based on the locale of the device, and must comply with the ISO 639-1 and ISO 3166-2 standards.

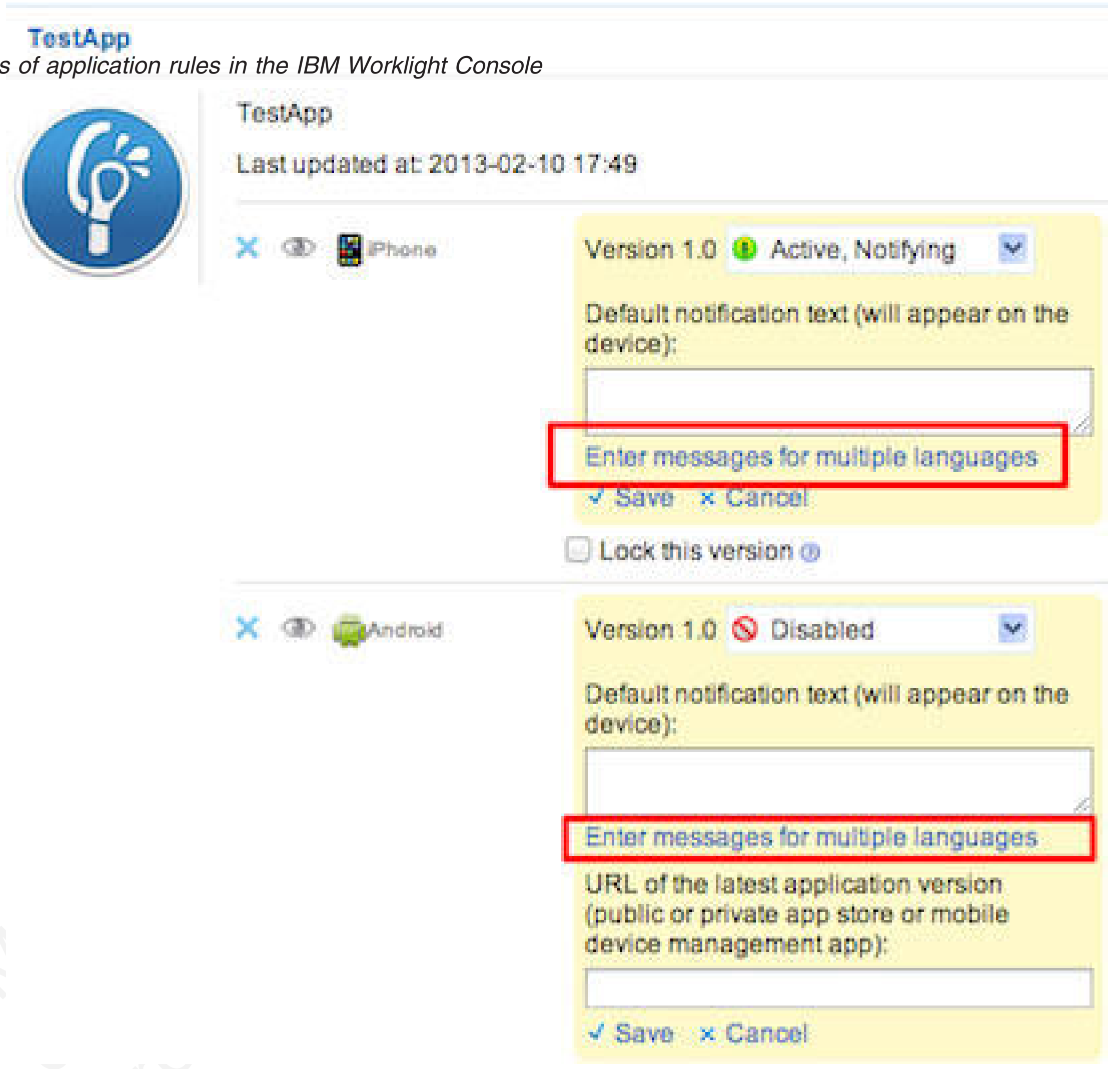
Defining administrator messages from the IBM Worklight Console in multiple languages

You can set the deny and notification messages from the IBM Worklight Console in multiple languages. The messages are sent based on the locale of the device, and must comply with the ISO 639-1 and ISO 3166-2 standards.

Procedure

To add the deny and notification messages for multiple languages, follow these steps.

1. In the IBM Worklight Console, select the status **Active, Notifying**, or **Disabled** in the list of application rules.
2. Click **Enter messages for multiple languages**.



TestApp

TestApp

Last updated at: 2013-02-10 17:49

iPhone

Version 1.0 Active, Notifying

Default notification text (will appear on the device):

Enter messages for multiple languages

✓ Save × Cancel

Lock this version

Android

Version 1.0 Disabled

Default notification text (will appear on the device):

Enter messages for multiple languages

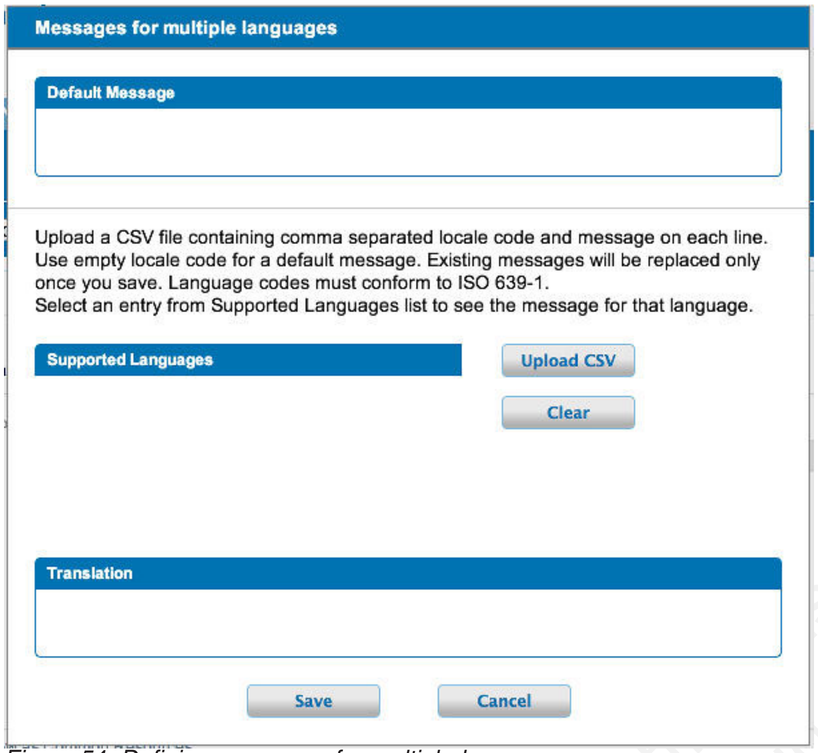
URL of the latest application version (public or private app store or mobile device management app):

✓ Save × Cancel

Figure 53. Defining the status of application rules in the IBM Worklight Console

3. In the **Messages for multiple languages** window that opens, notice that you can upload a CSV file.

Experimental IBM
This
In case of issues


Messages for multiple languages

Default Message

Upload a CSV file containing comma separated locale code and message on each line.
Use empty locale code for a default message. Existing messages will be replaced only once you save. Language codes must conform to ISO 639-1.
Select an entry from Supported Languages list to see the message for that language.

Supported Languages

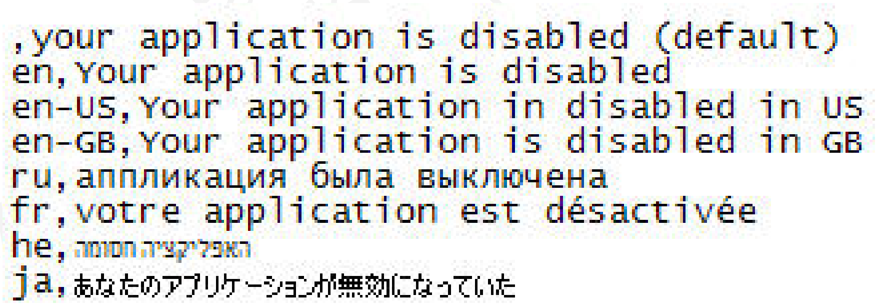
Translation

Figure 54. Defining messages for multiple languages

Such a CSV file must define a series of lines. Each line contains a locale code, such as "*fr-FR*" for French (France) or "*en*" for English, a comma, and the corresponding message text. The specified locale codes must comply with the ISO 639-1 and ISO 3166-2 standards. The first line with an empty locale defines the default message. If you did not define an alternative, or if the locale from the client matches none of the uploaded locales, this default message is displayed

Note: To create a CSV file, you must use an editor that supports UTF-8 encoding, such as NotePad. In the CSV file.

The following figure shows an example of a CSV file:

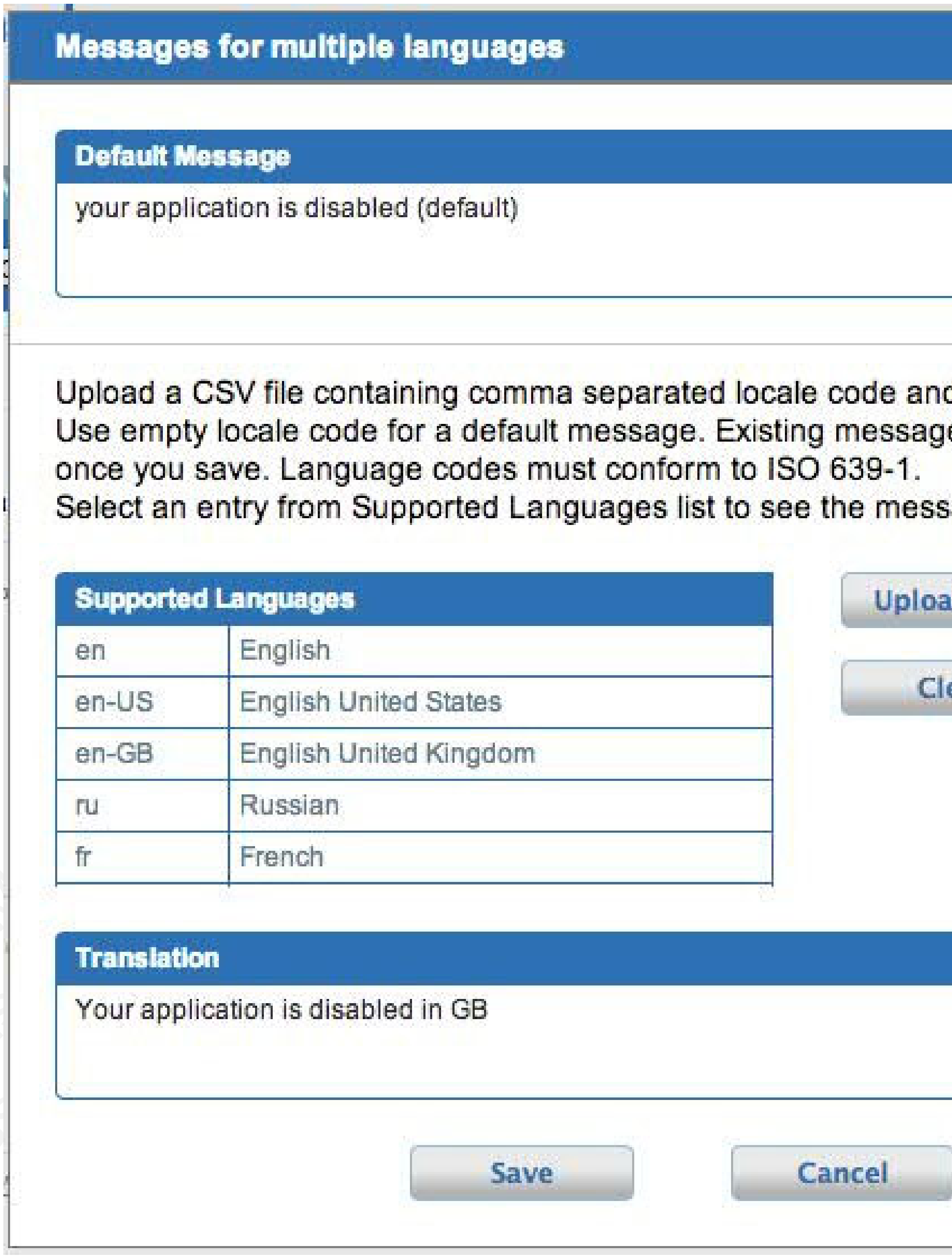


```
,your application is disabled (default)
en,Your application is disabled
en-US,Your application in disabled in US
en-GB,Your application is disabled in GB
ru,аппликация была выключена
fr,votre application est désactivée
he,האפליקציה הושבתה
ja,あなたのアプリケーションが無効になっていた
```

Figure 55. Sample CVS file

4. Click **Upload CSV** to browse and select the CSV file that you want to upload. You can see the languages that you uploaded in the **Supported Languages** list.
5. Click a language in the **Supported Languages** list to see the translation of your message in this language in the **Translation** box.

Figure 56. View of your uploaded languages, and the default message with its translation



your application is disabled (default)

Upload a CSV file containing comma separated locale code anc
Use empty locale code for a default message. Existing message
once you save. Language codes must conform to ISO 639-1.
Select an entry from Supported Languages list to see the mess

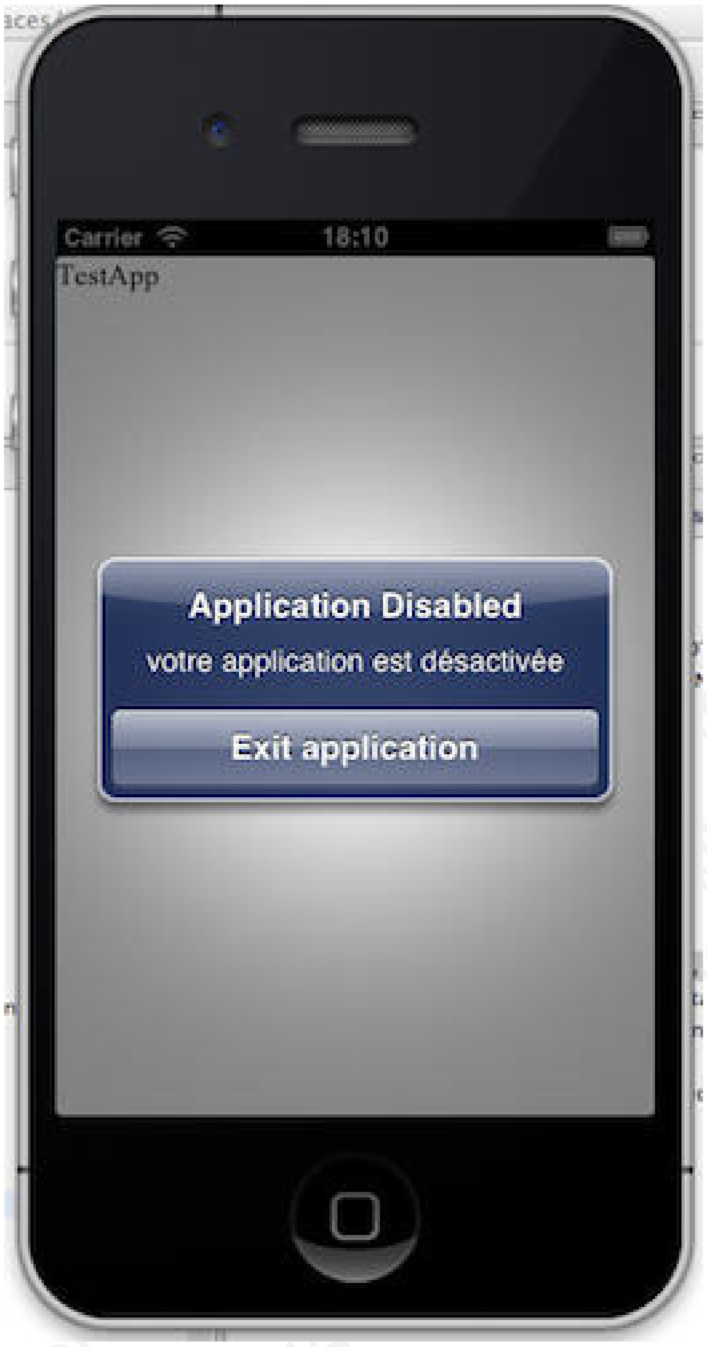
Supported Languages	
en	English
en-US	English United States
en-GB	English United Kingdom
ru	Russian
fr	French

Your application is disabled in GB

Experimental IBM
This
In case of iss

6. Optional: Click **Clear** to clear the **Supported Languages** list. This action does not clear the default message.
7. Click **Save** to save the messages that you uploaded, or **Cancel** to discard the changes and return to the IBM Worklight Console.

   Note: If you modified the default message, then the new default message shows.



Application Disabled

votre application est désactivée

This figure displays the mobile device of the user, which shows the localized message. The title and the button caption are in English. If the locale does not supply any messages, the default message is returned.
Figure 57. Application Disabled message

Controlling authenticity testing for an app

You can control authenticity testing for apps that connect to the IBM Worklight Server.

When an app first connects to the IBM Worklight Server, the server tests the authenticity of the app. This test helps to protect apps against some malware and repackaging attacks. This option is available for iPhone, iPad, and Android apps.

The application developer must configure the app to enable authenticity testing (see “Authentication configuration” on page 137 for details).

- If the app is configured with authenticity testing disabled for a specific version, then the Authenticity Testing drop down menu in the Console is disabled. An example for the iPhone environment is shown in the following figure.
- If the app is configured with authenticity testing enabled for a specific version, then the Authenticity Testing drop-down menu in the Console is enabled. An example for the Android environment is shown in the following figure.

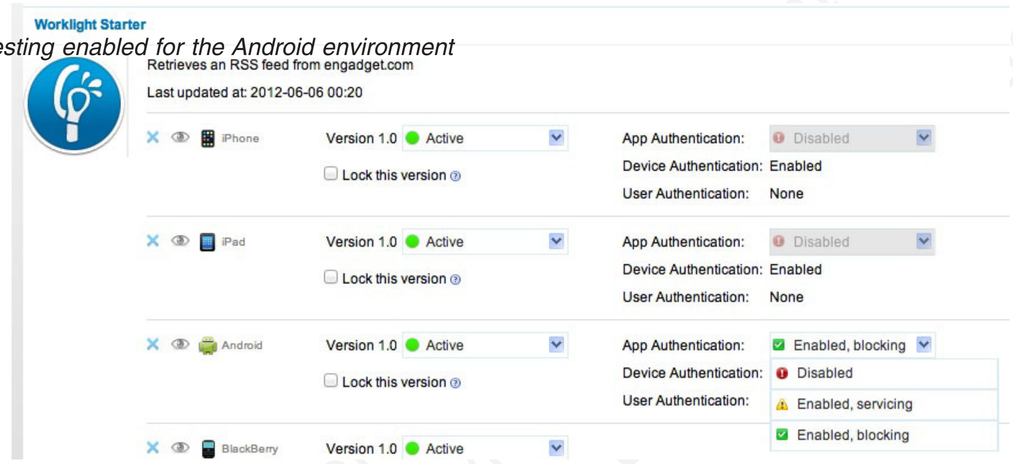


Figure 58. Authenticity testing enabled for the Android environment

Worklight Starter

Retrieves an RSS feed from engadget.com

Last updated at: 2012-06-06 00:20

iPhone	Version 1.0 Active	App Authentication:	Disabled
	Lock this version	Device Authentication:	Enabled
		User Authentication:	None
iPad	Version 1.0 Active	App Authentication:	Disabled
	Lock this version	Device Authentication:	Enabled
		User Authentication:	None
Android	Version 1.0 Active	App Authentication:	Enabled, blocking
	Lock this version	Device Authentication:	Disabled
		User Authentication:	Enabled, servicing
			Enabled, blocking
BlackBerry	Version 1.0 Active		

The menu has three options:

- **`Disabled`** – the IBM Worklight Server does not test the authenticity of the app (despite the developer's settings).
- **`Enabled, servicing`** – the IBM Worklight Server tests the authenticity of the app. If the app fails the test, the IBM Worklight Server outputs an information message to the log but services the app.
- **`Enabled, locking`** – the IBM Worklight Server tests the authenticity of the app. If the app fails the test, the IBM Worklight Server outputs an information message to the log and blocks the app.

Note: The authenticity feature is only enabled for apps that use the customer version of the IBM Worklight Development Studio. Since the non-customer version of the studio is available on the web, it is a common developer mistake to use it instead of the customer version.

Setting up existing applications with a new server version

Using applications that were built with an earlier version of IBM Worklight requires extra actions for each application.

- Upgrading to a newer version of IBM Worklight involves upgrading all the studio instances and the development environments, including the production environment.
- You must uninstall and reinstall IBM Worklight Server (for more information, see the “Installation” on page 314 topics). When you do so, the existing data in the server database (such as subscriptions to notifications) is saved.
- You must rebuild all the existing applications using the new version of IBM Worklight, and redeploy the customization `.war`, `.wlapp`, and `.adapter` files to the new server.

Experimental IBM Worklight
In case of issues, refer to the online documentation.

Federal standards support in Worklight

IBM Worklight supports Federal Desktop Core Configuration (FDCC), and United States Government Configuration Baseline (USGCB) specifications. IBM Worklight also supports the Federal Information Processing Standards (FIPS) 140-2, which is a security standard that is used to accredit cryptographic modules.

For more information about the Federal Desktop Core Configuration and United States Government Configuration Baseline, see FDCC and USGCB.

For more information about the Federal Information Processing Standards 140-2, see FIPS 140-2 support.

FDCC and USGCB support

The United States federal government mandates that federal agency desktops that run on Microsoft Windows platforms adopt Federal Desktop Core Configuration (FDCC) or the newer United States Government Configuration Baseline (USGCB) security settings.

IBM Worklight 5.0.6 was tested by using the USGCB and FDCC security settings via a self-certification process. Testing includes a reasonable level of testing to ensure that installation and core features function on this configuration.

References

For more information about the Federal Desktop Core Configuration, see FDCC.

For more information about the United States Government Configuration Baseline, see USGCB.

FIPS 140-2 support

Federal Information Processing Standards (FIPS) are standards and guidelines that are issued by the United States National Institute of Standards and Technology (NIST) for federal government computer systems. FIPS Publication 140-2 is a security standard that is used to accredit cryptographic modules.

FIPS 140-2 on the IBM Worklight server, and SSL communications with the IBM Worklight server

The IBM Worklight server runs in an application server, such as the WebSphere Application Server. The WebSphere Application Server can be configured to enforce the use of FIPS 140-2 validated cryptographic modules for inbound and outbound Secure Socket Layer (SSL) connections. Also, for the cryptographic operations that are performed by the applications by using the Java™ Cryptography Extension (JCE). Since the IBM Worklight server is an application that runs on the application server, it uses the FIPS 140-2 validated cryptographic modules for the inbound and outbound SSL connections.

When an IBM Worklight client transacts a Secure Socket Layer (SSL) connection to an IBM Worklight server, which is running on an application server that is using the FIPS 140-2 mode, the results are the successful use of the FIPS 140-2 approved cipher suite. If the client platform does not support one of the FIPS 140-2 approved cipher suites, the SSL transaction fails and the client is not able to establish an SSL connection to the server. If successful, the client uses a FIPS 140-2 approved cipher suite.

Experimental IBM Worklight offline user documentation
This document is provided as is
In case of issues, refer to the online user documentation.

Note: The cryptographic module instances that are used on the client are not necessarily FIPS 140-2 validated.
Specifically, the client and server are using the same cipher suite (SSL_RSA_WITH_AES_128_CBC_SHA) for example, but the client side cryptographic module perhaps did not go through the FIPS 140-2 validation process, whereas the server side is using FIPS 140-2 certified modules.

See the References section for links to documentation to enable FIPS 140-2 mode in WebSphere Application Server.

FIPS 140-2 on the IBM Worklight client device for protection of data at rest in JSONStore

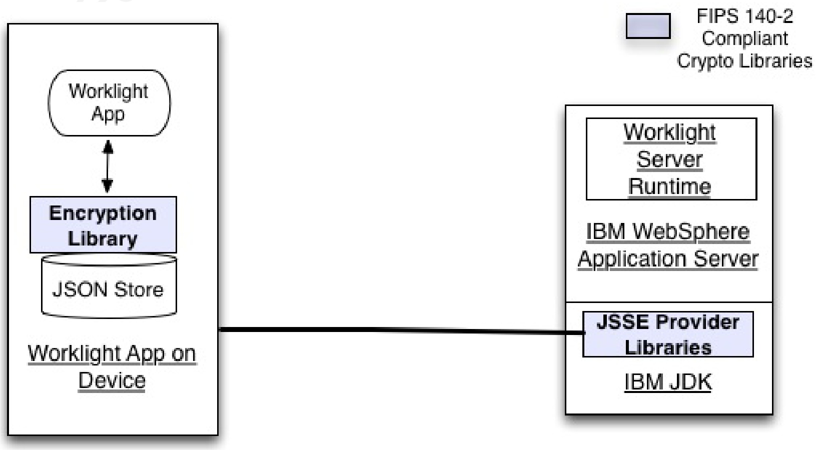
Protection of data at rest on the client device is provided by the JSONStore feature of IBM Worklight. By default, the JSONStore feature uses non-FIPS 140-2 validated libraries. But for iOS and Android devices, there is an option to use FIPS 140-2 validated libraries for the protection (encryption and decryption) of the local data that is stored by JSONStore. This option is enabled by using an OpenSSL library that achieved FIPS 140-2 validation. The required libraries and instructions for using them are provided as a module in “Getting started tutorials and samples” on page 29.

Note: There are some restrictions to be aware of:

- This FIPS 140-2 validated mode applies only to the protection (encryption) of local data that is stored by the JSONStore feature.
- It is only supported on the iOS and Android platforms.
- On Android, it is only supported on devices or simulators that use the x86 or `armv7` architectures. It is not supported on Android using `armv5` or `armv6` architectures. The reason is because the OpenSSL library used did not obtain FIPS 140-2 validation for `armv5` or `armv6` on Android.

For more information, see the module *JSONStore - Encrypting sensitive data with FIPS*, under category 5, *Advanced client-side development*, in “Getting started tutorials and samples” on page 29.

Figure 59. Example



For more information about JSONStore, see Data synchronization with JSONStore.

Experimental IBM Worklight offline user documentation. This document provided as-is. In case of [illegible]es, refer to the online user documentation.

References

For information about how to enable FIPS 140-2 mode in the WebSphere Application Server, see Federal Information Processing Standard support.

For the WebSphere Liberty Profile, there is no administrative console option to enable FIPS 140-2 mode. But FIPS 140-2 can be enabled by configuring the Java runtime to use the FIPS 140-2 validated modules. For more information, see Java Secure Socket Extension (JSSE) IBMJSSE2 Provider Reference Guide.

Reports

IBM Worklight provides an extensible mechanism for enterprises to use their own reporting tools to integrate with IBM Worklight and use the analytics data that is captured.

IBM Worklight provides raw data reports and a number of device reports that are aggregated from the raw data report table. IBM Worklight also comes bundled with a third-party Business Intelligence Report Tools (BIRT) feature, which provides a range of predefined report templates. Use the links to read about each of these options.

IBM Worklight provides two reporting mechanisms: raw data feeds and device usage reports.

Raw Data Feeds.
IBM Worklight emits raw data, which enables an OLAP system to extract the required information and present it through corporate reporting mechanisms. For more information, see "Using raw data reports" on page 375.

Device usage reports.
IBM Worklight provides reports on device usage. Device usage reports are default aggregations that are based on raw data and are provided for the benefit of organizations that do not have OLAP systems or choose not to integrate IBM Worklight with an OLAP system. For more information, see "Device usage reports" on page 378.

Note: Device usage reports are functional only in the Customer and Enterprise Editions of IBM Worklight.

BIRT reports
IBM Worklight comes bundled with predefined BIRT report to use either as they are or as templates to modify. For more information, see "Predefined BIRT Reports" on page 380.

Experimental IBM Worklight offline user documentation.
This document is provided "as-is".
In case of issues, refer to the online user documentation.

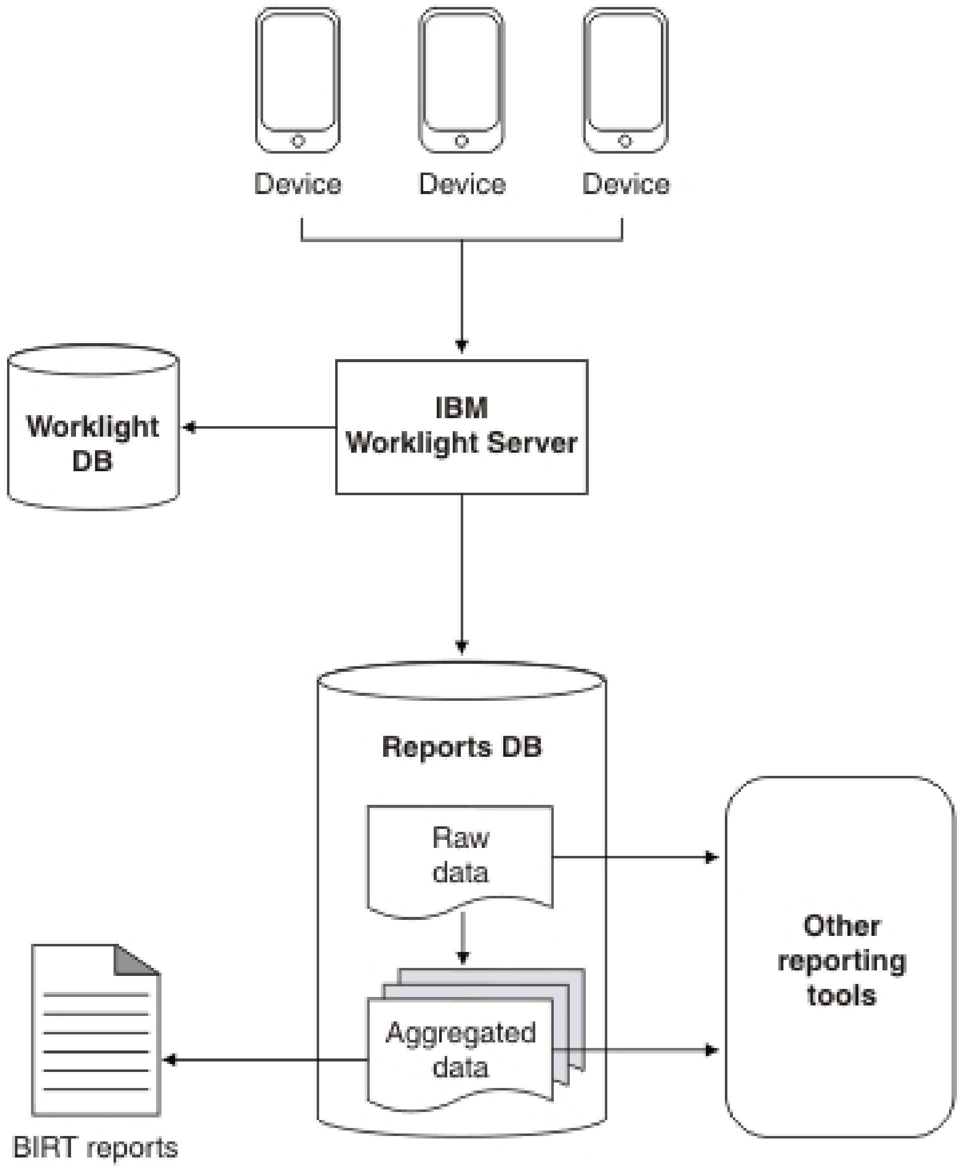


Figure 60. High-level overview of the reports architecture

Important: When you are working with report generation, you must update the `.rptdesign` file with your reports database user name and password, which are considered sensitive information. You are responsible for protecting it against unauthorized access.

Using raw data reports

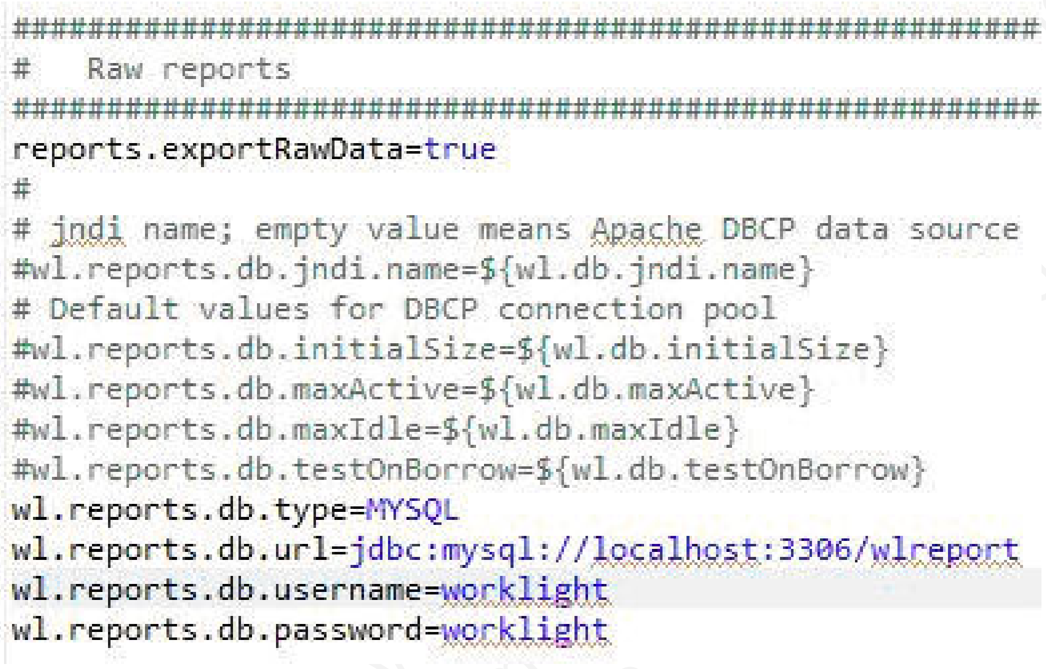
You can use the raw data reports feature to extract raw data to different databases and view it in the form of reporting tables.

About this task

Raw data reports provide you with analytics information about your applications and adapter usage, such as activity type, device information, and application version. Complete the following steps to enable the raw data reports feature:

Procedure

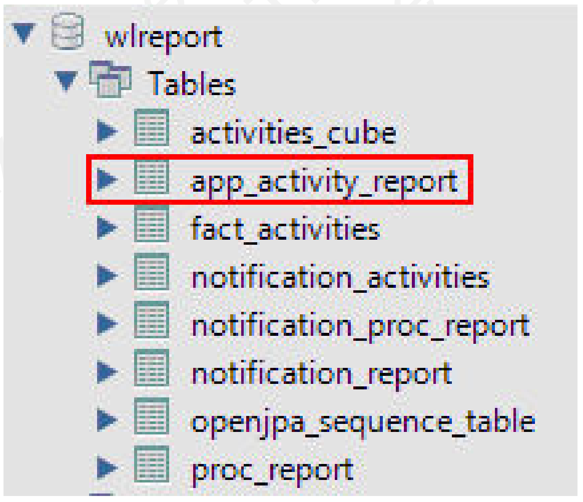
1. Ensure that the IBM Worklight Server application server is not running.
2. Create a separate database for reports. This is not mandatory but useful, because the raw data table is rapidly-populated. For more information, see "IBM Worklight database setup" on page 407.
3. Edit the `worklight.properties` file. Uncomment the **`reports.exportRawData`** property and set its value to **true**.
4. Modify the `wl.reports.db` properties to contain your database settings as shown in the following screen capture.



```
###############################################################
#   Raw reports
###############################################################
reports.exportRawData=true
#
# jndi name; empty value means Apache DBCP data source
#wl.reports.db.jndi.name=${wl.db.jndi.name}
# Default values for DBCP connection pool
#wl.reports.db.initialSize=${wl.db.initialSize}
#wl.reports.db.maxActive=${wl.db.maxActive}
#wl.reports.db.maxIdle=${wl.db.maxIdle}
#wl.reports.db.testOnBorrow=${wl.db.testOnBorrow}
wl.reports.db.type=MYSQL
wl.reports.db.url=jdbc:mysql://localhost:3306/wlreport
wl.reports.db.username=worklight
wl.reports.db.password=worklight
```

5. Ensure that the **`wl.reports.db.url`** property contains the URL of the database you are planning to use for raw data.
6. Restart your application server.

   The `app_activity_report` table of the raw data database is populated with data as you begin using your applications and adapters.



The raw data app_activity_report table contains the following information:

Column	Description
ACTIVITY_TIMESTAMP	Time of entry (in UTC)
GADGET_NAME	IBM Worklight Application name
GADGET_VERSION	Application version
ACTIVITY	Activity type
ENVIRONMENT	Application environment name (iPhone, Android, etc.)
SOURCE	User identifier
ADAPTER	IBM Worklight adapter name
PROC	IBM Worklight adapter procedure name
USERAGENT	User agent from HTTP header of client device
SESSION_ID	A unique identifier for the user's session on the server
IP_ADDRESS	IP address of the client
DEVICE_ID n	A unique device ID
DEVICE_MODEL	Manufacturer model, for example Galaxy I9000
DEVICE_OS	Device operating system version

Possible activities include:

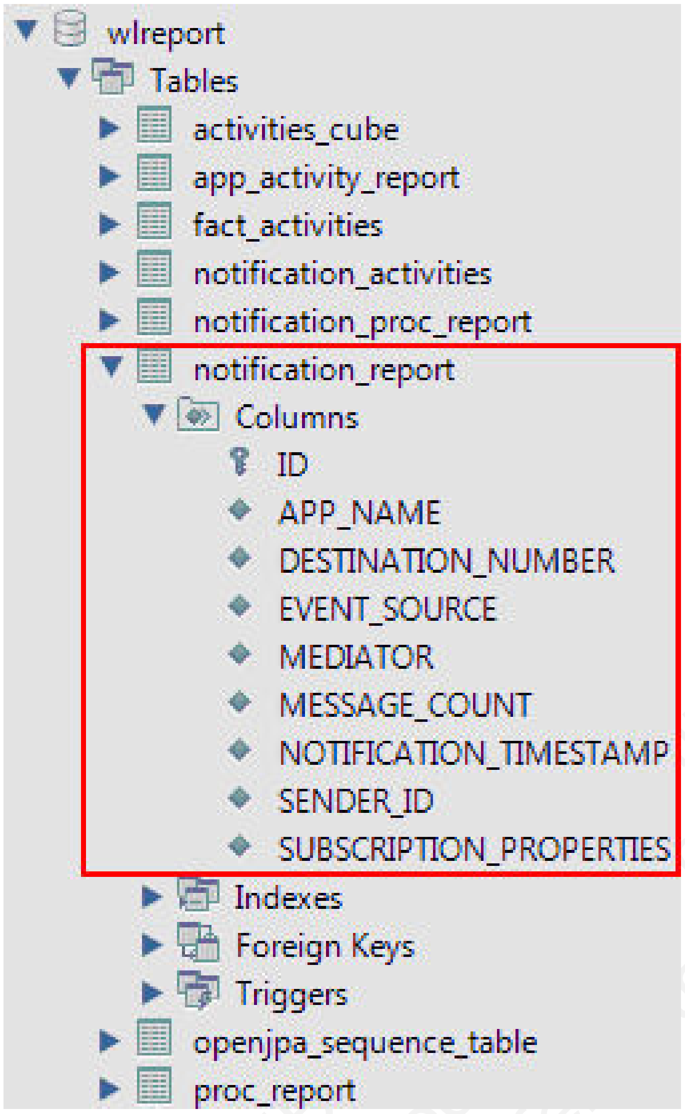
Activity	Description
Init	Application initialization
Login	Successful authentication in using the application
Adoption New	Not supported in version 5.0
Adoption	Not supported in version 5.0
Query	Procedure call to an adapter
Logout	User logout

In addition to predefined activity types, custom activities can be logged by using **WL.Client.logActivity("custom-string")** APIs.

Important: Worklight raw data feed can increase rapidly. The data is typically used by a BI system such as Cognos® or Business Objects. It is the administrator's responsibility to purge built-in tables periodically.

In addition to the app_activity_report table, the raw data engine also populates the notification_report table. This raw data table contains information about notifications that are sent from SMS event sources.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

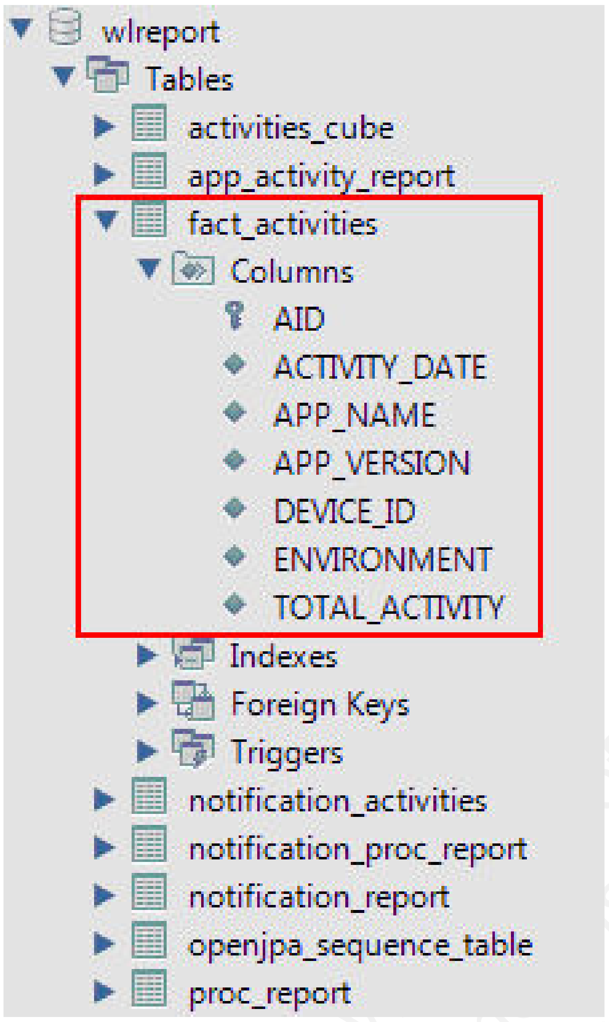


- wlreport
  - Tables
    - activities_cube
    - app_activity_report
    - fact_activities
    - notification_activities
    - notification_proc_report
    - notification_report
      - Columns
        - ID
        - APP_NAME
        - DESTINATION_NUMBER
        - EVENT_SOURCE
        - MEDIATOR
        - MESSAGE_COUNT
        - NOTIFICATION_TIMESTAMP
        - SENDER_ID
        - SUBSCRIPTION_PROPERTIES
      - Indexes
      - Foreign Keys
      - Triggers
    - openjpa_sequence_table
    - proc_report

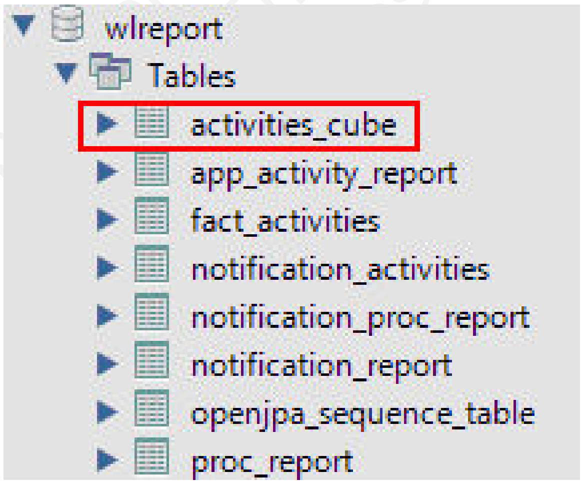
Device usage reports

For simpler and faster access to the reports data, the IBM Worklight Server runs an analytics data processor task at a default time interval of every 24 hours.

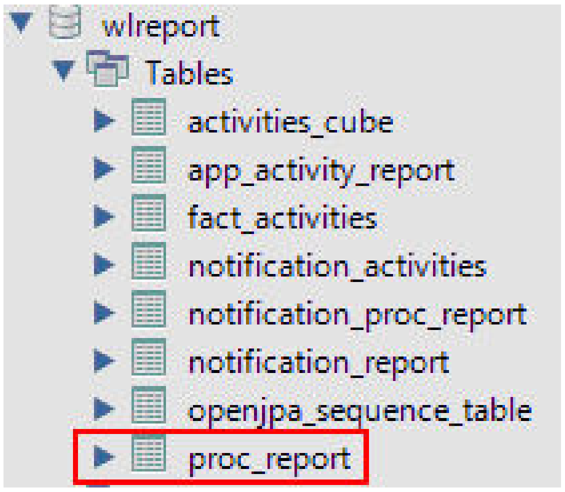
The analytics data processor task retrieves raw entries for the specified time interval from the `app_activity_report` table and processes them to populate the `fact_activities` table.



The `fact_activities` table contains a total activity count (number of logged actions) per application, application version, device, and environment. The `fact_activities` data is also processed and put into the `activities_cube` table. This table has the same structure as the `fact_activities` table and only contains records for the last 30 days.

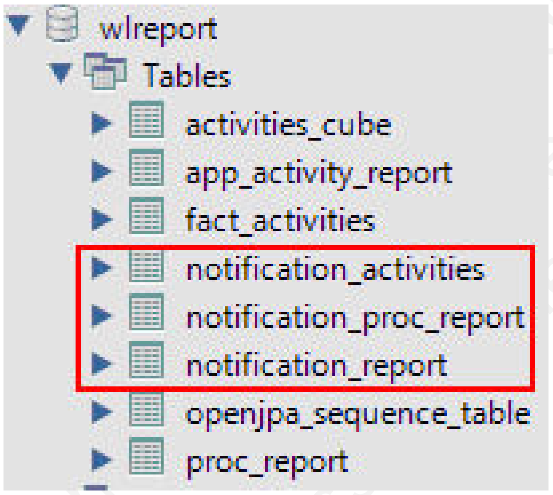


Each time the above data processing is performed, a timestamp is added to a `proc_report` table with the processing result (timestamp and number of processed entries).



- wlreport
  - Tables
    - activities_cube
    - app_activity_report
    - fact_activities
    - notification_activities
    - notification_proc_report
    - notification_report
    - openjpa_sequence_table
    - proc_report

In addition, `notification_report` table data is also processed to populate the `notification_activities` table with consolidated data. This is populated in the same way as the `fact_activities` table. Every time the `notification_report` table data is processed, an entry is added to the `notification_proc_report` table, which is similar to the `proc_report` table.



- wlreport
  - Tables
    - activities_cube
    - app_activity_report
    - fact_activities
    - notification_activities
    - notification_proc_report
    - notification_report
    - openjpa_sequence_table
    - proc_report

The processing interval can be modified by adding the following property to your `worklight.properties` file and setting the required interval in seconds.

```
# Default interval value for analytics processing task
wl.db.factProcessingInterval=86400
```

Predefined BIRT Reports

You can use predefined BIRT reports to generate and display information about mobile devices and usage.

IBM Worklight generates raw reports, which are stored in an `app_activity_report` table. IBM Worklight also includes device usage reports, which are aggregations of data from the `app_activity_report`, and are described in “Using raw data reports” on page 375 on page 375. Users can view or extract data from the app_activity_report table or from the device usage reports, and process it using their own business intelligence systems.

For users with no existing business intelligence analysis system, IBM Worklight provides a selection of predefined Business Intelligence Reporting Tool (BIRT) reports. BIRT is a third-party tool, and is not created or supported by IBM. IBM Worklight provides several *.rptdesign files that contain logic that allows you to connect to the reports database, pull data from device usage tables, process, and display the data.

IBM Worklight Customer and Enterprise Editions include the following predefined BIRT reports:

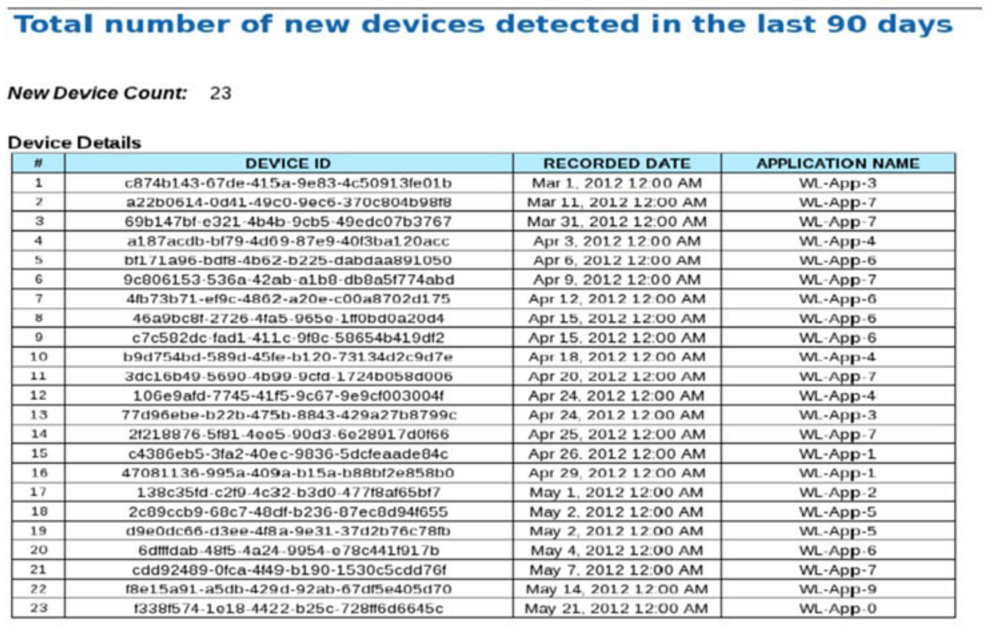
Table 101. Predefined BIRT reports

Report Name	Description	Report file name
Active Users	Active users in last 30 days.	report_active_users.rptdesign
Daily Hits	The daily aggregated hits for last 30 days. Any action from the user/device that caused a request to the server is counted as a hit. This number, aggregated over a day, equals the daily hits.	report_daily_hits.rptdesign
Daily Visits	The number of discreet visits by separate user/device in last 30 days. All actions by a user/device that caused one or more requests to the server within a day is counted as a visit.	report_daily_visits.rptdesign
Environment Usage	Application version and application environment used: number of visits that were recorded in the last 30 days.	report_environment_usage.rptdesign
New Devices	A record of unique devices that were connected in the last 30 days.	report_new_devices.rptdesign
Notification Messages Per Day	Number of messages sent each day in the past 90 days per data source.	report_notification_messages_per_day.rptdesign
Notification Messages Per Source	Total number of messages that were sent in the last 90 days per data source.	report_notification_messages_per_source.rptdesign

Experimental IBM Worklight online user documentation.
This document is provided "as is".
In case of issues, refer to the online user documentation.

Table 101. Predefined BIRT reports (continued)

Report Name	Description	Report file name
License Total New Device Count	A record of unique devices that were connected over a specified period (90 days as default), for licensing purposes.	`report_license_total_device_count.rptdesign`



Total number of new devices detected in the last 90 days

New Device Count: 23

Device Details

#	DEVICE ID	RECORDED DATE	APPLICATION NAME
1	c874b143-67de-415a-9e83-4c50913fe01b	Mar 1, 2012 12:00 AM	WL-App-3
2	a22b0614-0d41-49c0-9ec6-370c804b98f8	Mar 11, 2012 12:00 AM	WL-App-7
3	69b147bf-e321-4b4b-9cb5-49edc07b3767	Mar 31, 2012 12:00 AM	WL-App-7
4	a187acdb-bf79-4d69-87e9-40f3ba120acc	Apr 3, 2012 12:00 AM	WL-App-4
5	bf171a96-bdf8-4b62-b225-dabdaa891050	Apr 6, 2012 12:00 AM	WL-App-6
6	9c806153-536a-42ab-a1b8-db8a5f774abd	Apr 9, 2012 12:00 AM	WL-App-7
7	4fb73b71-ef9c-4862-a20e-c00a8702d175	Apr 12, 2012 12:00 AM	WL-App-6
8	46a9bc8f-2726-4fa5-965e-1ff0bd0a20d4	Apr 15, 2012 12:00 AM	WL-App-6
9	c7c582dc-fad1-411c-9f8c-58654b419df2	Apr 15, 2012 12:00 AM	WL-App-6
10	b9d754bd-589d-45fe-b120-73134d2c9d7e	Apr 18, 2012 12:00 AM	WL-App-4
11	3dc16b49-5690-4b99-9cfd-1724b058d006	Apr 20, 2012 12:00 AM	WL-App-7
12	106e9afd-7745-41f5-9c67-9e9cf003004f	Apr 24, 2012 12:00 AM	WL-App-4
13	77d96ebe-b22b-475b-8843-429a27b8799c	Apr 24, 2012 12:00 AM	WL-App-3
14	2f218876-5f81-4ee5-90d3-6e28917d0f66	Apr 25, 2012 12:00 AM	WL-App-7
15	c4386eb5-3fa2-40ec-9836-5dcfeaade84c	Apr 26, 2012 12:00 AM	WL-App-1
16	47081136-995a-409a-b15a-b88bf2e858b0	Apr 29, 2012 12:00 AM	WL-App-1
17	138c35fd-c2f9-4c32-b3d0-477f8af65bf7	May 1, 2012 12:00 AM	WL-App-2
18	2c89ccb9-68c7-48df-b236-87ec8d94f655	May 2, 2012 12:00 AM	WL-App-5
19	d9e0dc66-d3ee-4f8a-9e31-37d2b76c78fb	May 2, 2012 12:00 AM	WL-App-5
20	6dfffdab-48f5-4a24-9954-e78c441f917b	May 4, 2012 12:00 AM	WL-App-6
21	cdd92489-0fca-4f49-b190-1530c5cdd76f	May 7, 2012 12:00 AM	WL-App-7
22	f8e15a91-a5db-429d-92ab-67df5e405d70	May 14, 2012 12:00 AM	WL-App-9
23	f338f574-1e18-4422-b25c-728ff6d6645c	May 21, 2012 12:00 AM	WL-App-0

Figure 61. An example of a report generated by BIRT, in this case report_license_total_device_count.rptdesign

There are several ways of viewing predefined reports, by using one of the following.

- The Eclipse report designer plug-in. For instructions, see "BIRT in Eclipse" on page 388
- The BIRT Viewer application that is installed on your Tomcat, WebSphere® Full Profile or WebSphere Liberty Profile application server.

Installing BIRT on Apache Tomcat

You can use the Business Intelligence Reporting Tool (BIRT) to generate and render report content. You can view this content either by using an Eclipse plug-in, or an application server and browser.

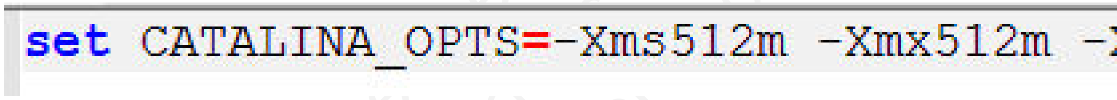
About this task

The IBM Worklight installation contains a number of predefined BIRT reports. These reports are configurable XML files that are designed to retrieve and present data from the IBM Worklight reports database tables. These files have an `.rptdesign` extension.

Complete the following steps to set up the BIRT Reports for viewing in an Apache Tomcat application server. For information about how to set up the BIRT Reports on other application servers, refer to the BIRT Reports website at Birt Tools.

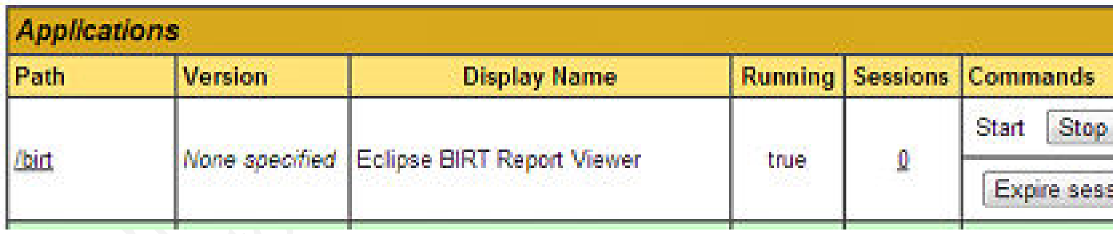
Procedure

1. Ensure that your Tomcat instance is NOT running.
2. Download the BIRT Reports runtime archive from Birt Report Downloads.
3. Unzip the BIRT Reports runtime archive.
4. Copy the `WebViewerExample` folder to the `webapps` folder of your Tomcat server.
5. Rename the `WebViewerExample` folder to `birt` (this step is optional, just to simplify later execution).
6. Copy your database `jdbc` connector jar file package to the Tomcat `\lib` folder (if you are using the same Tomcat instance that is running IBM Worklight server the `jdbc` connector package is already in the `\lib` folder).
7. In some cases, Tomcat might not have enough memory allocated to run BIRT Reports. To resolve this problem, edit the `catalina.bat` file under your Tomcat `\bin` folder and add the following line at the start of it. You might want to consult with your IT manager before adding it.



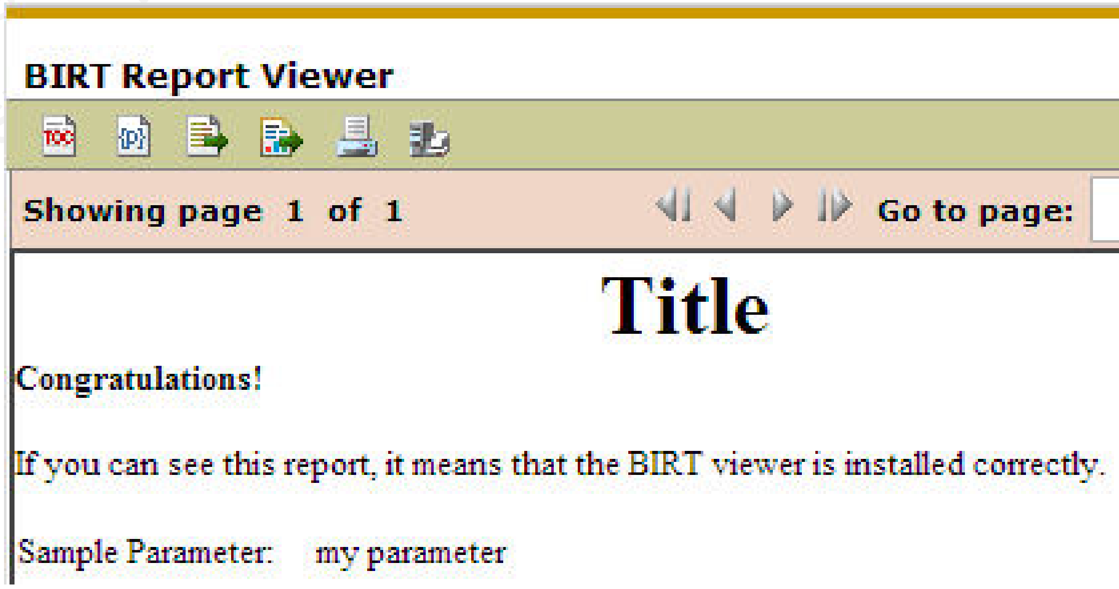
```
set CATALINA_OPTS=-Xms512m -Xmx512m -X
```

8. Restart your Tomcat.
9. Go to theTomcat manager application at `http://your-server/manager/` to verify that the BIRT Reports application started.



Applications					
Path	Version	Display Name	Running	Sessions	Commands
/birt	None specified	Eclipse BIRT Report Viewer	true	0	Start Stop Expire sess

10. Your BIRT Reports viewer application is accessible at `http://your-server/birt/`.
11. You can test the BIRT Reports installation by going to `http://your-server/birt/frameset?__report=test.rptdesign&sample=my+parameter`.



BIRT Report Viewer

Showing page 1 of 1 Go to page:

Title

Congratulations!

If you can see this report, it means that the BIRT viewer is installed correctly.

Sample Parameter: my parameter

Installing BIRT on WebSphere Application Server Liberty Profile

You can install Business Intelligence Reporting Tools (BIRT) on the WebSphere® Application Server Liberty Profile.

Procedure

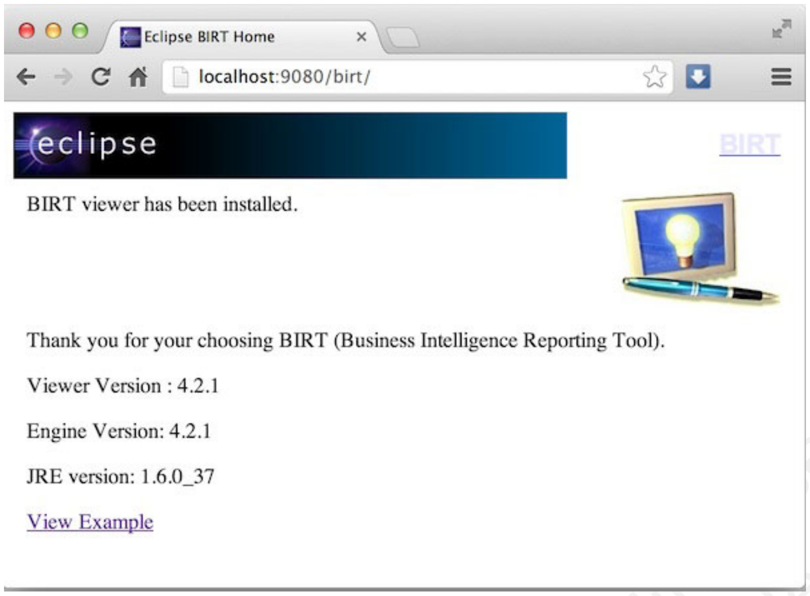
1. Verify that your WebSphere Application Server Liberty Profile instance is not running.
2. Go to your WebSphere Application Server Liberty Profile folder and create two folders as follows:
   - apps
   - libs
3. Locate the jdbc connector driver that you will be using and copy it to the `libs` folder.
4. Download the latest release of BIRT runtime from http://download.eclipse.org/birt/downloads/
5. Extract the downloaded file and go to the extracted folder.
6. Rename `WebViewerExample` folder to `birt.`
7. Go to the folder `birt\WEB-INF\lib` and delete the following files.
   - org.apache.xerces*.jar
   - org.apache.xml.resolver*.jar
   - org.apache.xml.serializer*.jar

   Setup the BIRT Viewer application on a Liberty instance by doing the following.
8. Copy the `birt` folder to `{your-liberty-instance}\usr\servers\{your-server-name}\apps\`
9. Update the `server.xml` file of your Liberty server profile.
10. Make sure that the JSP feature is enabled.
11. Add an application definition.
12. Add `classloader` with `privateLibrary` the definitions configured to point to your JDBC connector driver.

```
<server description="new server">
    <featureManager>
        <feature>jsp-2.2</feature>
    </featureManager>

    <httpEndpoint   id="defaultHttpEndpoint"
                    host="*"
                    httpPort="9080"
                    httpsPort="9443" />

    <application    id="birt"
                    name="birt"
                    type="war"
                    location="${server.config.dir}/apps/birt"
                    context-root="/birt">
        <classloader delegation="parentLast">
            <privateLibrary>
                <fileset    dir="${server.config.dir}/libs"
                            includes="mysql-connector*.jar" />
            </privateLibrary>
        </classloader>
    </application>
</server>
```

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues refer to the online user documentation.

13. Start your Liberty instance.
14. Browse to http://server:port/birt The BIRT Viewer landing page appears.



BIRT viewer has been installed.

Thank you for your choosing BIRT (Business Intelligence Reporting Tool).

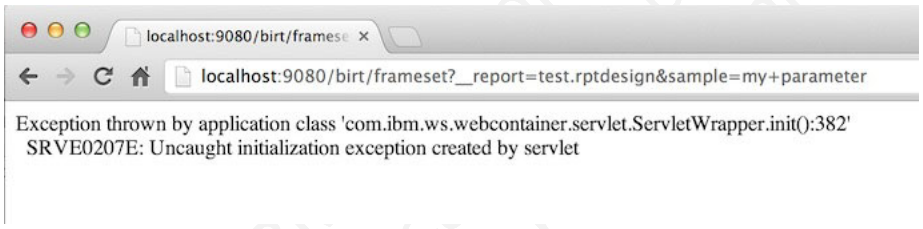
Viewer Version : 4.2.1

Engine Version: 4.2.1

JRE version: 1.6.0_37

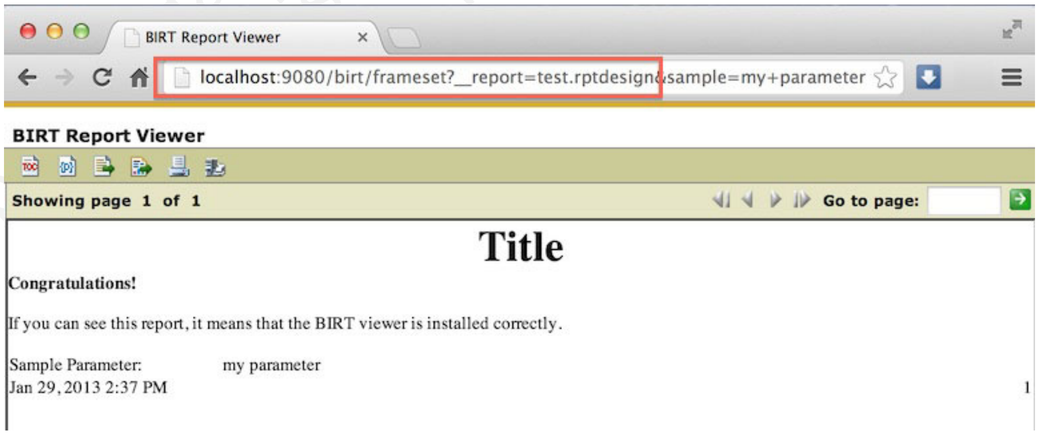
View Example

15. Click **View Example** link.
16. If you see the following error message, refresh your page.


localhost:9080/birt/framese
localhost:9080/birt/frameset?__report=test.rptdesign&sample=my+parameter

```
Exception thrown by application class 'com.ibm.ws.webcontainer.servlet.ServletWrapper.init():382'
 SRVE0207E: Uncaught initialization exception created by servlet
```

17. The BIRT Viewer sample report appears.


BIRT Report Viewer
localhost:9080/birt/frameset?__report=test.rptdesign&sample=my+parameter
BIRT Report Viewer
Showing page 1 of 1
Go to page:

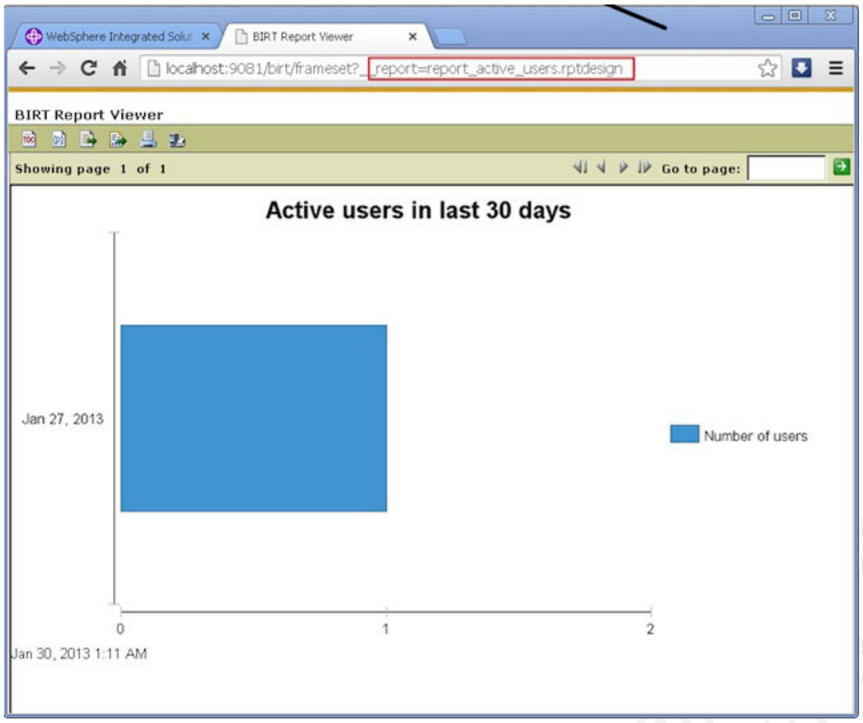
Title

Congratulations!

If you can see this report, it means that the BIRT viewer is installed correctly.

Sample Parameter: my parameter
Jan 29, 2013 2:37 PM 1

Note `test.rptdesign` in the page URL. You can replace this with the name of other **rptdesign** files, as shown here for example:


WebSphere Integrated Solut
BIRT Report Viewer
localhost:9081/birt/frameset?__report=report_active_users.rptdesign
BIRT Report Viewer
Showing page 1 of 1
Go to page:
Active users in last 30 days
Jan 27, 2013
Number of users
0
1
2
Jan 30, 2013 1:11 AM

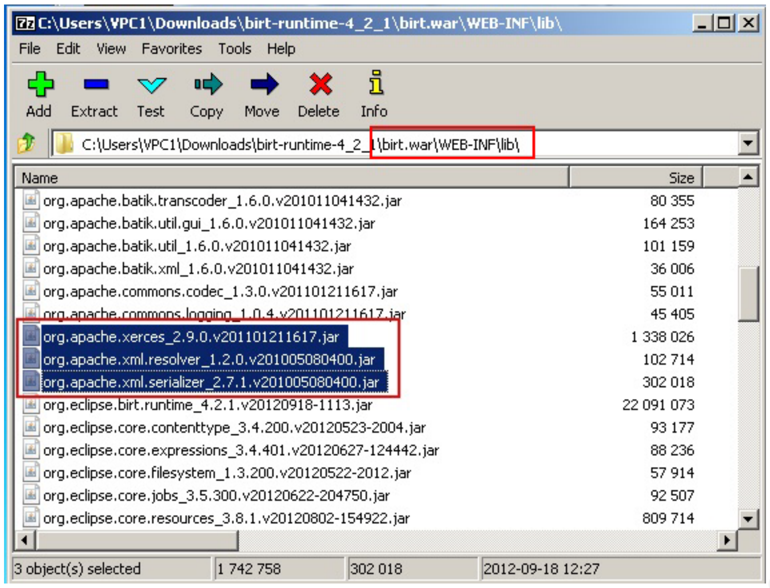
Installing BIRT on WebSphere Application Server Full Profile

You can install Business Intelligence Reporting Tools (BIRT) on the WebSphere® Application Server Full Profile.

Procedure

1. Download the BIRT package and extract the contents.
2. From the folder `birt-runtime-version\WebViewerExample\WEB-INF\lib`, delete (or remove) the following packages:
   - `org.apache.xerces.jar`
   - `org.apache.resolver.jar`
   - `org.apache.serializer.jar`

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues refer to the online user documentation.



```
C:\Users\VPC1\Downloads\birt-runtime-4_2_1\birt.war\WEB-INF\lib\

Name                                                      Size
org.apache.batik.transcoder_1.6.0.v201011041432.jar       80 355
org.apache.batik.util.gui_1.6.0.v201011041432.jar        164 253
org.apache.batik.util_1.6.0.v201011041432.jar            101 159
org.apache.batik.xml_1.6.0.v201011041432.jar              36 006
org.apache.commons.codec_1.3.0.v201101211617.jar          55 011
org.apache.commons.logging_1.0.4.v201101211617.jar        45 405
org.apache.xerces_2.9.0.v201101211617.jar              1 338 026
org.apache.xml.resolver_1.2.0.v201005080400.jar          102 714
org.apache.xml.serializer_2.7.1.v201005080400.jar        302 018
org.eclipse.birt.runtime_4.2.1.v20120918-1113.jar     22 091 073
org.eclipse.core.contenttype_3.4.200.v20120523-2004.jar   93 177
org.eclipse.core.expressions_3.4.401.v20120627-124442.jar 88 236
org.eclipse.core.filesystem_1.3.200.v20120522-2012.jar    57 914
org.eclipse.core.jobs_3.5.300.v20120622-204750.jar        92 507
org.eclipse.core.resources_3.8.1.v20120802-154922.jar    809 714

3 object(s) selected | 1 742 758 | 302 018 | 2012-09-18 12:27
```

Figure 62. Deleting three files

3. Use a .war command to package the directory `WebViewerExample` into a war file named `birt.war`
4. Start the WebSphere Server.
5. Open the console web page.
6. Log in.
7. From the console, install BIRT package by installing `birt.war` from the runtime download.
8. Click **Enterprise Applications** in left menu.
9. Click the name of the deployed application, `birt_war`, to enter the configuration page.
10. Under the heading **Modules**, click **Manage Modules**.
11. In the Module list, click **Eclipse BIRT Report Viewer**.
12. In the **General Properties** page, under **Class loader order**, select **Classes loaded with application class loader first** option.
13. Click **OK**.
14. Save the Master Configuration.

Configuring BIRT Reports For Your Application Server

To use BIRT reports, you must update them with your web application server settings.

About this task

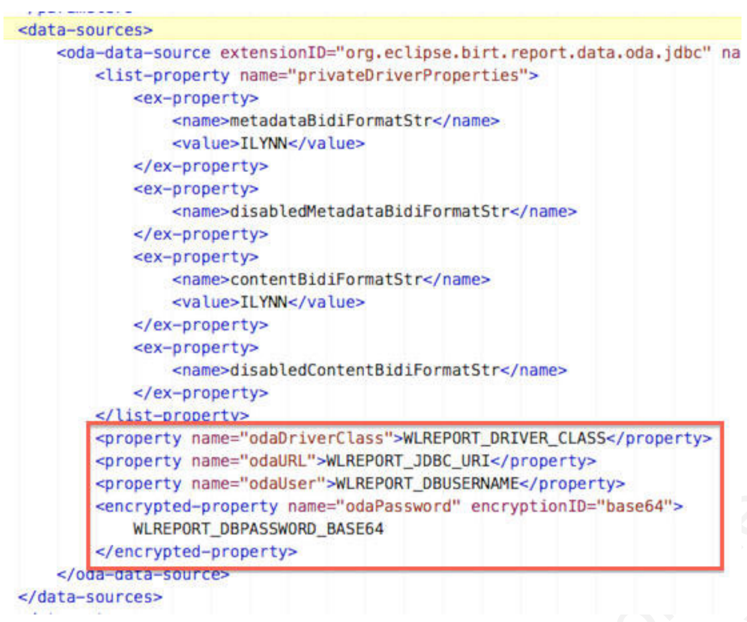
Before using the BIRT Viewer application to see predefined reports, you must edit them to adjust the IBM Worklight Reports database settings, and then copy the reports to a specific folder on the application server.

Procedure

1. Go to your IBM Worklight Server installation folder created by the IBM Installation Manager.
2. Locate the `\report-templates\` folder, which contains a set of `.rptdesign` files.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues refer to the online user documentation.

3. Copy all of the files with the `.rptdesign` extension from the `\report-templates\` folder to your server web applications folder.
4. Edit each `.rptdesign` file as needed and adjust the **`<data-sources>`** element with the properties of your Worklight reports database.



```
<data-sources>
    <oda-data-source extensionID="org.eclipse.birt.report.data.oda.jdbc" na
        <list-property name="privateDriverProperties">
            <ex-property>
                <name>metadataBidiFormatStr</name>
                <value>ILYNN</value>
            </ex-property>
            <ex-property>
                <name>disabledMetadataBidiFormatStr</name>
            </ex-property>
            <ex-property>
                <name>contentBidiFormatStr</name>
                <value>ILYNN</value>
            </ex-property>
            <ex-property>
                <name>disabledContentBidiFormatStr</name>
            </ex-property>
        </list-property>
        <property name="odaDriverClass">WLREPORT_DRIVER_CLASS</property>
        <property name="odaURL">WLREPORT_JDBC_URI</property>
        <property name="odaUser">WLREPORT_DBUSERNAME</property>
        <encrypted-property name="odaPassword" encryptionID="base64">
            WLREPORT_DBPASSWORD_BASE64
        </encrypted-property>
    </oda-data-source>
</data-sources>
```

5. Make sure that BIRT Viewer application is installed and running on your application server
6. To view or edit a BIRT Report, go to the path `http://your-server/birt/frameset?__report=[report name].rptdesign.`, where `[report name].rtpdesign` represents one of the following files:
   - `report_active_users.rptdesign`
   - `report_daily_hits.rptdesign`
   - `report_daily_visits.rptdesign`
   - `report_environment_usage.rptdesign`
   - `report_license_total_device_count.rptdesign`
   - `report_new_devices.rptdesign`
   - `report_notification_messages_per_day.rptdesign`
   - `report_notification_messages_per_source.rptdesign`

BIRT in Eclipse

BIRT installed in Eclipse displays reports through the Eclipse interface.

You can install Business Intelligence Reporting Tools (BIRT) as either a stand-alone instance of Eclipse, or as a plug-in added to your existing IBM Worklight Eclipse instance, or any other instance of Eclipse. Each of these choices has potential advantages, depending on your needs.

Installing a stand-alone Eclipse instance means having a dedicated tool for creating reports. This option involves downloading an Eclipse installer that comes with BIRT included.

Experimental IBM Worklight online user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Installing BIRT as a plug-in to your existing Eclipse instance that is running IBM Worklight can provide you with a more integrated interface, for both IBM Worklight and reports. Use the following links to select the option you want to install.

Installing BIRT in Stand-alone Eclipse

You can install BIRT including the BIRT Report Designer in a stand-alone instance of Eclipse as a dedicated reporting tool.

About this task

To use the BIRT Report Designer in a stand-alone, dedicated instance of Eclipse, follow these steps:

Procedure

1. In your web browser, go to http://www.eclipse.org/downloads/
2. Download the **`Eclipse IDE for Java and Report Developers`**
3. Follow the instructions in the installation package. Eclipse as well as the BIRT components, including the Report Designer, are installed.

Installing BIRT in Worklight Eclipse

You can install BIRT in the instance of Eclipse on which IBM Worklight is running, and use the Report Designer as an integrated tool.

About this task

To install BIRT in the existing instance of Eclipse that is running Worklight, follow these steps:

Procedure

1. Click **Help** > **Install new software**
2. In the **Work with...** dropbox, select `http://download.eclipse.com/release/indigo Note that you might have a different Eclipse version installed, replace indigo with juno or another version as needed.`
3. Select `Business Intelligence Reporting and Charting`
4. Click **Next** and follow the installation instructions. When the installation is completed, you must install the reports.
5. Click **Window** > **Open perspective** > **Other...**
6. Select the **Report Design** perspective
7. Click **File** > **New** > **Project**
8. Select **Report project** and click **Next**
9. Enter a project name and click **Finish**
10. Using the import command, go to your IBM Worklight Server installation folder created by IBM Installation Manager.
11. Locate the `\report-templates\` folder, which contains a set of `.rptdesign` files.
12. Import all files with the suffix **`.rptdesign`** from the `\report-templates\` folder into the Eclipse project. Eclipse comes with a bundled driver for Apache Derby database. If you use another database type, you must add a JDBC connector driver manually.
13. Click **Manage Drivers...**
14. Click **Add...** and add the JDBC connector driver package to communicate with your Worklight reports database

Experimental IBM Worklight online user documentation. This document is provided as-is. In case of issues, refer to the online user documentation.

15. Select **Driver Class** and adjust the rest of your database settings
16. Click **Test Connection...** to validate that database settings are correct.

Viewing BIRT reports in Eclipse

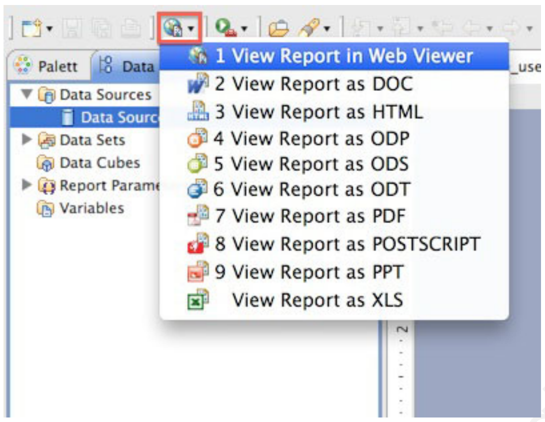
With BIRT in Eclipse, you can view reports through the Eclipse interface.

About this task

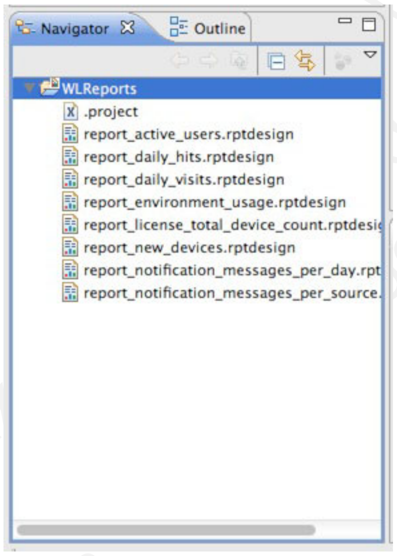
To view BIRT reports in Eclipse, follow these steps:

Procedure

1. Click the black arrow next to **View Report**.



2. Select the output format for your report
3. See the report.


Navigator
Outline
WLReports
.project
report_active_users.rptdesign
report_daily_hits.rptdesign
report_daily_visits.rptdesign
report_environment_usage.rptdesign
report_license_total_device_count.rptdesi
report_new_devices.rptdesign
report_notification_messages_per_day.rpt
report_notification_messages_per_source.

Notification reports database schema

The IBM Worklight platform uses a database schema to store the notification reports data derived from the raw data.

Worklight offline user documentation. This document is provided "as is". For updates, refer to the online user documentation.

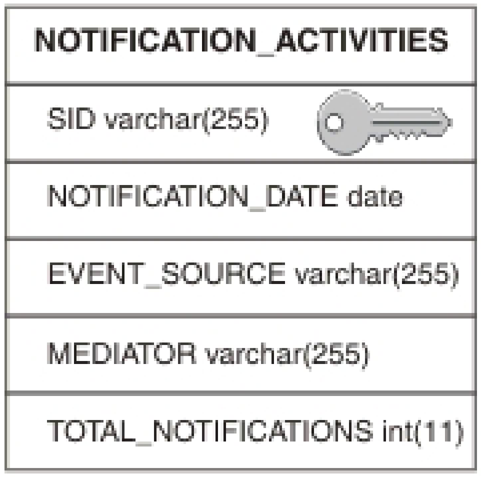


Figure 63. NOTIFICATION_ACTIVITIES schema

A notification activities table is populated to simplify the use of report construction. This notification activities table, **NOTIFICATION_ACTIVITIES**, is populated as part of the analytics setup.

High availability

High availability is provided through clustering, the ability to provide multiple IBM Worklight Servers acting together.

Multiple IBM Worklight Servers enable horizontal scaling of the software as well as the prevention of a single point of failure.

Clustering

The IBM Worklight Server creates a cluster by deploying multiple servers that share the database instance.

The basic setup consists of the load balancer, the cluster nodes, and a database that is shared by the cluster nodes.

All cluster nodes are identical, that is, the content of the installation folder is the same in all nodes. Cluster nodes do not synchronize with each other at run time. All shared runtime data is in the database so that database changes made through one node are immediately available to all other nodes. The exception is the customization WAR, which is not held in the database, and each node must have its own copy and can be secured individually. With WebSphere Application Server Network Deployment, you can use built in clustering support for distributing the IBM Worklight customization WAR (and the Worklight Shared library). For more information, see the IBM WebSphere Application Server V8 user documentation.

IBM Worklight Servers can run on a VMware virtual machine. In such cases, one machine image is created and then deployed again and again.

IBM Worklight is *stateful*. It caches session state within the server memory. The result is that if one Worklight Server is taken offline, active user sessions are lost and the client is asked to log on again.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues refer to the online user documentation.

Configuring the load balancer

You can use hardware-based or software-based load balancers.

If you do not want to use a hardware-based load balancer, you can use a simpler, software-based load balancer or reverse proxy such as the Apache Tomcat web server. Any load balancer that can support the following features is adequate:

- Sticky session (recommended configuration)
- Reverse proxy capabilities
- Optional: SSL Acceleration

Configuration of the load balancer depends on the vendor and is not covered in this document. It is common to define the range of the node addresses so that they can be added or deleted dynamically.

Adding a node to the cluster

Follow the instructions for creating an IBM Worklight Server to add a node to the cluster.

About this task

You can add a node to the cluster, by following the instructions for creating an IBM Worklight Server:

Procedure

1. Add the IP address of the node to the load balancer or use an existing address from a range that was pre-allocated to IBM Worklight Servers.
2. Install the IBM Worklight Server.
3. Apply the customization `WAR`.

Upgrading a production cluster

Upgrading a production cluster is part of the upgrade procedure when upgrading your IBM Worklight Server. The client-server protocol supported by the IBM Worklight V5.0 Server is different from the one supported by the IBM Worklight V4.2 Server and, as a result, an IBM Worklight V5.0 Server cannot service applications that are built on top of IBM Worklight V4.2.

About this task

Before you upgrade a production cluster, you must choose how to migrate users from V4.2-based to V5.0-based apps from the following options:

Option 1
You can maintain two server clusters in production:

- A V5.0 server cluster for the V5.0-compatible apps.
- A V4.2 server cluster for the V4.2-compatible apps.

You must then remote disable the V4.2-compatible apps on the V4.2 server, directing users to download the V5.0-compatible apps.

Option 2
You can upload both the original V4.2-compatible apps as well as the V5.0-compatible apps to the V5.0 server.

You must then remote disable the V4.2-compatible apps.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

You must then configure your reverse proxy to redirect or forward V4.2 requests to the V5.0 server, so that requests from devices running V4.2-compatible apps reach the V5.0 server.

Migrating your apps from V4.2 to V5.0 includes updating the native IBM Worklight libraries that are packaged with your applications. If you distribute your iOS applications via the Apple app store, you must certify the new V5.0-compatible version of your apps before you can make them available for public use. To do so, you must make a V5.0 production server accessible to the Internet so that your apps can be run and certified. This V5.0 server must be exposed through a different public URL than the V4.2 servers, to avoid confusion between the two.

To upgrade a production cluster, complete the following procedure:

Procedure

1. Install 5.0 server cluster.
2. Duplicate the 4.2 database and migrate it to 5.0. Under `<`*`Worklight Installation Directory`*`>/WorklightServer/databases`, run `upgrade⌂Worklight-mysql.sql` or `upgrade⌂Worklight⌂oracle.sql`, according to the type of database you are using.
3. Migrate the reports database. Under `<`*`Worklight Installation Directory`*`>/WorklightServer/databases`, run `upgrade⌂Worklightreports-mysql.sql` or `upgrade⌂Worklightreports-oracle.sql`, according to the type of database you are using.
4. Install the production customization.
5. Deploy old-apps so that you can remote disable them.
6. Deploy new-apps and new-adapters.

   At this point you are having two server clusters: One of 4.2 and another one of 5.0.
7. If you chose Option 1 above for migrating users from V4.2-based to V5.0-based apps, optionally remote disable the 4.2 apps in the 4.2 server, directing users to the new 5.0-compatible apps.
8. If you chose Option 2 above for migrating users from V4.2-based to V5.0-based apps, remote disable the old-apps in the 5.0 server and configure the reverse proxy to forward or redirect requests to 4.2 server to the 5.0 server. The upgrade procedure is now complete.

Firewalls

Firewalls can be configured at various layers of the IBM Worklight architecture.

Firewalls in front of an IBM Worklight Server use the typical configuration.

Special attention must be given to a firewall layer between the IBM Worklight servers and the IBM Worklight database.

- IBM Worklight Server employs database connection pooling. Firewalls may detect idle database connections and terminate them resulting in unexpected behavior.
- Firewalls limit the number of connections allowed. This is done to prevent Denial of Service (DoS) attacks. However, with multiple clustered IBM Worklight servers, the number of connections might be higher than usual.

Experimental IBM Worklight user documentation. This document is provided "as is". In case of issues refer to the online user documentation.

Disaster Recovery Site

IBM Worklight supports the creation of a separate disaster recovery site that becomes operational if the original site goes down.

A disaster recovery site is a second, physically separate IT center on which a copy of the IT systems exists and springs into operation if the original site is down. IBM Worklight has such a site for some of its customers.

Within the site, IBM Worklight provides redundancy at every level: compensating load balancers, multiple IBM Worklight servers that scale linearly, and database redundancy through Oracle RAC. Some customers prefer to provide another level of redundancy by using a disaster recovery site.

The key administrative factors for such a site are:
- Architecture
- Data mirroring from master to backup site
- Switching to back up site on disaster

Architecture

The architecture of the backup site is a copy of the original site. Special care must be taken to:
- Provide access to all corporate back-end systems.
- Create a switch that transfers incoming requests from master to backup site.

IBM Worklight relies on one single database, so an active-active configuration of master and back-up sites is not encouraged (unless you have the required bandwidth to perform database WAN replication).

Data mirroring

For the backup site to work, data on the master site must be mirrored to the backup on a regular basis:

Table 102. Data mirroring

Component	Description	Mirror frequency
IBM Worklight Database	All tables must be mirrored except for report tables and cache tables, which are relatively small in size.	Highly dependent on implementation and can range from few minutes to 24 hours. For further details contact software support.
IBM Worklight Software, customization, and content	Any change in IBM Worklight software, customization, or content must also be installed on the mirror servers.	As it occurs.

Switching to backup site

When you switch to the backup site, some information might be lost:
- All clients lose context and disconnect. In the case of an authenticated app, the user is prompted to log in again.
- Report information is lost (unless previously mirrored).
- Cache is lost. If Cache was implemented for various queries, an additional server fetch is required to fill cache.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

Switching back to Master Site

Before you switch back to the master site, you must mirror the database back to the master site.

Important: The success of a recovery site is in the details. To ensure the successful functioning of such a site, you must develop and follow a strict written procedure, which you test on a regular basis.

Push Notification

The IBM Worklight unified push notification mechanism enables sending mobile notifications to mobile phones.

Notifications are sent through the vendor infrastructure. For example, iPhone notifications are sent from the IBM Worklight Server to specialized Apple servers and from there to the relevant phones.

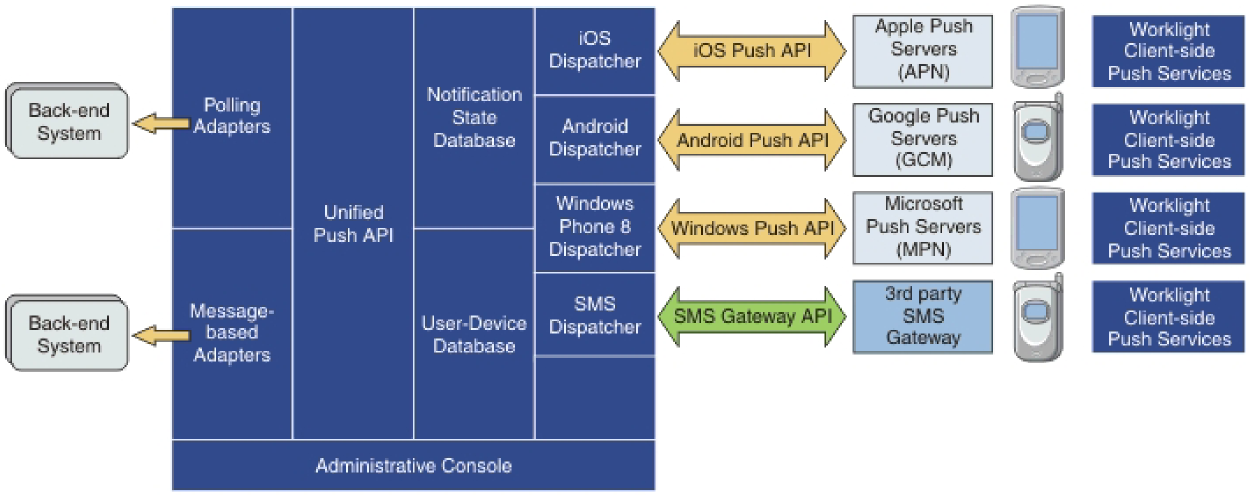


Figure 64. Push notification mechanism

Note: Push notification currently works for iOS, Android, and Windows Phone 8 only. SMS push notifications are supported on iOS, Android, Windows Phone 7.5, Windows Phone 8, and Blackberry devices that support SMS functions.

Proxy settings

Use the proxy settings to set the optional proxy through which notifications are sent to APNS and GCM. You can set the proxy by setting the properties **`push.apns.proxy.*`** and **`push.GCM.proxy.*`** in `worklight.properties`.

For further information about these and other settings, see "*Push Notification Settings*" in: “IBM Worklight properties” on page 406.

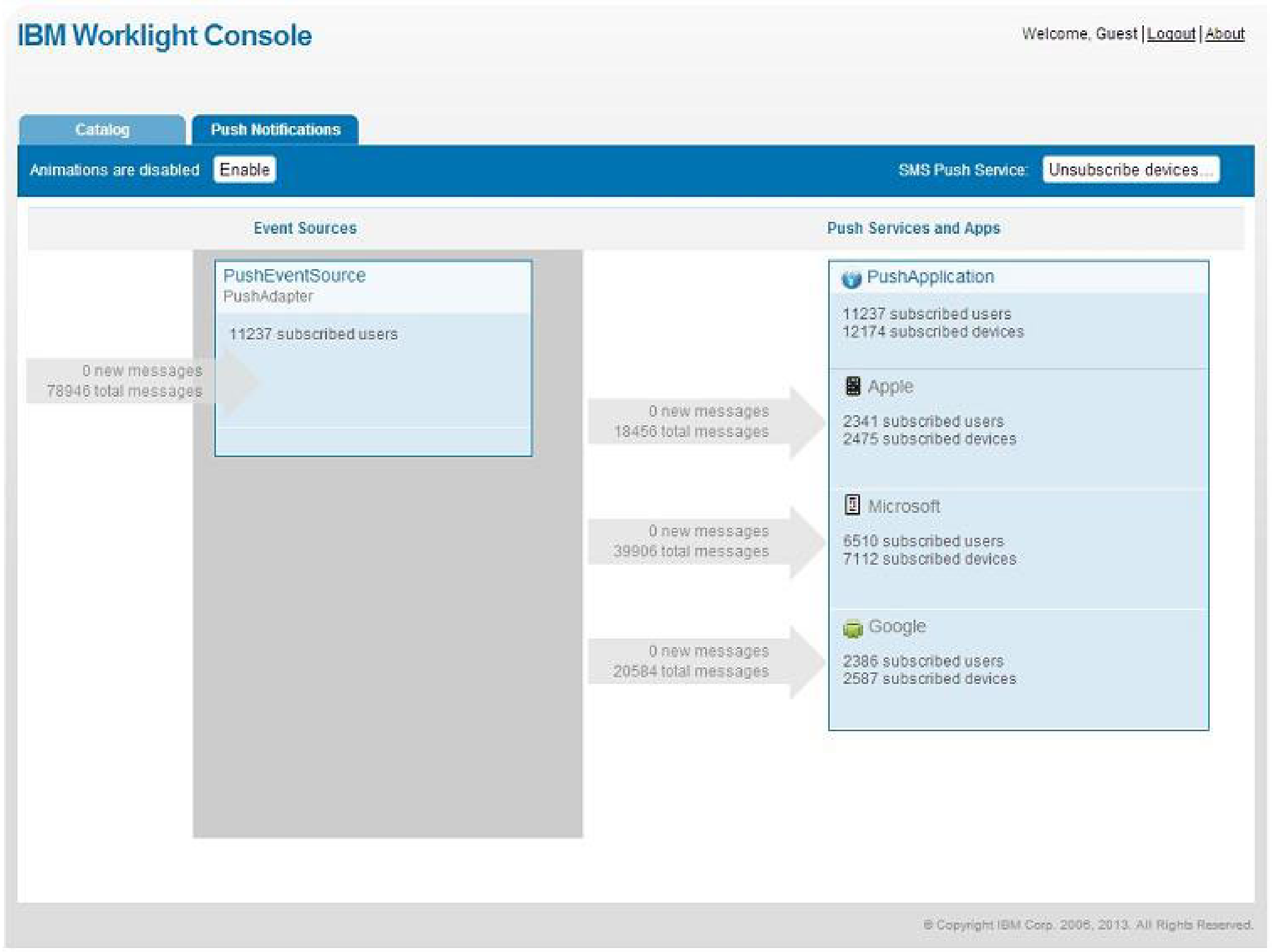
Architecture

Unlike other IBM Worklight servers, the push server requires outbound connections to Apple, Google, and Microsoft servers using ports defined by these companies.

When running a cluster of application servers, only one node will actually send push messages to Apple, Google, and Microsoft servers. This server is selected randomly.

The push notification console

The Push Notifications tab in the IBM Worklight Console provides you with a quick view of the various entities in the push notification chain.



11237 subscribed users

0 new messages
78946 total messages

11237 subscribed users
12174 subscribed devices

0 new messages
18456 total messages

2341 subscribed users
2475 subscribed devices

0 new messages
39906 total messages

6510 subscribed users
7112 subscribed devices

0 new messages
20584 total messages

2386 subscribed users
2587 subscribed devices

Figure 65. Push notifications in IBM Worklight Console

The left column displays the list of data sources that are configured in your IBM Worklight Server, including the number of users that are subscribed to notifications from each source.

The right column displays deployed applications, which can receive push notifications. For each application, the push notification services available for this application are also displayed. The console displays the number of notifications that are retrieved by an event source and sent to each application since system startup. It also displays errors that are related to connectivity to the push notification services.

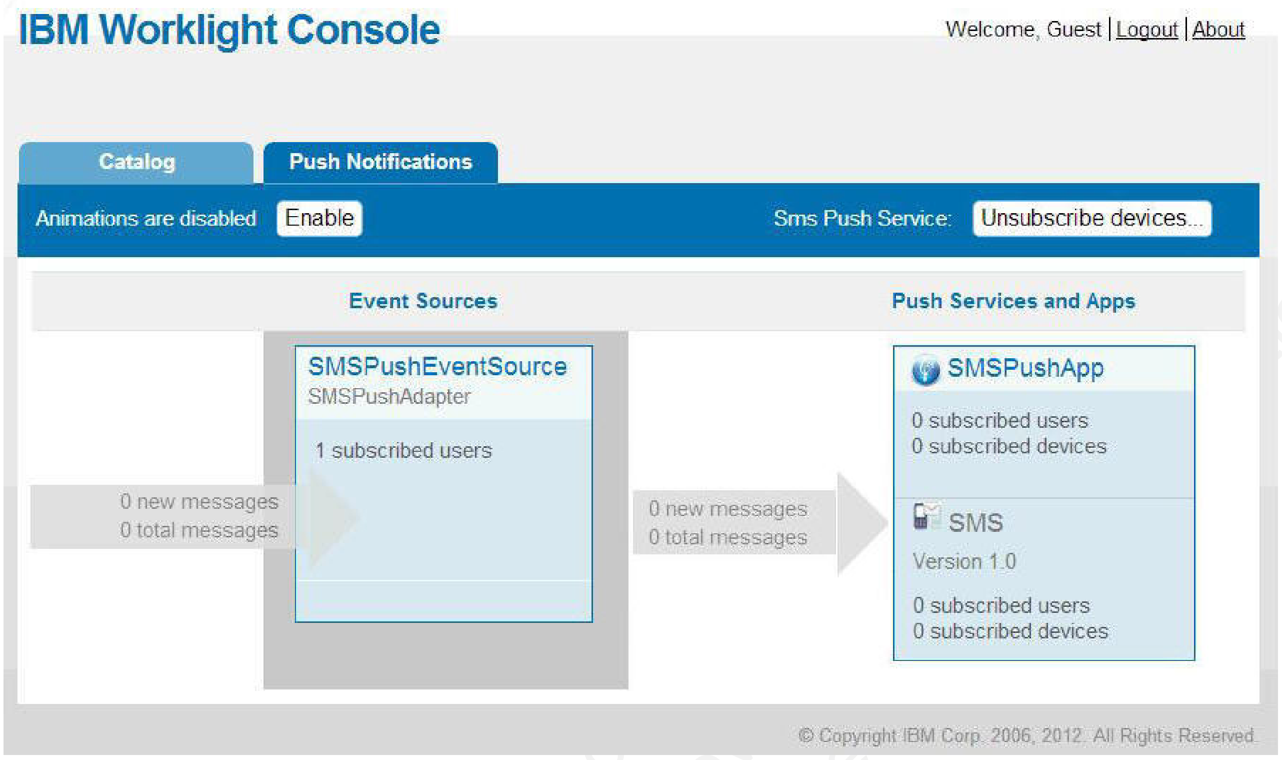
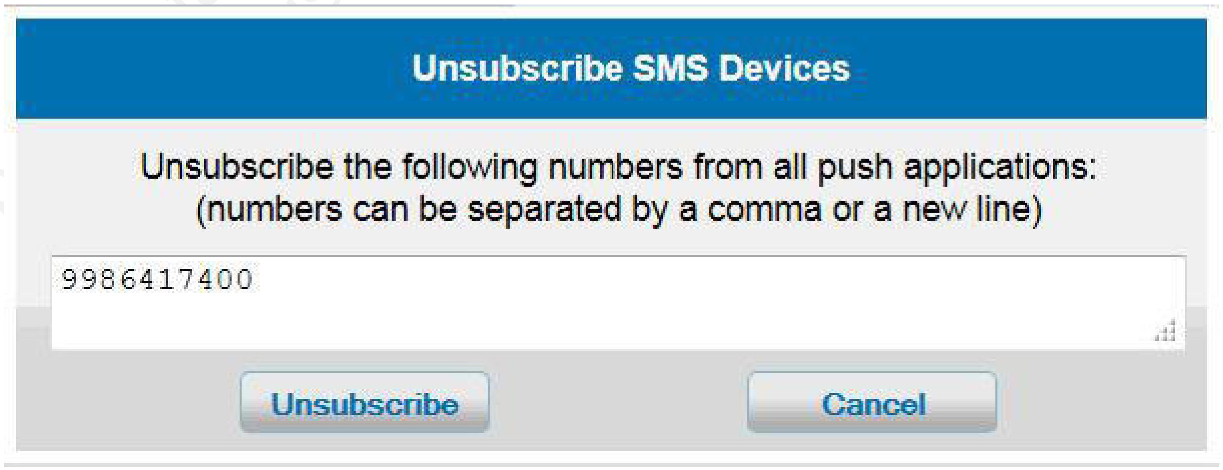


Figure 66. SMS push notifications in IBM Worklight Console

Administrators can forcibly unsubscribe existing SMS subscriptions by clicking **Unsubscribe devices**. The Unsubscribe SMS Devices window opens, and the administrator enters the mobile phone numbers to be unsubscribed.

Note: It is possible to have two subscriptions for the same phone number and user name; one created by using the device and one created by using the subscribe SMS servlet. If there are two subscriptions for the same phone number and user name, unsubscription by using the IBM Worklight Console unsubscribes both subscriptions.


Unsubscribe SMS Devices

Unsubscribe the following numbers from all push applications:
(numbers can be separated by a comma or a new line)

```
9986417400
```

Figure 67. Unsubscribe existing SMS subscriptions

Subscribe SMS servlet

Subscription, and unsubscription, to SMS notifications can be performed by making HTTP GET requests to the subscribe SMS servlet. The subscribe SMS servlet can be used for SMS subscriptions without the requirement for a user to have an app installed on their device.

Enter the following URL to access the subscribe SMS servlet:

```
http://<hostname>:<port>/<context>/subscribeSMS
```

This URL can be used to subscribe and unsubscribe.

You must create an application and an event source within an adapter and deploy them on the IBM Worklight Server before you make calls to the subscribe SMS servlet. See "Method WL.Server.createEventSource" on page 286 for information about creating an event source.

Table 103. Subscribe SMS servlet URL parameters

URL parameter	URL parameter description
option	Optional string. Subscribe or unsubscribe action to perform. The default option is `subscribe`. If any non-blank string other than `subscribe` is supplied, the unsubscribe action is performed.
eventSource	Mandatory string. The name of the event source. The event source name is in the format *AdapterName.EventSourceName*.
alias	Optional string. A short ID defining the event source during subscription. This ID is the same ID as provided in WL.Client.Push.subscribeSMS.
phoneNumber	Mandatory string. User phone number to which SMS notifications are sent. The phone number can contain digits (0-9), plus sign (+), minus sign (-), and space (␣) characters only.
userName	Optional string. Name of the user. If no user name is provided during subscription, an anonymous subscription is created by using the phone number as the user name. If a user name is provided during subscription, it must also be provided during unsubscription.
appId	Mandatory string for subscribe. The ID of the application that contains the SMS gateway definition. The application ID is constructed from the application name, application environment, and application version. For example, version 1.0 of Android application SMSPushApp has **appId** = SMSPushApp-android-1.0.

Note: If any parameter value contains special characters, this parameter must be encoded by using URL encoding, also known as percent encoding, before the URL is constructed. Parameter values containing only the following characters do not need to be encoded:

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online documentation.

a-z, A-Z, 0-9, period (.), plus sign (+), minus sign (-), and underscore (_)

Subscriptions that are created by using the subscribe SMS servlet are independent of subscriptions that are created by using a device. For example, it is possible to have two subscriptions for the same phone number and user name; one created by using the device and one created by using the subscribe SMS servlet. If there are two subscriptions for the same phone number and user name, unsubscription by using the subscribe SMS servlet unsubscribes only the subscription that is made through the subscribe SMS servlet. However, unsubscription by using the IBM Worklight Console unsubscribes both subscriptions.

Security

It is important to secure the subscribe SMS servlet because it is possible for entities with malicious intent to call the servlet and create spurious subscriptions. By default, IBM Worklight protects static resources such as the subscribe SMS servlet. The `authenticationConfig.xml` file is configured to reject all requests to the subscribe SMS servlet with a rejecting login module. To allow restricted access to the subscribe SMS servlet, IBM Worklight administrators must modify the `authenticationConfig.xml` file with appropriate authenticator and login modules.

For example, the following configuration in the `authenticationConfig.xml` file ensures only requests with a specific user name in the header of the HTTP request are allowed:

```
<staticResources>
 <resource id="subscribeServlet" securityTest="SubscribeServlet">
  <urlPatterns>/subscribeSMS*</urlPatterns>
 </resource>
 ...
</staticResources>

<securityTests>
 <customSecurityTest name="SubscribeServlet">
  <test realm="SubscribeServlet" isInternalUserID="true"/>
 </customSecurityTest>
 ...
</securityTests>

<realms>
 <realm name="SubscribeServlet" loginModule="headerLogin">
  <className>com.worklight.core.auth.ext.HeaderAuthenticator</className>
 </realm>
 ...
</realms>

<loginModules>
 <loginModule name="headerLogin">
  <className>com.worklight.core.auth.ext.HeaderLoginModule</className>
  <parameter name="user-name-header" value="username"/>
 </loginModule>
 ...
</loginModules>
```

Backup and recovery

You can back up the customization and the content (adapters and applications) outside the IBM Worklight instance, for example in a source control system.

Experimental IBM Worklight offline user documentation. This document is provided as-is. In case of issues, refer to the online user documentation.

It is advisable to back up the IBM Worklight database as-is. When Reports are enabled the database can be quite large. Consider the benefits of backing them up separately. Report tables can be configured to store on a different database instance.

Logging and monitoring mechanisms

IBM Worklight reports errors, warnings, and informational messages into a log file. The underlying logging mechanism varies by application server.

Worklight Server

For Worklight Server, logging models vary according to the server platform. For more information, including the location of the log files, see the documentation for the relevant platform, as shown in the following table.

Table 104. Documentation for different server platforms

Server platform	Location of documentation
Apache Tomcat	http://tomcat.apache.org/tomcat-7.0-doc/logging.html#Using_java.util.logging_(default)
WebSphere Application Server Version 7.0	http://pic.dhe.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.base.doc/info/aes/ae/ttrb_trcover.html
WebSphere Application Server Version 8.0	http://pic.dhe.ibm.com/infocenter/wasinfo/v8r0/topic/com.ibm.websphere.base.doc/info/aes/ae/ttrb_trcover.html
WebSphere Application Server Version 8.5 Full Profile	http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.multiplatform.doc/ae/ttrb_trcover.html
WebSphere Application Server Version 8.5 Liberty Profile	http://publib.boulder.ibm.com/infocenter/radhelp/v8r5/topic/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/rwlp_logging.html

Worklight Studio

Worklight Studio uses the standard JDK logging model. To use logging in Worklight Studio, update a file called `development.logging.properties`. The `development.logging.properties` file is created in the `WorklightServerHome/conf` directory of your workspace when you create a project. This file contains default logging parameters, held as name-value pairs in the standard JDK format. To use logging in Worklight Studio, update these parameters. When a project is activated, any properties that are set in the `development.logging.properties` file are used to override the default logging properties.

When a project is active, log files are written to the `WorklightServerHome/project/logs` directory, where *project* is the name of your project. If no projects are active, the log files are written to the `WorklightServerHome/logs` directory. Two log files are written: `server.log` and `audit.log`. The `server.log` file includes all server messages and the `audit.log` file includes all adapter procedure calls, if **`audit`**`=true` is set in the adapter definition.

Every message that is written to the `server.log` and `audit.log` files is also written to the Studio console. Studio messages are also written to the Studio console. In the case of an exception in the Studio, the exception reason is written to the studio console and a full stack trace to an external `.log` file in the *`eclipse_workspace`*`\.metadata\.log` directory, where *eclipse_workspace* is the Eclipse workspace folder.

Experimental IBM Worklight user documentation
This document is provided "as is".
In case of issues refer to the official documentation.

To check for errors, look first at the logs for the project and if none are found then look at the logs in *eclipse_workspace*`\WorklightServerHome\logs`. Additionally, look in the `.log` file for exceptions that come from the studio.

Log monitoring tools

For Apache Tomcat, you can use industry standard log file monitoring tools such as Splunk to monitor logs and highlight errors and warnings.

For WebSphere Application Server, use the log viewing facilities that are described in the information centers that are listed in the table in the Worklight Server section.

Back-end connectivity

To enable trace to monitor back-end connectivity, see the documentation for your server platform in the table in the Worklight Server section. The packages to be enabled for trace are `com.worklight.adapters` and `com.worklight.integration`. Set the log level to `FINEST` for each package.

Audit logs

To write audit log information, use the standard facilities of java.util.logging. Use the `audit` category.

Login and authentication issues

To diagnose login and authentication issues, enable the package `com.worklight.auth` for trace and set the log level to `FINEST`.

Vitality queries

Use IBM Worklight vitality queries to determine your server vitality.

You generally use the IBM Worklight vitality queries from a load balancer or from a monitoring app (for example, Patrol).

You can run vitality queries for the server as a whole, for a specific adapter, for a specific app, or for a combination of. The following table shows some examples of vitality queries.

Table 105. Examples of queries that help determine server vitality

Query	Purpose
`http://<server>:<port>/<publicWorkLightContext>/ws/rest/vitality`	Checks the server as a whole.
`http://<server>:<port>/<publicWorkLightContext>/ws/rest/vitality?app=MyApp`	Checks the server and the `MyApp` application.
`http://<server>:<port>/<publicWorkLightContext>/ws/rest/vitality?app=MyApp&adapter=MyAdapter`	Checks the server, the `MyApp` application, and the `MyAdapter` adapter.

Note: Do not include the `/<publicWorkLightContext>` part of the URL if you use IBM Worklight Developer Edition. You must add this part of the URL only if

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Worklight Server is running on another application server, such as Apache Tomcat or WebSphere Application Server (full profile or Liberty profile).

Vitality queries return an XML content that contains a series of <ALERT> tags, one for each test.

Example query and response

By running the `http://<server>:<port>/ws/rest/vitality?app=MyApp` query, you might have the following successful response, with an <ALERT> tag for each of the two tests:

```
<ROOT>
  <ALERT>
    <DATE> 2011-05-17T15:31:35.583+0300 </DATE>
    <EVENTID>0</EVENTID>
    <SUBJECT>SRV</SUBJECT>
    <TYPE>I</TYPE>
    <COMPUTER>worklight.acme.com</COMPUTER>
    <DESCRIPTION>Server is running</DESCRIPTION>
  </ALERT>
  <ALERT>
    <DATE> 2011-05-17T15:31:35.640+0300 </DATE>
    <EVENTID>0</EVENTID>
    <SUBJECT>APPL</SUBJECT>
    <TYPE>I</TYPE>
    <COMPUTER>worklight.acme.com</COMPUTER>
    <DESCRIPTION>Application 'MyApp' is deployed</DESCRIPTION>
  </ALERT>
</ROOT>
```

Return values

The following table lists the attributes that might be returned, and their possible values.

Table 106. Return values and values

Return attribute	Possible values
DATE	Date value in JavaScript™ format
EVENTID	0 for the running server, deployed adapter, or deployed application 1 for not deployed adapter 2 for not deployed application 3 for malfunctioning server
SUBJECT	SRV for Worklight Server ADPT for adapter APPL for application
TYPE	I – valid E – error
COMPUTER	Reporting computer name
DESCRIPTION	Status description in plain text

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

The returning XML contains more attributes, which are undocumented constants that you must not use.

Routing logging to Windows event log

If you are using Apache Tomcat on Windows, you can change the log output destination by configuring Windows Event Viewer.

Procedure

1. If it does not exist, create the file *TOMCAT_DIRECTORY*/bin/setenv.bat, where *TOMCAT_DIRECTORY* is the directory in which Tomcat is installed.
2. Edit the setenv.bat file to include the following line: CLASSPATH=*WORKLIGHT_DIR*/WorklightServer/logging/worklight-windows-event-viewer-logging.jar, where *WORKLIGHT_DIR* is the directory in which you installed Worklight.
3. Modify *TOMCAT_DIRECTORY*/conf/logging.properties to set up Windows Event Viewer, by including the following lines:

```
handlers = 1catalina.org.apache.juli.FileHandler, 2localhost.org.apache.juli.FileHandler,
3manager.org.apache.juli.FileHandler, 4host-manager.org.apache.juli.FileHandler,
java.util.logging.ConsoleHandler, com.worklight.core.logging.windows.WindowsEventViewerHandler

.handlers = 1catalina.org.apache.juli.FileHandler, java.util.logging.ConsoleHandler

com.worklight.handlers = com.worklight.core.logging.windows.WindowsEventViewerHandler
com.worklight.core.logging.windows.WindowsEventViewerHandler.level = ALL
com.worklight.core.logging.windows.WindowsEventViewerHandler.formatter = java.util.logging.Sim
```

4. Modify the value of your system variable **PATH** to include the path to the Worklight .dll files. You can find the .dll files in the following folders, depending on your system

   WORKLIGHT_DIR/WorklightServer/logging/bin.windows-x86/IBMEventLog.dll

   WORKLIGHT_DIR/WorklightServer/logging/bin.windows-x86/EventLogCategories.dll

   WORKLIGHT_DIR/WorklightServer/logging/bin.windows-x64/IBMEventLog.dll

   WORKLIGHT_DIR/WorklightServer/logging/bin.windows-x64/EventLogCategories.dll
5. Create the registry key EventMessageFile in the registry folder HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\eventlog\Application\IBMEventLog. The value of the key is the full path to EventLogCategories.dll.

Results

Events are logged in the Windows Event Viewer.

Enabling trace for adapters in an Eclipse-hosted server

To enable trace for adapters that are hosted in the Worklight Server embedded in Eclipse, you must modify the development.logging.properties file.

Procedure

1. In the WorklightServerHome/conf directory of your workspace, open the development.logging.properties file.
2. Uncomment the following lines:

```
. com.worklight.level = FINE
. java.util.logging.FileHandler.level = FINE
```

Experimental IBM Worklight documentation
This document is provided as is.
In case of issues, refer to the online user documentation.

Results

When you use logger methods, the messages are then sent to the Worklight console view in Eclipse.

Note: A large amount of other information might also be sent. To clearly identify your debug messages, mark them with identifiers.

Optimizing and tuning of IBM Worklight Server

Memory allocation is performed on the Java instance of the application server.

For best results, you should install IBM Worklight Servers on a 64-bit server.

You must set the required memory size of the application server:
- `Liberty`: set **`JAVA_ARGS`** in <*`install_dir`*>`/Worklight/server/wlp/bin/securityUtility`.
- `WebSphere Application Server`: Log in to the admin console. Go to **Servers** > **Server types** > **WebSphere application servers**: choose each server and set Java memory settings under **Java Process definition** > **JVM arguments**.
- `Apache Tomcat`: find the `catalina` script and set **`JAVA_OPTS`** to inject memory.

For information about how to calculate memory size, see the following documents:
- Scalability and Hardware Sizing (PDF)
- Hardware Calculator (XLS)

Troubleshooting Worklight Server

You can troubleshoot to locate the server and databases on Windows 8, Windows 7, and Windows XP, or to find the cause of installation or database creation failure.

Troubleshooting to locate the server and databases on Windows

You can troubleshoot to locate the server and databases on Windows 8, Windows 7, and Windows XP.

About this task

If you cannot locate the directories even though the documentation states that the server and databases are located in the `C:\ProgramData\IBM\Worklight` directory for Windows 8 and Windows 7 and the `C:\Documents and Settings\All Users\Application Data\IBM\Worklight` directory for Windows XP, complete the following procedure to troubleshoot this problem:

Procedure

1. Click **Tools** > **Folder Options** dialog.
2. Select the **View** panel and enable the visibility of hidden files and folders. The `ProgramData` folder in Windows 8 and Windows 7 and the `Application Data` folder in Windows XP are hidden folders and visibility must be enabled to see them.

Troubleshooting to find the cause of installation failure

You can troubleshoot to find the cause of installation failure.

Experimental IBM Worklight offline user documentation. This document is provided as is. In case of issues, refer to the online user documentation.

About this task

If installation failed but the cause is not obvious, you can troubleshoot by completing the following procedure:

Procedure

See the `install.log` file in the installation directory. On Windows systems, if the default installation location was chosen, the directory is `C:\Program Files\IBM\Worklight\`. This file contains details about the installation process.

What to do next

If you still cannot determine the cause of installation failure, you can use the manual installation instructions to continue making progress. See "Manually configuring the application server" on page 342.

Failed to create the DB2 database

An incompatible database connection mode might result in failure to create the DB2 database.

About this task

Use this procedure if the following message is displayed when you attempt to create a DB2 database:

```
"Creating database <WL_DB> (this may take 5 minutes) ... failed: Cannot
connect to database <WL_DB> after it was created:
com.ibm.db2.jcc.am.SqlException: DB2 SQL Error: SQLCODE=-1035,
SQLSTATE=57019, SQLERRMC=null, DRIVER=<driver_version>"
```

Procedure

1. Wait a few minutes for the current DB2 database connections to close, and then click **Back** followed by **Next** to check if the issue is solved.
2. If the problem persists, contact your database administrator to solve the database connection issue that is documented on the following web page: http://pic.dhe.ibm.com/infocenter/db2luw/v10r1/index.jsp?topic=%2Fcom.ibm.db2.luw.messages.sql.doc%2Fdoc%2Fmsql01035n.html.

Installation blocked by DB2 connection errors

An incorrect DB2 JDBC driver can prevent connection to the database, and block the IBM Worklight Server installation.

About this task

During installation, the IBM Worklight Server installer attempts to ensure that the specified databases exist. If the database is present but attempting to access it produces an error, the Worklight Server installer blocks the **Next** button, and prevents the user from moving forward to complete the installation, displaying a message similar to the following:

```
Cannot access database: WRKLGHT. Details: com.ibm.db2.jcc.am.io:
[jcc][t4][2057][11264][4.7.85] The application server rejected
```

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues refer to the online user documentation.

```
establishment of the connection. An attempt was made to access a database,
which was either not found or does not accept transactions.
ERRORCODE=-4499, SQLSTATE=08004
```

This error can be caused by an incorrect or outdated DB2 JDBC driver.

Procedure

1. If you receive this error, verify the current DB2 JBDC driver.
2. Newer fix packs of the DB2 JDBC driver may solve the issue. These fix pack drivers are available from DB2 JDBC Driver Versions.

IBM Worklight properties

You can check and change the settings configured by the IBM Worklight Installer during installation in the `worklight.properties` file.

IBM Worklight contains many configuration options that are important for the production environment. These options are stored in the customization `WAR` file under `worklight.properties`, which is located in the `<`*`Worklight Root Directory`*`>\conf` directory.

Configuring the IBM Worklight Server location

You can change the settings that are normally configured by the IBM Worklight Installer during installation if necessary.

To configure the IBM Worklight Server location you must set the values of the following properties in the `worklight.properties` file:

Properties for configuring the IBM Worklight Server location found in the `worklight.properties` file

Property Key	Property Value
`publicWorklightHostname`	The IP address or host name of the computer running IBM Worklight. If the IBM Worklight Server is behind a reverse proxy, the value is the IP address or host name of the reverse proxy. This property must be identical for nodes within the same cluster. Default: IP address of current server.
`publicWorklightPort`	The port for accessing the IBM Worklight Server. If the IBM Worklight Server is behind a reverse proxy, the value is the port for accessing the reverse proxy. This property must be identical for nodes within the same cluster. Default: Same as **`local.httpPort.`**

Experimental IBM Worklight online user documentation
This document is provided "as-is".
In case of issues, refer to the online user documentation.

Properties for configuring the IBM Worklight Server location found in the `worklight.properties` file

Property Key	Property Value
publicWorklightProtocol	The protocol for accessing the IBM Worklight Server. The valid values are `HTTP` and `HTTPS`. If the IBM Worklight Server is behind a reverse proxy, the value is the protocol for accessing the reverse proxy. This property must be identical for nodes within the same cluster. Default: **`HTTP`**.

IBM Worklight database setup

IBM Worklight uses defaults to access the IBM Worklight database. You can use a set of property keys to change the settings.

JDBC-based properties

The `worklight.properties` file might contain one of two methods to connect to a database: old fashioned `JDBC` url or the newer `JNDI` resource name method. You must not mix the two methods in one file.

Property keys and values for JDBC-based properties

Property Key	Property Value
wl.db.username	IBM Worklight database user name. Default: `Worklight`
wl.db.type	The database vendor. Options include: DB2, MYSQL, ORACLE, HSQL, DERBY Default: HSQL
wl.db.password	IBM Worklight database password. Default: `Worklight`
wl.db.driver	The class that implements a JDBC driver for each vendor. For example: MySQL: `com.mysql.jdbc.Driver` Oracle: `oracle.jdbc.OracleDriver` DB2: `com.ibm.db2.jcc.DB2Driver` Derby: `org.apache.derby.jdbc.EmbeddedDriver`
wl.db.reports-url(*)	JDBC path to IBM Worklight Reports database Default: refers to IBM Worklight database
wl.db.reports-username(*)	IBM Worklight Reports database user name. Default: refers to IBM Worklight database

Experimental IBM Worklight online user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

Property keys and values for JDBC-based properties

Property Key	Property Value
wl.db.reports-password(*)	IBM Worklight Reports database password Default: refers to IBM Worklight database
wl.reports.db.type(*)	The database vendor. Options include: DB2, MYSQL, ORACLE, HSQL, DERBY Default: HSQL

Note: (*) By default all IBM Worklight report mechanisms use a single reports database. The reports database is set to be the same database as the IBM Worklight database. This default can be changed by setting the parameters in the following table. Enabling Raw Data Reports

JNDI-based properties

Property keys and values for JNDI-based properties

Property Key	Property Value
wl.db.jndi.name	JNDI name for the Worklight database
wl.reports.db.jndi.name(*)	JNDI name for the Worklight reports database
wl.db.type	The database vendor. Options include: DB2, MYSQL, ORACLE, HSQL, DERBY Default: HSQL
wl.reports.db.type (*)	The database vendor. Options include: DB2, MYSQL, ORACLE, HSQL, DERBY Default: HSQL **Note:** Must be identical to `wl.db.type value`

Note: (*) By default all IBM Worklight reports mechanisms use a single reports database. The reports database is set to be the same database as the IBM Worklight database, but it is possible to change this default.

Protecting IBM Worklight Console

You can protect IBM Worklight Console by defining user credentials required to access it.

The user credential settings that you define to protect the IBM Worklight Console can be encrypted as described in “Storing properties in encrypted format” on page 411.

Property Key	Property Value
console.username	Name of the user that can access the Console
console.password	User password

Experimental IBM Worklight offline user documentation.
This document is provided "as is".
In case of issues, refer to the online user documentation.

In addition to defining these two properties, configure the `authenticationConfig.xml` file, located under `<*Worklight Root Directory*>\server\conf`. This file is described in "The authentication configuration file" on page 140

Push notification settings

Use the following GCM and APNS proxy settings for Android and iOS push notification respectively.

Push notification settings

GCM proxy settings	Value
`push.gcm.proxy.enabled=false`	Shows whether Google GCM must be accessed through a proxy. Default is **`false`**.
`push.gcm.proxy.protocol=`	Can be either **`http`** or **`https`**.
`push.gcm.proxy.host=`	GCM proxy host. Negative value means default port.
`push.gcm.proxy.port=-1`	GCM proxy port. Use **`-1`** for the default port.
`push.gcm.proxy.user=`	Proxy user name, if the proxy requires authentication. Empty user name means no authentication.
`push.gcm.proxy.password=`	Proxy password, if the proxy requires authentication.

APNS proxy settings	Value
`push.apns.proxy.enabled`	Shows whether APNS must be accessed through a proxy. Default is **`false`**
`push.apns.proxy.type`	Must be **`SOCKS`**
`push.apns.proxy.host`	APNS proxy host
`push.apns.proxy.port`	APNS proxy port

SSL certificate keystore setup

Mobile applications often connect to multiple back-end systems. Some back-end systems require access through an HTTP adapter, and each back-end system can require a different SSL certificate for secure communication using HTTPS. These SSL certificates are stored in a keystore that is configured to the IBM Worklight Server by using property keys.

IBM Worklight provides a default keystore. You can choose to use this default keystore or replace it with your own keystore.

To configure an SSL certificate keystore, you must set the values of the following property keys in the `worklight.properties` file:

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Properties for configuring an SSL certificate keystore found in the `worklight.properties` file

Property Key	Property Value
`ssl.keystore.path`	SSL certificate keystore location. Default: `conf/default.keystore`. **`ssl.keystore.path`** can be configured in 2 ways: • By using the relative path, for example, **`ssl.keystore.path`**`=conf/default.keystore`, where the root directory is `<`*`Worklight Root Directory`*`>\server\`. • By using the absolute path, for example, **`ssl.keystore.path`**`=/opt/wl/default.keystore` for UNIX, or `C:\\wl\\default.keystore` for Windows.
`ssl.keystore.type`	SSL certificate keystore type. Valid keystore types: • `jks` • `PKCS12` Default: `jks`.
`ssl.keystore.password`	SSL certificate keystore password. Default: `worklight`.

For example:

```
############################
# SSL-Security
############################
ssl.keystore.path=/opt/worklight/sslcertificates.keystore
ssl.keystore.type=PKCS12
ssl.keystore.password=worklight
```

In addition to defining these three properties, configure the HTTP adapter XML file, which is located under `<`*`Worklight Root Directory`*`>\adapters\<`*`HTTP adapter name`*`>`. This file is described in "The adapter XML File" on page 99.

If you use SSL with mutual authentication between the IBM Worklight Server and a back-end system, be aware of the following requirement:

- Define an alias and password for the private key of the keystore where the SSL certificate is stored. The alias and password are defined in the `<connectionPolicy>` element of the HTTP adapter XML file, *`adaptername`*`.xml`. The `<sslCertificateAlias>` and `<sslCertificatePassword>` subelements are described in "The <connectionPolicy> element of the HTTP adapter" on page 103.

  Note: The password that is specified in **`ssl.keystore.password`** is not the same password that is specified in `<sslCertificatePassword>`. **`ssl.keystore.password`** is used to access the keystore itself. `<sslCertificatePassword>` is used to access the correct SSL certificate within the keystore.

Experimental IBM Worklight offline user documentation.
This document is provided "as is".
In case of issues refer to the online user documentation.

Miscellaneous Settings

Configure the **serverSessionTimeout**, **bitly.username**, and **bitly.apikey** parameters.

Property keys and values for **serverSessionTimeout**, **bitly.username**, and **bitly.apikey** parameters.

Property Key	Property Value
serverSessionTimeout	Client inactivity timeout, after which the IBM Worklight session is invalidated. Default is 10 minutes.
bitly.username	User name for accessing the bit.ly API for creating a shortened URL for mobile web apps through IBM Worklight Console.
bitly.apikey	The bit.ly API Key.

Storing properties in encrypted format

You must encrypt the properties that are too sensitive to be written in clear text within the properties file.

Storing properties in encrypted format

You can keep properties contained in `worklight.properties` either in open or in encrypted form.

An encrypted property is determined by a suffix `.enc` on its name, for example:

```
console.password.enc=TYakEHRba3rIU7pNjxtDxoAdqijKIEt7cy4mCr0iaEj0rY08ODK00yqR
```

The IBM Worklight configuration is accessed for a property. If the property is not found, but the same encrypted property (with `.enc` suffix) is defined, IBM Worklight automatically decrypts the value and returns it to the caller.

Storing the master key

All of the encrypted values use the same secret key, which is stored in the special variable called **worklight_enc_password**. This variable is defined as an operating system environment variable:

- On Windows systems: Set an environment variable under the user running the IBM Worklight Server. When running under a Windows NT service, define the password as a service property by using the registry editor. For more information, see the Microsoft support website.
- On Linux systems: Set the environment variable.

Encryption

To encrypt IBM Worklight properties on Windows systems, use the `encrypt.bat` utility under < *worklight_install_dir*>`/WorklightServer`.

This utility accepts a file that contains the properties to be encrypted and the encryption password. The utility outputs the encrypted values to the same file (so that sensitive data is deleted).

On Linux systems, use the `encrypt.sh` utility.

Experimental IBM Worklight offline user documentation
This document is provided as is.
In case of issue refer to the online user documentation.

The input file for the encryption is called `secret.properties` and contains the following data:

```
worklight_enc_password=abc123
certificate.password.enc=certificatepwd123
wl.db.password.enc=edf545
```

After running the `encrypt.sh` tool, the file `secret.properties` contains the following data:

```
#Copy the contents of this file to the worklight.properties file.
#Keep the password value in the secure system property worklight_enc_password.
#Wed Nov 28 10:10:44 CST 2012
certificate.password.enc=dR4lnMQDaNEQyLQl7b2RmpdE99HKpqaSJ6mce0uJgaY\=
wl.db.password.enc=6boxojGZsUNTXwOOGgI6dg\=\=
```

Obsolete properties

Some properties are no longer required.

Category	Properties
Proxy settings	**`proxy.enabled, proxy.nonProxyHosts, proxy.host, proxy.port, proxy.username, proxy.password, https.proxy.host, https.proxy.port`**
Public resource server settings	**`publicResourceServer.deployDestination, publicResourceServer.host, publicResourceServer.port, publicResourceServer.filesRootDir`**
Environments	**`environment.igoogle, environment.netvibes, environment.iphone, environment.vista, environment.dashboard, environment.embedded, environment.facebook, environment.air, environment.android, environment.blackberry`**
Certificate settings	**`certificate.certificatesDirPath, certificate.keyStoreFilePath, certificate.keyAlias, certificate.keyStorePassword, certificate.keyAliasPassword, certificate.PFXFilePath, certificate.password, certificate.DERFilePath, certificate.P7BFilePath, vista.linux.osslsigncodeFilepath`**
Push notification settings	**`push.apns.certificatePassword, push.gcm.senderID, push.gcm.senderPassword`**
Miscellaneous settings	**`devmode, guid, wlclientTimeout, backend.request.timeout, reports.produceReports,wl.db.initialSize,wl.db.maxActive,wl.d`**
Tomcat settings	**`local.bindAddress,local.httpPort`**
Console security settings	**`console.username, console.password`**

SMS gateway configuration

An SMS gateway, or SMS aggregator, is a third-party entity which is used to forward SMS notification messages to a destination mobile phone number. IBM Worklight routes the SMS notification messages through the SMS gateway.

Experimental IBM Worklight offline documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

To send SMS notifications from IBM Worklight, one or more SMS gateways must be configured in the `SMSConfig.xml` file, which is in the `/server/conf` folder of your project. To configure an SMS gateway, you must set the values of the following elements, subelements, and attributes in the `SMSConfig.xml` file. The Worklight Server must be restarted when any changes are made in the `SMSConfig.xml` file.

Table 107. `SMSConfig.xml` elements and subelements

Element	Element Value
`gateway`	Mandatory. The `<gateway>` element is the root element of the SMS gateway definition. It includes 6 attributes: • `hostname` • `id` • `port` • `programName` • `toParamName` • `textParamName` These attributes are described in Table 108
`parameter`	Optional. The `<parameter>` subelement is dependent on the SMS gateway. Each SMS gateway may have its own set of parameters. The number of `<parameter>` subelements is dependent on SMS gateway-specific parameters. If an SMS gateway requires the user name and password to be set, then these parameters can be defined as `<parameter>` subelements. Each `<parameter>` subelement has the following attributes: • `name` • `value`

Table 108. `<gateway>` element attributes

Attribute	Attribute Value
`hostname`	Mandatory. The host name of the configured SMS gateway.
`id`	Mandatory. A unique ID that identifies the SMS gateway. Application developers specify the ID in the application descriptor file, `application-descriptor.xml`, when they develop an application.
`port`	Optional. The port number of the SMS gateway. The default value is 80.
`programName`	Optional. The name of the program that the SMS gateway expects. For example, if the SMS gateway expects the following URI: `http://<hostname>:port/sendsms` then **`programName`**="sendsms"

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Table 108. `<gateway>` element attributes (continued)

Attribute	Attribute Value
toParamName	Optional. The name that is used by the SMS gateway to specify the destination mobile phone number. The default value is *to*. The destination mobile phone number is sent as a name-value pair when SMS notifications are sent; that is, *toParamName=destination mobile phone number*.
textParamName	Optional. The name that is used by the SMS gateway to specify the SMS message text. The default value is *text*.

If the SMS gateway expects an HTTP post in the following format to forward SMS messages to a mobile device:

http://myhost:13011/cgi-bin/sendsms?to=*destination mobile phone number*&text=*message text*&username=fcsuser&password=fcspass

The SMSConfig.xml file is configured as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<sms:config xmlns:sms="http://www.worklight.com/sms/config" xmlns:xsi="http://www.w3.org/2001/XMLSche

 <gateway hostname="myhost" id="kannelgw" port="13011" programName="cgi-bin/sendsms" toParamName="to
  <parameter name = "username" value = "fcsuser" />
  <parameter name = "password" value = "fcspass" />
 </gateway>

</sms:config>
```

Internal IBM Worklight Database Tables

You can access a database of common tables from the IBM Worklight Server. The database must not be written to, and it might change from one release to another.

The following table provides a list of common IBM Worklight database tables, their description, and how they are used.

Name	Description	Order of Magnitude
ADAPTER_SYNC_DATA	Replication across cluster nodes table for adapters.	10s of rows.
APP_ACTIVITY_REPORT	Statistics for application activity. Records a record per server accessing user.	Size depends on Application. Customer is responsible for purging older entries after aggregating to Data Warehouse
APP_SYNC_DATA	Replication across cluster nodes table for applications.	10s of rows.
APP_VERSION_ACCESS_DATA	Exceptions on App availability used for the **Deny App** feature	10s of rows.

Experimental IBM Worklight online user documentation. This document is provided "as-is". In case of issues, refer to the online user documentation.

Name	Description	Order of Magnitude
AUTH_ASSOCIATED_IDENTITY	Association between the user IDs in different realms. Out of the box associations are not defined. This feature may be enabled using code.	If configured, will contain a single row per user.
BACKEND_SUBSCRIPTIONS	Obsolete. Exists in pre v5.0 versions. Definitions of the cached data: procedures and parameter values	-
CLUSTER_SYNC	Cluster synchronization time	10s of rows
GADGET_APPLICATIONS	Environments (for example, iPhone, Android) of deployed applications. References the GADGETS table.	10s of rows
GADGET_USER_PREF	User preferences according to unique user identifier. Out of the box, there are no user preferences – preferences may be added by App developer.	1 row per preference, per user.
GADGETS	Deployed applications	10s of rows
NOTIFICATION_APPLICATION	Push notification table	10s of rows
NOTIFICATION_DEVICE	Push notification table. Stores a record per device, per user subscription. Many to one relationship with NOTIFICATION_USER table.	1 row per device subscribing to event source
NOTIFICATION_MEDIATOR	Push notification table	Less than 10 rows
NOTIFICATION_USER	Push notification table. Stores a record per user subscription to event sources	1 row per user subscribing to an event
PROPERTIES	N/A	10s of rows
USAGE_DATA	Usage report	1 row per active user per day.

The following table provides a list of common IBM Worklight WLREPORT database tables and their usage.

Name	Description	Order of Magnitude
ACTIVITIES_CUBE	A materialized table of the 4 dimensional data cube. Populated every night based on the last 30 days of data. Can be used by BIRT or other reporting tools.	Size depends on app and device usage, but is limited to last 30 days for faster access to the last 30 days of activities.

Experimental IBM Worklight online documentation
This document is provided "as-is".
In case of issues, refer to the online user documentation.

Name	Description	Order of Magnitude
`FACT_ACTIVITIES`	Summarization of activities used for device analytics. Updated by Worklight Server every 20 minutes with data from `APP_ACTIVITY_REPORT` table. Primarily used by BIRT reports and by other reporting tools.	Size depends on app/device usage.
`NOTIFICATION_ACTIVITIES`	Summarization of activities used for notification analytics. Updated with data from `NOTIFICATION_REPORT` table. Primarily used by BIRT reports and by other reporting tools.	Size depends on app/notification usage.
`PROC_REPORT`	Internal table used for housekeeping and maintaining the state of the scheduler tasks.	About 72 rows per day.

HTTP Interface of the production server

You can use the HTTP interface of the production server to make application API requests or web application resource requests. Use the following request structures, headers, and elements.

Application API requests

Use the following request structure to perform an application API request:

```
{Protocol}://{Worklight Server}/apps/services/api/{Application ID}/{Application Environment}/{Action
```

Application API request headers

Header Name	Data Type	Description	Valid values
`x-wl-app-version`	String	Version of the application	
`WL-Instance-ID`	String	Protection mechanism for XSS attacks.	

Application API request elements

Header Name	Data Type	Description	Valid values
`Protocol`	String		`HTTP`
`Worklight Server`	String	Host name or IP address (and possibly port) identifying the IBM Worklight Server	

Experimental IBM Worklight offline user documentation.
This document is provided "as is".
In case of issues, refer to the online user documentation.

Application API request elements

Header Name	Data Type	Description	Valid values
Application ID	String	Unique Identifier of the application within the IBM Worklight Server. Every application deployed on the IBM Worklight Server must have a unique identifier	Up to 256 alphanumeric and underscore characters
Application Environment	String	Name of the environment the **application**is running on	air, android, Androidnative, blackberry, dashboard, desktopbrowser, facebook, igoogle,iOSnative, ipad, iphone, JavaMEnative, mobilewebapp, vista, windows8, windowsphone
Action	String	Requested action	Details in following table

Actions

Action	HTTP Request	Parameters
init	POST	**x, isAjaxRequest** – see the following table showing common parameters.
heartbeat	POST	**x, isAjaxRequest** – see the following table showing common parameters.
logactivity	POST	**x, isAjaxRequest** – see the following table showing common parameters. **activity** – string.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

Actions

Action	HTTP Request	Parameters
query	POST	**x, isAjaxRequest** – see the following table showing common parameters.**filterList** – **JSON** block **parameterList** – **JSON** block **sorterList** – **JSON** block **Note:** When the action is **query**, the request URL has the following structure: `.../query/{Adapter Name}/{Procedure Name}` where **Adapter Name** and **Procedure Name** are strings.
logout	POST	**x, isAjaxRequest** - see the following table showing common parameters.
login	POST	**x, isAjaxRequest** – see the following table showing common parameters. **realm** – string.
updates	POST	**x, isAjaxRequest** – see the following table showing common parameters. **skin** – current skin name (string) **checksum** – the checksum of the current skin (string) **skinLoaderChecksum** – the checksum of the skin selection code (string)
getup	POST	**x, isAjaxRequest** - see the following table showing common parameters.
deleteup	POST	**x, isAjaxRequest** – see the following table showing common parameters. **userprefkey** – the user preference to delete.
getuserinfo	POST	**x, isAjaxRequest** – see the following table showing common parameters.
getgadgetprefs	POST	**x, isAjaxRequest** - see the following table showing common parameters.

Experimental IBM Worklight offline user documentation.
This document is provided "as is".
In case of issues, refer to the online user documentation.

Actions

Action	HTTP Request	Parameters
notifications	POST	**x, isAjaxRequest** – see the following table showing common parameters. **subscribe** – **JSON** string containing subscribe options **unsubscribe** – when specified, designates an unsubscribe action **updateToken** – the update notification token (string) **adapter** – the name of the notification adapter (string) **eventSource** – the name of the notification event source (string) **alias** – notification subscription alias (string)
fbcallback	GET or POST	**x, isAjaxRequest** – see the following table showing common parameters. **popup** – string
composite	POST	**x, isAjaxRequest** - see the following table showing common parameters. **requests** – a **JSON** string containing information about other actions to invoke. This action is used to combine several actions in a single **HTTP** request.
appversionaccess	GET	**x, isAjaxRequest** – see the following table showing common parameters.
setup	POST	**x, isAjaxRequest** - see the following table showing common parameters. **userprefs** contains JSON pairs of preference key and value
authentication	POST	**x, isAjaxRequest** - see the following table showing common parameters. **action** values are `popup`, `test`, or `test_img`

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

Actions

Action	HTTP Request	Parameters
authenticate	POST	**x, isAjaxRequest** - see the following table showing common parameters. This is an empty handler used to allow the client to respond to authentication challenges with a challengeResponse that cannot fit in a single header or when all headers combined are bigger than the limit for header size.

Common parameters

Parameter	Values	Comments
x	A random decimal number	Included with all **GET** and **POST** requests to prevent caching by the Vista Sidebar **HTTP** engine.
isAjaxRequest	`true`	Included with every **GET** and **POST** request only from Adobe™ AIR application.
_	None	Included with every **POST** request only from Webkit-based browsers and application frameworks: Safari, Chrome, and Adobe AIR.

Web application resource requests

Use the following request structure to perform a web application resource request:

```
{Protocol}://{Worklight Server}/apps/services/www/{Application ID}/{Application Environment}/{Applica
```

Request elements

See *Application API Request Elements* for details about the following request elements: **Protocol**, **Worklight Server**, **Application ID**, and **Application Environment**.

Element	Data Type	Description	Valid Values
Application Resource Path	String	HTML, image, JavaScript, CSS, and any other `application resource`	Example values: `img/bg.png`, `myWidget.html`, `js/myWidget.js`

Preview application resource requests

Use the following request structure to preview application resource requests:

```
{Protocol}://{Worklight Server}/apps/services/preview/{Application ID}/{Application Environment}/{App
```

Experimental IBM Worklight offline user documentation. This document is provided "as-is". In case of issues, refer to the online user documentation.

Request elements

See *Application API Request Elements* for details about the following request elements: **`Protocol`**, **`Worklight Server`**, **`ApplicationID`**, and **`Application Environment`**.

Element	Data Type	Description	Valid Values
`Application Resource Path`	String	HTML, image, JavaScript, CSS, and any other application resource	Example values: `img/bg.png`, `myWidget.html`, `js/myWidget.js`

Console API requests

Use the following request structure to perform a console API request:

```
http://{hostname}:{port}/{context-root}/console/api/{api-context}/{action}/{parameters}
```

Actions

API Context	Action	HTTP Request	Parameters
Adapters	**`delete`**	POST	**`adapterName`**
	`get`	GET	**`adapterName`**
	`all`	GET	None
	`upload`	POST	**`adapterName`**, **`input`**
Applications	**`getPublishUrl`**	GET	**`gadgetAppId`**
	`parseCSV`	POST	**`CSV file`**
	`delete`	POST	**`applicationName`**
	`deleteGadgetApplication`	POST	**`gadgetAppId`**
	`setAccessRule`	POST	**`gadgetAppId`** (mandatory), **`action`** (mandatory: `delete`, `block`, or `notify`), **`message`** (mandatory), **`downloadLink`** (optional)
	`setAuthenticityRule`	POST	**`gadgetAppId`** (mandatory), **`action`** (mandatory: `disabled`, `ignored`, or `enabled`)
	`setVersionLock`	POST	**`gadgetAppId`**, **`lock`** (`true` or `false`)
	`getBinaryApp`	GET	**`gadgetAppId`**
	`all`	GET	None
	`get`	GET	**`applicationName`**
	`upload`	POST	**`input`**, **`applicationFolderPath`**
Push	**`unsubscribeSMS`**	POST	**`phoneNumbers`**
Push, Applications	**`all`**	GET	None
Push, Mediators	**`all`**	GET	None
	`get`	GET	`gcm`, `apns`, or `mpns`
Push, Event Sources	**`all`**	GET	None
	`get`	GET	**`adapterName/eventSourceName`**

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Actions

API Context	Action	HTTP Request	Parameters
Users	**`userName`**	GET	None
	`logout`	GET	None

- To retrieve a specific adapter:
  `http://myWorklightServerHost:10080/myProjectRoot/console/api/adapters/get/myAdapterName`
- To retrieve all applications:
  `http://myWorklightServerHost:10080/myProjectRoot/console/api/applications/all`

Experimental IBM Worklight offline user documentation.
This document is provided "as is".
In case of issues, refer to the online user documentation.

Chapter 7. Application Center

Install, configure, administer, and use an enterprise application store.

The Application Center provides an enterprise application store for sharing applications across your organization for use in that organization.

Introduction to the Application Center

Tells you about the Application Center: what it is for, the different components, how to get started, and the files in the distribution.

The sale of mobile devices now exceeds that of personal computers. Consequently, mobile applications become critical for businesses.

The Application Center is a tool to make sharing mobile applications within an organization easier.

You can use the Application Center as an enterprise application store. With the Application Center, you can target some mobile applications to particular groups of users within the company.

A development team can also use the Application Center during the development phase of an application to share applications with testers, designers, or executives in the company. In such a scenario, it makes collaboration easier between all the people who are involved in the development process.

Concept of the Application Center

The Application Center can be used as an Enterprise application store and is a means of sharing information among different team members within a company.

The concept of the Application Center is similar to the concept of the Apple public App Store or the Android Market, except that it targets only private usage within a company.

By using the Application Center, users from the same company or organization download applications to mobile phones or tablets from a single place that serves as a repository of mobile applications.

The Application Center targets mobile applications that are installed on the device itself. Those applications can be native applications that are built by using the device SDK or hybrid applications that mix native and web content. The Application Center does not target mobile web applications; such applications are delivered to the mobile device web browser through a URL like a website.

In the current version, the Application Center supports applications that are built for the Google Android platform, the Apple iOS platform, and the BlackBerry platform for OS versions 6 and 7. (BlackBerry OS 10 is not supported by the current version of the Worklight Application Center.)

Experimental IBM Worklight of the user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

The Application Center manages mobile applications; it supports any kind of Android, iOS, or BlackBerry application, including applications that are built on top of the IBM Worklight platform.

You can use the Application center as part of the development process of an application. A typical scenario of the Application Center is a team building a mobile application; the development team creates a new version of an Android, iOS, or BlackBerry application. The development team wants this new version to be reviewed and tested by the extended team. A developer goes to the Application Center console and uploads the new version of the application to the Application Center. As part of this process, the developer can enter a description of the application version. For example, the description could mention the elements that the development team added or fixed from the previous version. The new version of the application is then available to the other members of the team.

Another person, for example, a beta tester, can launch the Application Center installer application, the mobile client, to locate this new version of a mobile application in the list of available applications and install it on his mobile device. After testing the new version, the beta tester can rate the application and submit feedback. The feedback is visible to the developer from the Application Center console.

The Application Center is a convenient way to share mobile applications within a company or a group; it is a means of sharing information among team members.

General architecture

The Application Center is composed of these main elements: a server-side component, a repository, an administration console, and a mobile client application.

Server-side component

The server-side component is a Java™ Enterprise application that must be deployed in a web application server such as IBM WebSphere or Apache Tomcat.

The server-side component consists of an administration console and a mobile application. This mobile application installs the mobile applications available to the client-side component.

The web console and the installer application communicate through REST services with the server component.

Several services compose the Application Center server-side component; for example, a service that lists available applications, a service that delivers the application binary files to the mobile device, or a service that registers feedback and ratings.

Repository

A database that stores information such as which application is installed on which devices, the feedback about applications, and the mobile application binary files. The Application Center application is associated with the database when you configure the Application Center for a particular web application server and a supported database.

Experimental IBM Worklight online user documentation. This document is provided as-is. In case of issues, refer to the online user documentation.

Administration console

A web console through which administrators can manage applications, user access rights to install applications, user feedback about mobile applications, and details about applications installed on devices. See "The Application Center console" on page 452.

Mobile client application

You use the mobile client to install applications on a mobile device and to send feedback about an application to the server. See "The mobile client" on page 473.

The following figure shows an overview of the architecture.

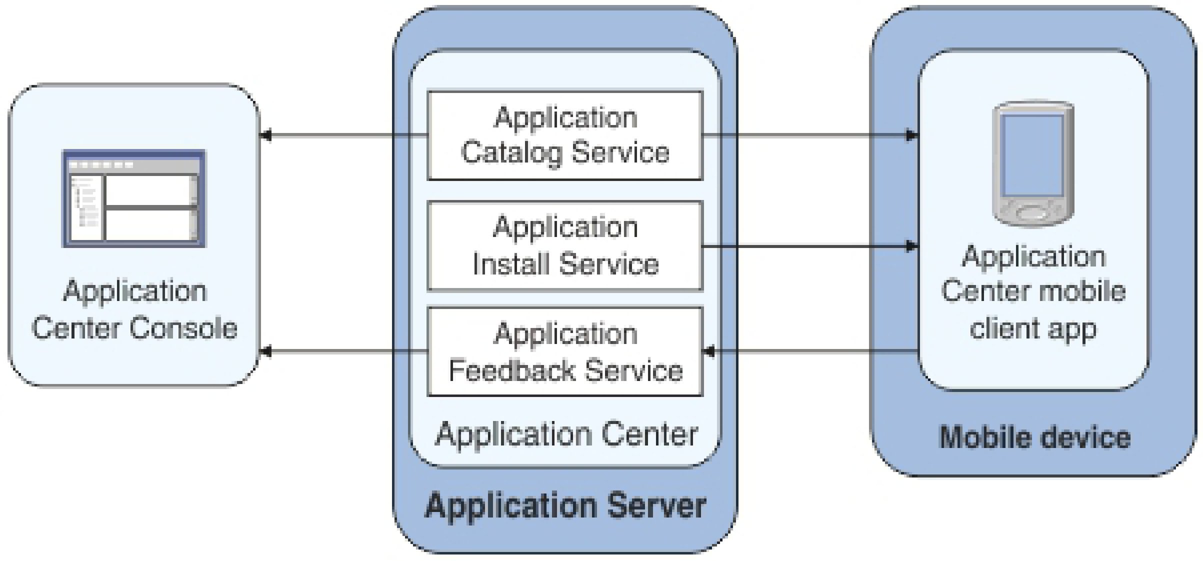

Application Center Console
Application Catalog Service
Application Install Service
Application Feedback Service
Application Center
Application Server
Application Center mobile client app
Mobile device

Figure 68. Architecture of the Application Center

From the Application Center console you can:
- Upload different versions of mobile applications.
- Remove unwanted applications.
- Control access to applications.

Access to the applications stored in the Application Center can be controlled from the Application Center console. Each application is associated with the list of people that can install the application.
- View feedback that mobile users have sent about an application.
- Obtain information about applications installed on a device.
- Make an application inactive so that it is not visible in the available applications for download.

From the mobile client you can:
- List available mobile applications.
- Install a new application on a device.
- Send feedback about an application.

Experimental IBM documentation. This document is for reference only. In case of issues, refer to the official documentation.

The Application Center supports applications for Android, iOS, and BlackBerry devices. Therefore, the mobile client comes in several versions: an Android, an iOS, and a BlackBerry version.

These mobile applications are built on the Worklight platform. You will find instructions in this document about how to configure the Application Center server-side component on various Java application servers after IBM Worklight is installed, as well as how to build Worklight applications for the Application Center client.

Preliminary information

To use the Application Center, you must configure security settings, start the web application server where IBM Worklight is installed, start the Application Center console, and log in.

When you install IBM Worklight, the Application Center is automatically installed in the specified application server.

If you install the Application Center in WebSphere Application Server Liberty profile, the server is created and located in `installation-directory/server`.

After the installation is complete, you must configure the security settings for the applications. See "Configuration of the Application Center after installation" on page 429 or, if you are using LDAP authentication, "Managing users with LDAP" on page 436.

The following example shows how to start the server and then the Application Center console on Liberty profile.

You can start the Liberty server by using the `server` command located in the directory `installation-directory/server/wlp/bin`.

To start the server, use the command:

```
server start worklightServer
```

When the server is running, you can start the Application Center console by entering this address in your browser:

```
http://localhost:9080/applicationcenter/
```

You are requested to log in. By default, the Application Center installed on Apache Tomcat or WebSphere Liberty Profile has two users defined for this installation:
- **demo** with password `demo`
- **appcenteradmin** with password `admin`

To start using the Application Center console, refer to "The Application Center console" on page 452.

To install and run the mobile client on:
- Android operating system: see "Installing the client on an Android mobile device" on page 473
- iOS operating system: see "Installing the client on an iOS mobile device" on page 477

Experimental IBM Worklight online user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

- BlackBerry OS 6 and OS 7: see “Installing the client on a BlackBerry mobile device” on page 478.

Distribution structure

The components of the Application Center are installed in the `ApplicationCenter` directory, which contains several subdirectories.

These elements are installed on disk in the `ApplicationCenter` directory.

Table 109. Content of the ApplicationCenter directory and its subdirectories

`ApplicationCenter` **and subdirectories**	**File name and description**
`ApplicationCenter/installer`	`IbmApplicationCenter.apk` The Android version of the Application Center Mobile client.
`ApplicationCenter/installer/IBMAppCenterBlackBerry6`	Contains the BlackBerry project for the mobile Client for OS V6 and V7. You must compile this project to create the BlackBerry version of the mobile client.
`ApplicationCenter/installer/IBMAppCenter`	Contains the Worklight Studio project for the mobile Client. You must compile this project to create the iOs version of the mobile client.

`ApplicationCenter/console/`	`appcenterconsole.war`
	
	The WAR file for the Application Center console user interface web application.
	
	`applicationcenter.war`
	
	The WAR file for the Application Center REST services web application.
	
	`applicationcenter.ear`
	
	The enterprise application archive (EAR) file to be deployed under IBM PureApplication System.

`ApplicationCenter/databases`	`create-appcenter-derby.sql` The SQL script to re-create the application center database on derby.`create-appcenter-oracle.sql` The SQL script to re-create the application center database on Oracle.`create-appcenter-db2.sql` The SQL script to re-create the application center database on DB2.`create-appcenter-mysql.sql` The SQL script to re-create the application center database on mySQL.

Experimental IBM Worklight offline user documentation
This document is provided as is
In case of issues, refer to the online user documentation.

Table 109. Content of the ApplicationCenter directory and its subdirectories (continued)

`ApplicationCenter` and subdirectories	File name and description

`ApplicationCenter/tools`	`android-sdk`
	
	The directory that contains the part of the Android SDK required by the Application Center console.
	
	`applicationcenterdeploytool.jar`
	
	The JAR file that contains the Ant task to deploy an application to the Application Center.
	
	`acdeploytool.bat`
	
	The startup script of the deploy tool for use on Microsoft Windows systems.`build.xml`
	
	Example of an Ant script to deploy applications to the Application Center.
	
	`dbconvertertool.sh`
	
	The startup script of the database converter tool for use on UNIX systems.
	
	`dbconvertertool.bat`
	
	The startup script of the database converter tool for use on Microsoft Windows systems.
	
	`dbconvertertool.jar`
	
	The main library of the database converter tool.
	
	`lib`
	
	The directory that contains all Java Archive (JAR) files required by the database converter tool.`json4j.jar`
	
	The required JSon4J java archive file.`README.TXT`
	
	Readme file that explains how to use the deployment tool.

Installation of the Application Center

You can install IBM Worklight with IBM Installation Manager.

The Application Center is part of IBM Worklight Server. To install the Application Center, see "Installation" on page 314 under IBM Worklight Server administration.

When you install an IBM Worklight edition through IBM Installation Manager, the Application Center and Worklight console are installed in the web application

Experimental IBM Worklight offline user documentation.
This document is provided as is.
In case of issues, refer to the online user documentation.

server that you designate. You have minimal additional configuration to do. See "Configuration of the Application Center after installation."

If you chose a manual setup in the installer, see the documentation of the server of your choice.

Configuration of the Application Center after installation

You configure user authentication and choose an authentication method; configuration procedure depends on the web application server that you use.

The Application Center requires user authentication.

You must perform some configuration after the installer deploys the Application Center web applications in the web application server.

The Application Center has two Java Platform, Enterprise Edition (JEE) security roles defined:

- The **`appcenteruser`** role that represents an ordinary user of the Application Center who can install mobile applications from the catalog to a mobile device belonging to that user.
- The **`appcenteradmin`** role that represents a user who can perform administrative tasks through the Application Center console.

You must map the roles to the corresponding sets of users.

If you choose to use an authentication method through a user repository such as LDAP, you can configure the Application Center so that you can use users and groups with the user repository to define the Access Control List (ACL) of the Application Center. This procedure is conditioned by the type and version of the web application server that you use. See "Managing users with LDAP" on page 436 for information about LDAP used with the Application Center.

After you configure authentication of the users of the Application Center, which includes configuring LDAP if you plan to use it, you can, if necessary, define the endpoint of the application resources. You must then build the Application Center mobile client. The mobile client is used to install applications on mobile devices. See "Preparations for using the mobile client" on page 449 for how to build the Application Center mobile client.

Related concepts:

"Managing users with LDAP" on page 436
Use the Lightweight Directory Access Protocol (LDAP) registry to manage users..

Related reference:

"Preparations for using the mobile client" on page 449
To use the mobile client to install applications on mobile devices, you must first import the **`IBMAppCenter`** project into Worklight Studio, or the **`IBMAppCenterBlackBerry6`** project into the BlackBerry Eclipse environment, build the project, and deploy the mobile client in the Application Center.

Definition of the endpoint of the application resources

When you add a mobile application from the Application Center console, the server-side component creates Uniform Resource Identifiers (URI) for the application resources (package and icons). The mobile client uses these URI to manage the applications on your device.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues refer to the online user documentation.

Purpose

To manage the applications on your device, the Application Center console must be able to locate the Application Center REST services and to generate the required number of URI that enable the mobile client to find the Application Center REST services.

By default, the URI protocol, hostname, and port are the same as those defined in the web application server used to access the Application Center console; the context root of the Application Center REST services is `applicationcenter`. When the context root of the Application Center REST services is changed or when the internal URI of the web application server is different from the external URI that can be used by the mobile client, the externally accessible endpoint (protocol, hostname, and port) of the application resources must be defined by configuring the web application server. (Reasons for separating internal and external URI could be, for example, a firewall or a secured reverse proxy that uses HTTP redirection.)

The following figure shows a configuration with a secured reverse proxy that hides the internal address (192.168...). The mobile client must use the external address (**`appcntr.net`**).

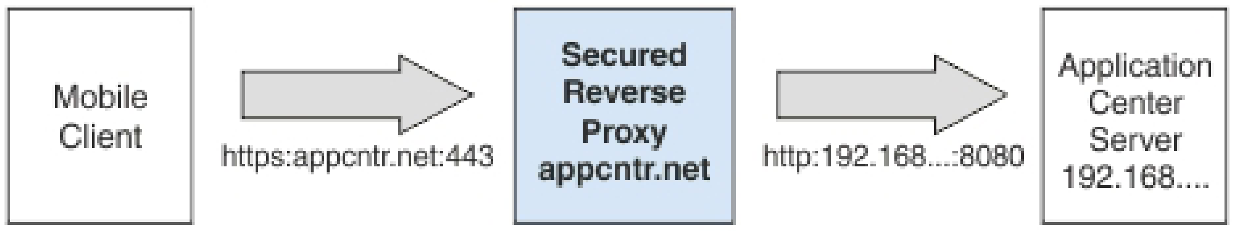

Mobile Client
https:appcntr.net:443
Secured Reverse Proxy appcntr.net
http:192.168...:8080
Application Center Server 192.168....

Figure 69. Configuration with secured reverse proxy

Table 110. The endpoint properties

Property name	Purpose	Example
`ibm.appcenter.services.endpoint`	This property enables the Application Center console to locate the Application Center REST services. The value of this property must be specified as the external address and context root of the `applicationcenter.war` web application.	`https://appcntr.net:443/applicationcenter`
`ibm.appcenter.proxy.protocol`	This property specifies the protocol required for external applications to connect to the Application Center.	https
`ibm.appcenter.proxy.host`	This property specifies the host name required for external applications to connect to the Application Center.	`appcntr.net`

Experimental IBM Worklight user documentation. This document is provided "as is". In case of issues, refer to the user documentation.

Table 110. The endpoint properties (continued)

Property name	Purpose	Example
`ibm.appcenter.proxy.port`	This property specifies the port required for external applications to connect to the Application Center.	443

Related tasks:

"Configuring the endpoint of the application resources (JVM server custom properties)" on page 432
For the full profile, configure the endpoint of the application resources in the custom properties of the JVM server.

Related reference:

"Configuring the endpoint of the application resources (bootstrap properties)" on page 434
For the Liberty profile, configure the endpoint of the application resources in the `bootstrap.properties` file.

"Configuring the endpoint of the application resources (catalina properties)" on page 435
For the Apache Tomcat server, configure the endpoint of the application resources in the `catalina.properties` file.

Configuring WebSphere Application Server full profile

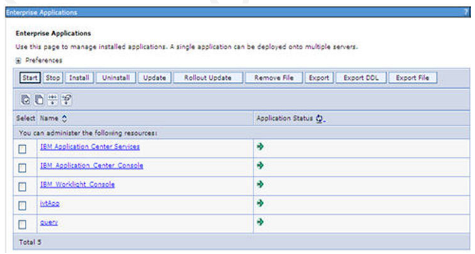
Configure security by mapping the Application Center JEE roles to a set of users for both web applications.

Procedure

You define the basics of user configuration in the WebSphere Application Server console. Access to the console is usually by this address: `https://localhost:9043/ibm/console/`

1. Select **Security** > **Global Security**.
2. Select **Security Configuration Wizard** to configure users.

   You can manage individual user accounts by selecting **Users and Groups** > **Manage Users**.
3. Map the roles **`appcenteruser`** and **`appcenteradmin`** to a set of users.
   a. Select **Servers** > **Server Types** > **WebSphere application servers**.
   b. Select the server.
   c. In the **Configuration** tab, select **Applications** > **Enterprise applications**.



Enterprise Applications

Use this page to manage installed applications. A single application can be deployed onto multiple servers.

⊞ Preferences

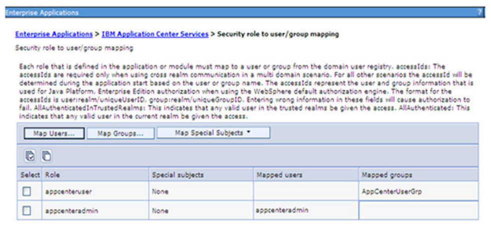
Select	Name	Application Status
You can administer the following resources:		
☐	IBM Application Center Services	➔
☐	IBM Application Center Console	➔
☐	IBM Worklight Console	➔
☐	ivtApp	➔
☐	query	➔
Total 5		

Figure 70. Mapping the Application Center roles

   d. Select **IBM_Application_Center_Services**.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

e. In the **Configuration** tab, select **Details** > **Security role to user/group mapping**.



Enterprise Applications

Enterprise Applications > IBM Application Center Services > Security role to user/group mapping

Security role to user/group mapping

Each role that is defined in the application or module must map to a user or group from the domain user registry. accessIds: The accessIds are required only when using cross realm communication in a multi domain scenario. For all other scenarios the accessId will be determined during the application start based on the user or group name. The accessIds represent the user and group information that is used for Java Platform, Enterprise Edition authorization when using the WebSphere default authorization engine. The format for the accessIds is user:realm/uniqueUserID, group:realm/uniqueGroupID. Entering wrong information in these fields will cause authorization to fail. AllAuthenticatedInTrustedRealms: This indicates that any valid user in the trusted realms be given the access. AllAuthenticated: This indicates that any valid user in the current realm be given the access.

Select	Role	Special subjects	Mapped users	Mapped groups
☐	appcenteruser	None		AppCenterUserGrp
☐	appcenteradmin	None	appcenteradmin	

Figure 71. Mapping the **appcenteruser** *and* **appcenteradmin** *roles: user groups*

f. Perform the necessary customization.

g. Click **OK**.

h. Repeat steps c to g to map the roles for the console web application; in step d, select **IBM_Application_Center_Console**.

i. Click **Save** to save the changes.

Configuring the endpoint of the application resources (JVM server custom properties)

For the full profile, configure the endpoint of the application resources in the custom properties of the JVM server.

About this task

Follow this procedure when you must change the URI protocol, hostname, and port used by the mobile client to manage the applications on your device.

Procedure

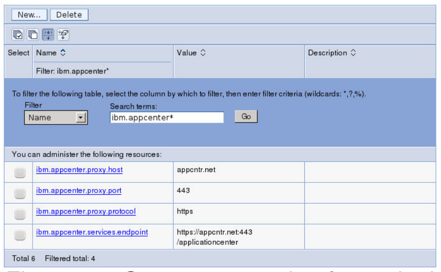
1. Log in to the WebSphere Application Server console.
2. Select **Server** > **Server Types** > **WebSphere application servers**.
3. Select the appropriate application server. In a clustered environment you must configure all the servers in the cluster in the same way.
4. In the **Configuration** tab, under "Server Infrastructure", click the **Java and Process Management** tab and select **Process definition**.
5. In the **Configuration** tab, under "Additional Properties", select **Java Virtual Machine**,
6. In the **Configuration** tab, under "Additional Properties", select **Custom properties**.
7. Click **New**.
   a. In the form, enter `ibm.appcenter.services.endpoint` and assign the full URI of the Application Center REST services (`applicationcenter.war`). In a scenario with a firewall or a secured reverse proxy, this URI must be the external URI and not the internal URI inside the local LAN.
   b. Click **OK**.
   c. In the form, enter `ibm.appcenter.proxy.host`, assign the external hostname, and click **OK**.
   d. **Optional:** In the form, enter `ibm.appcenter.proxy.port`, assign the external port, and click **OK**.
   e. **Optional:** In the form, enter `ibm.appcenter.proxy.protocol`, assign the external protocol, and click **OK**.

Experimental IBM Worklight online user documentation.
This document is provided 'as is'.
In case of issue, refer to the online user documentation.

f. Click **Save** to save the changes.

Results

The following figure shows an example of custom properties with the correct settings.



To filter the following table, select the column by which to filter, then enter filter criteria (wildcards: *,?,%).

Filter: Name

Search terms: ibm.appcenter*

Select	Name (Filter: ibm.appcenter*)	Value	Description
You can administer the following resources:			
☐	ibm.appcenter.proxy.host	appcntr.net	
☐	ibm.appcenter.proxy.port	443	
☐	ibm.appcenter.proxy.protocol	https	
☐	ibm.appcenter.services.endpoint	https://appcntr.net:443 /applicationcenter	

Total 6   Filtered total: 4

Figure 72. Custom properties for endpoint on WebSphere Application Server full profile

Configuring WebSphere Application Server Liberty Profile

Configure the JEE security roles of the Application Center and the data source in the `server.xml` file.

Before you begin

In WebSphere Application Server Liberty Profile, you configure the roles of `appcenteruser` and `appcenteradmin` in the `server.xml` configuration file of the server.

The usual location of this file depends on the operating system:

- On UNIX systems:

  `*INSTALL_DIR*/server/wlp/usr/servers/worklightServer`
- On Microsoft Windows 7 systems:

  `C:\ProgramData\IBM\Worklight\WAS85liberty-server\wlp\usr\servers\worklightServer`
- On Microsoft Windows XP systems:

  `C:\Documents and Settings\All Users\Application Data\IBM\Worklight\WAS85liberty-server\wlp\usr\servers\worklightServer`

About this task

To configure the security roles, you must edit the `server.xml` file. In the `<application-bnd>` element of each `<application>` element, create two `<security-role>` elements. One `<security-role>` element is for the **`appcenteruser`** role and the other is for the **`appcenteradmin`** role. Map the roles to the appropriate user group name **`appcenterusergroup`** or **`appcenteradmingroup`**. These groups are defined through the `<basicRegistry>` element. You can customize this element or replace it entirely with an `<ldapRegistry>` element or a `<safRegistry>` element.

Then, to maintain good response times with a large number of installed applications, for example with 80 applications, you should configure a connection pool for the Application Center database.

Experimental IBM Worklight offline user documentation
This document is provided "as-is".
In case of issues, refer to the online user documentation.

Procedure

1. Edit the server.xml file.

   For example:

```
<security-role name="appcenteradmin">
  <group name="appcenteradmingroup"/>
</security-role>
<security-role name="appcenteruser">
  <group name="appcenterusergroup"/>
</security-role>


<basicRegistry id="appcenter">
  <user name="admin" password="admin"/>
  <user name="guest" password="guest"/>
  <user name="demo" password="demo"/>
  <group name="appcenterusergroup">
        <member name="guest" />
        <member name=" demo" />
  </group>
  <group name="appcenteradmingroup">
        <member name="admin" id="admin"/>
  </group>
</basicRegistry>
```

2. Edit the server.xml file to define the **AppCenterPool** size.

```
<connectionManager id="AppCenterPool" minPoolSize="10" maxPoolSize="40"/>
```

3. In the <dataSource> element, define a reference to the connection manager:

```
<dataSource id="APPCNTR" jndiName="jdbc/AppCenterDS" connectionManagerRef="AppCenterPool"
 ...
</dataSource>
```

Configuring the endpoint of the application resources (bootstrap properties)

For the Liberty profile, configure the endpoint of the application resources in the bootstrap.properties file.

Purpose

Follow this procedure when you must change the URI protocol, hostname, and port used by the mobile client to manage the applications on your device.

Properties

Edit the bootstrap.properties file in the same directory as the server.xml file. If this file does not exist, create it. This file must contain these properties.

Table 111. Bootstrap properties for configuring the endpoint of the application resources

Property	Description
ibm.appcenter.services.endpoint	The URI of the Application Center REST services (applicationcenter.war). In a scenario with a firewall or a secured reverse proxy, this URI must be the external URI and not the internal URI inside the local LAN.
ibm.appcenter.proxy.protocol	The protocol of the application resources URI. This property is optional. It is only needed if the protocol of the external and of the internal URI are different.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Table 111. Bootstrap properties for configuring the endpoint of the application resources (continued)

Property	Description
ibm.appcenter.proxy.host	The hostname of the application resources URI.
ibm.appcenter.proxy.port	The port of the application resources URI. This property is optional. It is only needed if the protocol of the external and of the internal URI are different.

Example of setting properties for configuring the endpoint

This example shows the settings of the properties in the `bootstrap.properties` file required for configuring the endpoint of the application resources.

```
ibm.appcenter.services.endpoint=https://appcntr.net:443/applicationcenter
ibm.appcenter.proxy.protocol=https
ibm.appcenter.proxy.host=appcntr.net
ibm.appcenter.proxy.port=443
```

Configuring Apache Tomcat

You must configure the JEE security roles for the Application Center on the Apache Tomcat web application server.

Procedure

1. In the Apache Tomcat web application server, you configure the roles of **appcenteruser** and **appcenteradmin** in the `conf/tomcat-users.xml` file. The installation creates the following users:

   ```
   <user username="appcenteradmin" password="admin" roles="appcenteradmin"/>
   <user username="demo" password="demo" roles="appcenteradmin"/>
   <user username="guest" password="guest" roles="appcenteradmin"/>
   ```

2. You can define the set of users as described in the Apache Tomcat documentation, Realm Configuration HOW-TO.

Configuring the endpoint of the application resources (catalina properties)

For the Apache Tomcat server, configure the endpoint of the application resources in the `catalina.properties` file.

Purpose

Follow this procedure when you must change the URI protocol, hostname, and port used by the mobile client to manage the applications on your device.

Properties

Edit the `catalina.properties` file in the `conf` directory of your Apache Tomcat installation. This file must contain these properties.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

Table 112. Bootstrap properties for configuring the endpoint of the application resources

Property	Description
ibm.appcenter.services.endpoint	The URI of the Application Center REST services (`applicationcenter.war`). In a scenario with a firewall or a secured reverse proxy, this URI must be the external URI and not the internal URI inside the local LAN.
ibm.appcenter.proxy.protocol	The protocol of the application resources URI. This property is optional. It is only needed if the protocol of the external and of the internal URI are different.
ibm.appcenter.proxy.host	The hostname of the application resources URI.
ibm.appcenter.proxy.port	The port of the application resources URI. This property is optional. It is only needed if the protocol of the external and of the internal URI are different.

Example of setting catalina properties for configuring the endpoint

This example shows the settings of the properties in the `catalina.properties` file required for configuring the endpoint of the application resources.

```
ibm.appcenter.services.endpoint=https://appcntr.net:443/applicationcenter
ibm.appcenter.proxy.protocol=https
ibm.appcenter.proxy.host=appcntr.net
ibm.appcenter.proxy.port=443
```

Managing users with LDAP

Use the Lightweight Directory Access Protocol (LDAP) registry to manage users..

If you plan to use an LDAP registry with the Application Center, you must configure your WebSphere Application Server or your Apache Tomcat server to use an LDAP registry to authenticate users..

In addition to authentication of users, configuring the Application Center for LDAP also enables you to use LDAP to define the users and groups who can install mobile applications through the Application Center. The means of defining these users and groups is the Access Control List (ACL).

Configuring ACL management with LDAP and WebSphere Application Server V7

Use LDAP to authenticate users and to define the users and groups that can install mobile applications with the Application Center.

About this task

You can configure LDAP based on the federated repository configuration or with the stand-alone LDAP registry. See WebSphere Application Server V7.0 user documentation for how to configure LDAP authentication with WebSphere Application Server V7.

Experimental IBM Worklight online user documentation. In case of issues, refer to the online user documentation. This document is provided "as-is".

Procedure

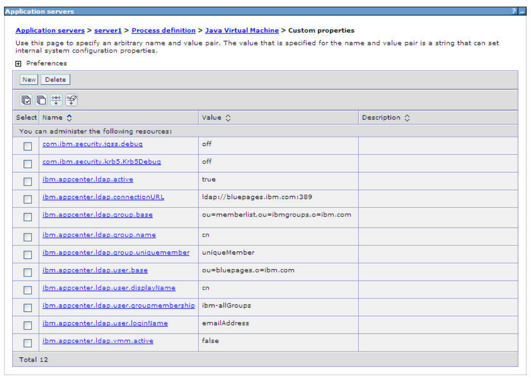
1. Log in to the WebSphere Application Server console.
2. Select **Servers** > **Server Types** > **WebSphere application servers**.
3. Select the appropriate application server.

   In a clustered environment you must configure all the servers in the cluster in the same way.
4. In the **Configuration** tab, under "Server Infrastructure", click the **Java and Process Management** tab and select **Process definition**.
5. In the **Configuration** tab, under "Additional Properties", select **Java Virtual Machine**,
6. In the **Configuration** tab, under "Additional Properties", select **Custom properties**.
7. Click **New**.
   a. In the form, enter `ibm.appcenter.ldap.vmm.active` and assign the value "false".
   b. Click **OK**.
   c. In the form, enter `ibm.appcenter.ldap.active` and assign the value "true".
   d. Click **OK**.
8. Continue to configure the remaining custom properties:
   - `ibm.appcenter.ldap.connectionURL`: LDAP connection URL.
   - `ibm.appcenter.ldap.user.base`: search base for users.
   - `ibm.appcenter.ldap.user.loginName`: LDAP login attribute.
   - `ibm.appcenter.ldap.user.displayName`: LDAP attribute for the user name to be displayed, for example, a person's full name.
   - `ibm.appcenter.ldap.group.base`: search base for groups.
   - `ibm.appcenter.ldap.group.name`: LDAP attribute for the group name.
   - `ibm.appcenter.ldap.group.uniquemember`: LDAP attribute that identifies the members of a group.
   - `ibm.appcenter.ldap.user.groupmembership`: LDAP attribute that identifies the groups that a user belongs to.

   a. In the form, enter the name of a custom property and its value.
   b. Click **OK**.
   c. Repeat steps a and b for each custom property.

   The following figure shows the values to assign to each custom property or resource.

Experimental IBM MobileFirst Platform user documentation. This document is provided 'as is'. In case of issues refer to the online user documentation.


Application servers

Application servers > server1 > Process definition > Java Virtual Machine > Custom properties

Use this page to specify an arbitrary name and value pair. The value that is specified for the name and value pair is a string that can set internal system configuration properties.

⊞ Preferences

Select	Name	Value	Description
You can administer the following resources:			
☐	com.ibm.security.jgss.debug	off	
☐	com.ibm.security.krb5.Krb5Debug	off	
☐	ibm.appcenter.ldap.active	true	
☐	ibm.appcenter.ldap.connectionURL	ldap://bluepages.ibm.com:389	
☐	ibm.appcenter.ldap.group.base	ou=memberlist,ou=ibmgroups,o=ibm.com	
☐	ibm.appcenter.ldap.group.name	cn	
☐	ibm.appcenter.ldap.group.uniquemember	uniqueMember	
☐	ibm.appcenter.ldap.user.base	ou=bluepages,o=ibm.com	
☐	ibm.appcenter.ldap.user.displayName	cn	
☐	ibm.appcenter.ldap.user.groupmembership	ibm-allGroups	
☐	ibm.appcenter.ldap.user.loginName	emailAddress	
☐	ibm.appcenter.ldap.vmm.active	false	
Total 12			

Figure 73. Custom properties and their values (LDAP and WebSphere Application Server V7)

9. **Option:** *If security binding is required, follow this step.* Configure the following custom properties:
   - `ibm.appcenter.ldap.security.binddn`: the distinguished name of the user permitted to search the LDAP directory.
   - `ibm.appcenter.ldap.security.bindpwd`: the password of the user permitted to search the LDAP directory. The password can be encoded with the "WebSphere PropFilePasswordEncoder" utility. Run the utility before you configure the `ibm.appcenter.ldap.security.bindpwd` custom property.

   a. In the form, enter the name of the optional custom property and its value. Set the value of the `ibm.appcenter.ldap.security.bindpwd` property to the encoded password generated by the "WebSphere PropFilePasswordEncoder" utility.

   b. Click **OK**.

   c. Repeat steps a and b for each optional custom property.

What to do next

Save the configuration and restart the server.

Configuration of LDAP authentication (WebSphere Application Server V8.x)

LDAP authentication is achieved based on the federated repository configuration. ACL management configuration of the Application Center uses the Virtual Member Manager API.

You must configure LDAP based on the federated repository configuration. The stand-alone LDAP registry is not supported.

Several different repositories, LDAP and non-LDAP, can be configured in the federated repository.

For information about configuring federated repositories, see the WebSphere Application Server V8.0 user documentation or the WebSphere Application Server V8.5 user documentation, depending on your version.

Experimental IBM Worklight user documentation.
This document is provided "as is".
In case of issues refer to the online user documentation.

Configuration of the Application Center for ACL management with LDAP

Some configuration details of ACL management are specific to the Application Center, because it uses the Virtual Member Manager (VMM) API.

The Application Center refers to these VMM attributes for users:

`uid` represents the user login name.

`sn` represents the full name of the user.

For groups, the Application Center refers only to the VMM attribute **`cn`**.

If VMM attributes are not identical in LDAP, you must map the VMM attributes to the corresponding LDAP attributes.

Configuring LDAP authentication for users and groups (WebSphere Application Server V8.x)

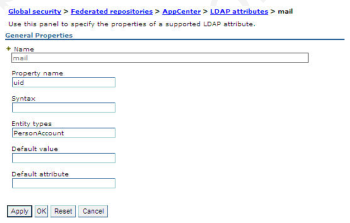
Use LDAP to authenticate users and groups, and to define the users who can install mobile applications with the Application Center.

About this task

You can configure LDAP based on the federated repository configuration only.

Procedure

1. Log in to the WebSphere Application Server console.
2. Select **Security** > **Global security**.
3. Select **Configure** in the "User account repository" section.
4. Select your LDAP repository entry.
5. Under **Additional Properties**, select **LDAP attributes** (WebSphere Application Server V8.0) or **Federated repositories property names to LDAP attributes mapping** (WebSphere Application Server V8.5).
6. Select **Add** > **Supported**.
7. Enter these property values:
   a. For **`Name`** enter your LDAP login attribute.
   b. For **`Property name`** enter **`uid`**.
   c. For **`Entity types`** enter the LDAP entity type.
   d. Click **OK**.


Global security > Federated repositories > AppCenter > LDAP attributes > mail

Use this panel to specify the properties of a supported LDAP attribute.

General Properties

* Name: mail

Property name: uid

Syntax:

Entity types: PersonAccount

Default value:

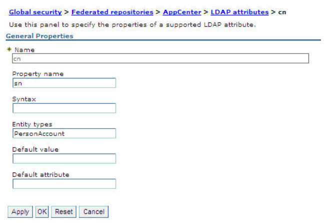
Default attribute:

Figure 74. Associating LDAP login with uid property (WebSphere Application Server V8.0)

8. Select **Add** > **Supported**.
   a. For **`Name`** enter your LDAP attribute for full user name.

Experimental IBM Worklight offline user documentation. This document is provided as is. In case of issues, refer to the online user documentation.

b. For **Property name** enter **sn**.
c. For **Entity types** enter the LDAP entity type.
d. Click **OK**.


Global security > Federated repositories > AppCenter > LDAP attributes > cn

Use this panel to specify the properties of a supported LDAP attribute.

General Properties

* Name	cn
Property name	sn
Syntax	
Entity types	PersonAccount
Default value	
Default attribute	

Figure 75. Associating LDAP full user name with sn property (WebSphere Application Server V8.0)

9. Select **Add** > **Supported** to configure a group name:
   a. For **Name** enter the LDAP attribute for your group name.
   b. For **Property name** enter **cn**.
   c. For **Entity types** enter the LDAP entity type.
   d. Click **OK**.

What to do next

You must enable ACL management with LDAP. See "Enabling ACL management with LDAP (WebSphere Application Server V8.x)."

Enabling ACL management with LDAP (WebSphere Application Server V8.x)

You can enable access control for the Application Center of the users and groups defined through LDAP.

About this task

After you configure LDAP authentication for users and groups, follow these steps from the WebSphere Application Server console to enable ACL management.

Procedure

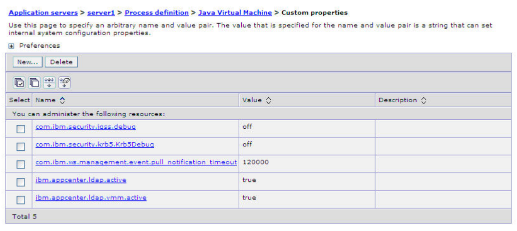
1. Select **Servers** > **Server Types** > **WebSphere application servers**.
2. Select the appropriate application server.
   In a clustered environment you must configure all the servers in the cluster in the same way.
3. In the **Configuration** tab, under "Server Infrastructure", click the **Java and Process Management** tab and select **Process definition**.
4. In the **Configuration** tab, under "Additional Properties", select **Java Virtual Machine**,
5. In the **Configuration** tab, under "Additional Properties", select **Custom properties**.
6. Click **New**.
   a. In the form, enter `ibm.appcenter.ldap.vmm.active` and assign the value "true".
   b. Click **Apply**.
   c. In the form, enter `ibm.appcenter.ldap.active` and assign the value "true".

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

d. Click **OK**.

Results

The following figure shows an example of custom properties with the correct settings.



Application servers > server1 > Process definition > Java Virtual Machine > Custom properties

Use this page to specify an arbitrary name and value pair. The value that is specified for the name and value pair is a string that can set internal system configuration properties.

Select	Name	Value	Description
You can administer the following resources:			
☐	com.ibm.security.jgss.debug	off	
☐	com.ibm.security.krb5.Krb5Debug	off	
☐	com.ibm.ws.management.event.pull_notification_timeout	120000	
☐	ibm.appcenter.ldap.active	true	
☐	ibm.appcenter.ldap.vmm.active	true	

Total 5

Figure 76. ACL management for Application Center with LDAP on WebSphere Application Server V8

What to do next

Save the configuration and restart the server.

To use the VMM API, you must assign the "IdMgrReader" role to the users who run the VMM code, or to the group owners of these users. You must assign this role to all users and groups who have the roles of "appcenteruser" or "appcenteradmin".

In the *<was_home>*`\bin` directory, where *<was_home>* is the home directory of your WebSphere application server, run the **`wsadmin`** command.

After connecting with the WebSphere Application Server administrative user, run the following command:

```
$AdminTask mapIdMgrGroupToRole {-roleName IdMgrReader -groupId your_LDAP_group_id}
```

Run the same command for all the groups mapped to the roles of "appcenteruser" and "appcenteradmin".

For individual users who are not members of groups, run the following command:

```
$AdminTask mapIdMgrUserToRole {-roleName IdMgrReader -userId your_LDAP_user_id}
```

You can assign the special subject "All Authenticated in Application's Realm" as roles for **`appcenteruser`** and **`appcenteradmin`**. If you choose to assign this special subject, **`IdMgrReader`** must be configured in the following way:

```
$AdminTask mapIdMgrGroupToRole {-roleName IdMgrReader -groupId ALLAUTHENTICATED}
```

Enter **`exit`** to end **`wsadmin`**.

Configuring LDAP authentication (Liberty profile)

You configure LDAP authentication by defining an LDAP registry in the `server.xml` file.

About this task

You can configure LDAP authentication of users and groups in the `server.xml` file by defining an LDAP registry.

Experimental IBM Worklight online user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

The usual location of this file depends on the operating system:

- On UNIX systems:

  INSTALL_DIR`/server/wlp/usr/servers/worklightServer`
- On Microsoft Windows 7 systems:

  `C:\ProgramData\IBM\Worklight\WAS85liberty-server\wlp\usr\servers\worklightServer`
- On Microsoft Windows XP systems:

  `C:\Documents and Settings\All Users\Application Data\IBM\Worklight\WAS85liberty-server\wlp\usr\servers\worklightServer`

Procedure

1. To open the `server.xml` descriptor file, enter `{server.config.dir}/server.xml`
2. Insert an LDAP registry definition after the `<httpEndpoint>` element.

Example

```
<ldapRegistry baseDN="o=ibm.com" host="employees.com" id="Employees"
ldapType="IBM Tivoli Directory Server" port="389" realm="AppCenterLdap"
recursiveSearch="true">
  <idsFilters
  groupFilter="(&amp;(cn=%v)(|(objectclass=groupOfNames)(objectclass=groupOfUniqueNames))) " id="Empl
  userFilter="(&amp;(emailAddress=%v)(objectclass=ibmPerson))"
  groupMemberIdMap="ibm-allGroups:member;ibm-allGroups:uniqueMember"
  userIdMap="*:emailAddress"/>
</ldapRegistry>
```

For information about the parameters used in this example, see the WebSphere Application Server V8.5 user documentation.

What to do next

Map users and groups to Application Center roles. See "Mapping users and groups to Application Center roles (LDAP on Liberty profile)."

Mapping users and groups to Application Center roles (LDAP on Liberty profile)

Map Application Center users and groups defined for LDAP to Application Center roles.

About this task

After configuring LDAP authentication, you map users and groups to Application Center roles. Mapping is configured in the `server.xml` file. The mapping configuration is the same for LDAP authentication and basic authentication.

Procedure

1. To open the `server.xml` descriptor file, enter `{server.config.dir}/server.xml`.
2. Insert a security role definition after the LDAP realm section or after the basic registry definition.

Example

This example includes two sets of sample code that show how to code when the group names are unique within LDAP and how to code when the group names are not unique within LDAP.

Experimental IBM Worklight online user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Group names unique within LDAP

This sample code shows how to use the group names **ldapGroupForAppcenteruser** and **ldapGroupForAppcenteradmin** when they exist and are unique within LDAP.

```
<application-bnd>
   <security-role name="appcenteruser" id="appcenteruser">
     <group name="ldapGroupForAppcenteruser" />
   </security-role>
   <security-role name="appcenteradmin" id="appcenteradmin">
     <group name="ldapGroupForAppcenteradmin" />
   </security-role>
 </application-bnd>
```

Group names not unique within LDAP

This sample code shows how to code the mapping when the group names are not unique within LDAP. The groups must be specified with the **access-id** attribute.

```
<application-bnd>
   <security-role name="appcenteruser" id="appcenteruser">
     <group name="ldapGroup"
                  id="ldapGroup"
                  access-id="group:AppCenterLdap/CN=ldapGroup,OU=myorg,
                  DC=mydomain,DC=AD,DC=myco,DC=com"/>
   </security-role>
   ...
 </application-bnd>
```

The **access-id** attribute must refer to the realm name used to specify the LDAP realm. In this sample code, the realm name is **AppCenterLdap**. The remainder of the **access-id** attribute specifies one of the LDAP groups named **ldapGroup** in a way that makes it unique.

If required, use similar code to map the **appcenteradmin** role.

What to do next

Configure ACL management. See “Configuring the Application Center for ACL management with LDAP (Liberty profile).”

Configuring the Application Center for ACL management with LDAP (Liberty profile)

Use LDAP to define the users and groups who can install mobile applications through the Application Center. The means of defining these users and groups is the Access Control List (ACL).

Purpose

To enable ACL management with LDAP. You enable ACL management after you configure LDAP and map users and groups to Application Center roles.

Properties

Edit the `bootstrap.properties` file in the same directory as the `server.xml` file. If this file does not exist, create it. This file must contain all properties, except **ibm.appcenter.ldap.security.binddn** and **ibm.appcenter.ldap.security.bindpwd**, which are only required for security binding. If security binding is required, these two properties must also be included in the `bootstrap.properties` file.

Table 113. Bootstrap properties for configuring ACL management with LDAP (Liberty profile)

Property	Description
ibm.appcenter.ldap.active	Set to true to enable LDAP; set to false to disable LDAP.
ibm.appcenter.ldap.connectionURL	LDAP connection URL.
ibm.appcenter.ldap.user.base	Search base of users.
ibm.appcenter.ldap.user.loginName	LDAP login attribute.
ibm.appcenter.ldap.user.displayName	LDAP attribute for the user name to be displayed, for example, a person's full name.
ibm.appcenter.ldap.group.base	Search base of groups.
ibm.appcenter.ldap.group.name	LDAP attribute for the group name.
ibm.appcenter.ldap.group.uniquemember	LDAP attribute that identifies the members of a group.
ibm.appcenter.ldap.user.groupmembership	LDAP attribute that identifies the groups to which a user belongs.
ibm.appcenter.ldap.security.binddn	Property that identifies the distinguished name of the user permitted to search the LDAP directory. Use this property only if security binding is required.
ibm.appcenter.ldap.security.bindpwd	Property that identifies the password of the user permitted to search the LDAP directory. Use this property only if security binding is required. The password can be encoded with the "Liberty Profile securityUtility" tool. Run the tool and then set the value of this property to the encoded password generated by the tool. Edit the Liberty Profile server.xml file to check whether the **classloader** is enabled to load the JAR file that decodes the password.

Experimental IBM Worklight offline user documentation. This document is provided "as-is". In case of issues, refer to the online documentation.

Example of the entry for checking classloader enablement

Add the following entry, if it is not already present, to the **<application context-root="applicationcenter">** entry in the server.xml file. This entry should appear just before the **</application>** closing tag.

```
<classloader delegation="parentLast">
    <commonLibrary>
      <fileset dir="${wlp.install.dir}/lib" includes="com.ibm.ws.crypto.passwordutil_1.0.jar"/>
    </commonLibrary>
</classloader>
```

Example of setting properties for ACL management with LDAP

This example shows the settings of the properties in the bootstrap.properties file required for ACL management with LDAP.

```
ibm.appcenter.ldap.active=true
ibm.appcenter.ldap.connectionURL="ldap://employees.com:389"
ibm.appcenter.ldap.user.base="ou=employees,o=ibm.com"
ibm.appcenter.ldap.user.loginName=emailAdress
ibm.appcenter.ldap.user.displayName=cn
ibm.appcenter.ldap.group.base="ou=groups,o=ibm.com"
ibm.appcenter.ldap.group.name=cn
ibm.appcenter.ldap.group.uniquemember=uniqueMember
ibm.appcenter.ldap.user.groupmembership=ibm-allGroups
```

Configuring ACL management with LDAP and Apache Tomcat

Configure the Apache Tomcat server for LDAP authentication and configure security (Java Platform, Enterprise Edition) in the `web.xml` file of the Application Center.

Purpose

To configure ACL management of the Application Center; configure LDAP for user authentication, map the Java Platform, Enterprise Edition (JEE) roles of the Application Center to the LDAP roles, and configure the Application Center properties for LDAP authentication.

LDAP user authentication

You must configure a `JNDIRealm` in the `server.xml` file in the `<Host>` element. See the Realm Component on the Apache Tomcat website for more information about configuring a realm.

Example of configuration on Apache Tomcat to authenticate against an LDAP server

This example shows how to configure user authentication on an Apache Tomcat server by comparing with the authorization of these users on a server enabled for LDAP authentication.

```
<Host appBase="webapps" autoDeploy="true" name="localhost" unpackWARs="true">
 ...
<Realm   className="org.apache.catalina.realm.JNDIRealm"

       connectionURL="ldap://bluepages.ibm.com:389"

       userSubtree="true"

       userBase="ou=bluepages,o=ibm.com"

       userSearch="(emailAddress={0})"

       roleBase="ou=ibmgroups,o=ibm.com"

       roleName="cn"

       roleSubtree="true"

       roleSearch="(uniqueMember={0})"

        allRolesMode="authOnly"

       commonRole="appcenter"

/>
 ...
/>
```

The value of **`connectionURL`** is the LDAP URL of your LDAP server.

Experimental IBM Worklight online user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

The **userSubtree**, **userBase**, and **userSearch** attributes define how to use the name given to the Application Center in login form (in the browser message box) to match an LDAP user entry.

In the example, the definition of **userSearch** specifies that the user name is used to match the email address of an LDAP user entry.

The basis or scope of the search is defined by the value of the **userBase** attribute. In LDAP, an information tree is defined; the user base indicates a node in that tree.

The value of **userSubtree** should be set to `true`; if it is `false`, the search is performed only on the direct child nodes of the user base. The child nodes of `ou=bluepages,o=ibm.com` indicate countries, for example, `c=fr`. It is important that the search penetrates the subtree and does not stop at the first level.

For authentication, you define only the **userSubtree**, **userBase**, and **userSearch** attributes. The Application Center also uses JEE security roles. Therefore, you must map LDAP attributes to some JEE roles. These attributes are used for mapping LDAP attributes to security roles:
- **roleBase**
- **roleName**
- **roleSubtree**
- **roleSearch**

In this example, the value of the **roleSearch** attribute matches all LDAP entries with a **uniqueMember** attribute whose value is the Distinguished Name (DN) of the authenticated user.

The **roleBase** attribute specifies a node in the LDAP tree below which the roles are defined.

The **roleSubtree** attribute indicates whether the LDAP search should search the entire subtree, whose root is defined by the value of **roleBase**, or only the direct child nodes.

The **roleName** attribute defines the name of the LDAP attribute.

The **allRolesMode** attribute specifies that you can use the asterisk (*) character as the value of **role-name** in the `web.xml` file. This attribute is optional.

The **commonRole** attribute adds a role shared by all authenticated users. This attribute is optional.

Mapping the JEE roles of the Application Center to LDAP roles

After you define the LDAP request for the JEE roles, you must change the `web.xml` file of the Application Center to map the JEE roles of "appcenteradmin" and "appcenteruser" to the LDAP roles.

These examples, where LDAP users have LDAP roles called "MyLdapAdmin" and "MyLdapUser", show where and how to change the `web.xml` file.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

The security-role-ref element in the JAX_RS servlet

```
<servlet>

    <servlet-name>MobileServicesServlet</servlet-name>


        <servlet-class>org.apache.wink.server.internal.servlet.RestServlet</servlet-class>

            <init-param>

                <param-name>javax.ws.rs.Application</param-name>

                <param-value>com.ibm.puremeap.services.MobileServicesServlet</param-value>

            </init-param>

            <load-on-startup>1</load-on-startup>

           <security-role-ref>

               <role-name>appcenteradmin</role-name>

               <role-link>MyLdapAdmin</role-link>

          </security-role-ref>

           <security-role-ref>

               <role-name>appcenteruser</role-name>

               <role-link>MyLdapUser</role-link>

           </security-role-ref>

</servlet>
```

The security-role element

```
    <security-role>

        <role-name>MyLdapAdmin</role-name>

    </security-role>
```

The auth-constraint element

After you edit the `security-role-ref` and the `security-role` elements, you can use the roles defined in the `auth-constraint` elements to protect the web resources. See the `appcenteradminConstraint` element and the `appcenteruserConstraint` element in this example for definition of the web resource collection to be protected by the role defined in the `auth-constraint` element.

```
    <security-constraint>

        <display-name>appcenteruserConstraint</display-name>


        <web-resource-collection>

            <web-resource-name>appcenteruser</web-resource-name>
```

Experimental IBM Worklight online user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

```
            <url-pattern>/installers.html</url-pattern>


            <url-pattern>/service/device/*</url-pattern>

            <url-pattern>/service/directory/*</url-pattern>

            <url-pattern>/service/plist/*</url-pattern>

            <url-pattern>/service/auth/*</url-pattern>

            <url-pattern>/service/application/*</url-pattern>

            <url-pattern>/service/desktop/*</url-pattern>

            <url-pattern>/service/principal/*</url-pattern>

            <url-pattern>/service/acl/*</url-pattern>


            <url-pattern>/service/userAndConfigInfo</url-pattern>

            <http-method>DELETE</http-method>

            <http-method>GET</http-method>

            <http-method>POST</http-method>
   <http-method>PUT</http-method>

            <http-method>HEAD</http-method>

        </web-resource-collection>


        <auth-constraint>

            <role-name>MyLdapUser</role-name>

        </auth-constraint>

        <user-data-constraint>

            <transport-guarantee>NONE</transport-guarantee>

        </user-data-constraint>

    </security-constraint>
```

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Configuring the Application Center with LDAP

You can define properties in the `catalina.properties` file or the `web.xml` file.

Table 114. Bootstrap properties for configuring ACL management for LDAP on Apache Tomcat

Property	Description
ibm.appcenter.ldap.active	Set to `true` to enable LDAP; set to `false` to disable LDAP.
ibm.appcenter.ldap.connectionURL	LDAP connection URL.
ibm.appcenter.ldap.user.base	Search base of users.
ibm.appcenter.ldap.user.loginName	LDAP login attribute.

Table 114. Bootstrap properties for configuring ACL management for LDAP on Apache Tomcat (continued)

Property	Description
ibm.appcenter.ldap.user.displayName	LDAP attribute for the user name to be displayed, for example, a person's full name.
ibm.appcenter.ldap.group.base	Search base of groups.
ibm.appcenter.ldap.group.name	LDAP attribute for the group name.
ibm.appcenter.ldap.group.uniquemember	LDAP attribute that identifies the members of a group.
ibm.appcenter.ldap.user.groupmembership	LDAP attribute that identifies the groups to which a user belongs.
ibm.appcenter.ldap.security.binddn	Property that identifies the distinguished name of the user permitted to search the LDAP directory. Use this property only if security binding is required.
ibm.appcenter.ldap.security.bindpwd	Property that identifies the password of the user permitted to search the LDAP directory. Use this property only if security binding is required.

The example shows properties defined in the catalina.properties file.

```
ibm.appcenter.ldap.active=true

ibm.appcenter.ldap.connectionURL=ldap://bluepages.ibm.com:389

ibm.appcenter.ldap.user.base=ou=bluepages,o=ibm.com

ibm.appcenter.ldap.user.loginName=emailAddress

ibm.appcenter.ldap.user.displayName=cn


ibm.appcenter.ldap.group.base=ou=memberlist,ou=ibmgroups,o=ibm.com

ibm.appcenter.ldap.group.name=cn

ibm.appcenter.ldap.group.uniquemember=uniquemember
```

Preparations for using the mobile client

To use the mobile client to install applications on mobile devices, you must first import the **IBMAppCenter** project into Worklight Studio, or the **IBMAppCenterBlackBerry6** project into the BlackBerry Eclipse environment, build the project, and deploy the mobile client in the Application Center.

Prerequisites for building the Application Center installer

The Application Center comes with an Android, an iOS, and a BlackBerry version of the client application that runs on the mobile device. This mobile application that supports installation of applications on your mobile device is called the mobile client. The mobile client is an IBM Worklight mobile application.

The Worklight project **IBMAppCenter** contains both the Android and the iOS versions of the client.

Experimental IBM Worklight offline user documentation. This document is provided as-is. In case of issues, refer to the online user documentation.

The BlackBerry project **IBMAppCenterBlackBerry6** contains the version of the client for BlackBerry OS 6 and OS 7 devices.

The Android version of the mobile client is included in the software delivery in the form of an Android application package (.apk) file. You can find the `ibmapplicationcenter.apk` file in the directory `ApplicationCenter/installer`.

To build the Android version, you must have the latest version of the Android development tools.

The iOS version for iPad and iPhone is not delivered as a compiled application. The application must be created from the Worklight project named **IBMAppCenter**. This project is also delivered as part of the distribution in the `ApplicationCenter/installer` directory.

To build the iOS version, you must have the appropriate Worklight and Apple software. The version of Worklight Studio must be the same as the version of Worklight Server on which this documentation is based. The Apple Xcode version is V4.5.

The BlackBerry version is not delivered as a compiled application. The application must be created from the BlackBerry project named **IBMAppCenterBlackBerry6**. This project is delivered as part of the distribution in the `ApplicationCenter/ installer` directory.

To build the BlackBerry version, you must have the BlackBerry Eclipse IDE (or Eclipse with the BlackBerry Java plug-in) with the BlackBerry SDK 6.0. The application also runs on BlackBerry OS 7 when compiled with BlackBerry SDK 6.0.

Download the software from: https://developer.blackberry.com/java/download/eclipse/.

1. Start the BlackBerry Eclipse IDE.
2. Select **Help** > **Install New Software** > **Work with: BlackBerry Update Site**.
3. Expand the BlackBerry Java Plug-in Category and select "BlackBerry Java SDK 6.0.x.y."

Android and iOS

You must import the **IBMAppCenter** project and then build the project.

Importing the IBMAppCenter project into Worklight Studio

Follow the normal procedure to import a project into Worklight Studio.

1. Select **File** > **Import**.
2. Select **General** > **Existing Project into Workspace**.
3. On the next page, select **Select root directory** and locate the root of the **IBMAppCenter** project.
4. Select "IBMAppCenter project".
5. Click **Finish** to import the **IBMAppCenter** project into Worklight Studio.

Experimental IBM Worklight official user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

Building the project

Build the **IBMAppCenter** project. The Worklight project contains a single application named AppCenter. Right-click the application and select **Run as** > **Build All and Deploy**.

Android

Worklight Studio generates a native Android project in IBMAppCenter/apps/AppCenter/android/native. A native Android development tools (ADT) project is located directly under the android/native folder. You can compile and sign this project using the ADT tools.

Refer to the Android site for developers https://developer.android.com/index.html for more specific Android information that affects the mobile client application.

iOS

Worklight Studio generates a native iOS project in IBMAppCenter/apps/AppCenter/iphone/native. The IBMAppCenterAppCenterIphone.xcodeproj file is in the ios/native folder. This file is the Xcode project that you have to compile and sign by using Xcode.

Refer to the Apple developer site https://developer.apple.com/ to learn more about how to sign the iOS mobile client application.

For signing the iOS application, you must change the **Bundle Identifier** of the application to a bundle identifier that can be used with the provisioning profile that you are using. The value is defined in the Xcode project settings as com.*your_internet_domain_name*.appcenter.

Refer to the documentation of Worklight Studio for more about how you can create hybrid mobile applications with Worklight Studio.

BlackBerry

You must import the BlackBerry project and then build the project.

Importing the IBMAppCenterBlackBerry6 project into Eclipse

Follow the normal procedure to import a project into the BlackBerry Eclipse IDE.

1. Select **File** > **Import**.
2. Select **General** > **Existing Project into Workspace**.
3. On the next page, select **Select root directory** and locate the root of the **IBMAppCenterBlackBerry6** project.
4. Select "IBMAppCenterBlackBerry6 project".
5. Click **Finish** to import the **IBMAppCenterBlackBerry6** project into the BlackBerry Eclipse IDE.

Building the BlackBerry project

The **IBMAppCenterBlackBerry6** project is a native BlackBerry application that requires protected BlackBerry API. Therefore, you must first obtain a signature to sign the project. In your web browser, open https://www.blackberry.com/SignedKeys/codesigning.html. Follow the instructions to obtain the signature,

Experimental IBM Worklight online user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

which consists of several keys. All signature keys must be imported into Eclipse by using **Window** > **Preferences** > **BlackBerry Java Plugin** > **Signature Tool**.

To build the **`IBMAppCenterBlackBerry6`** project:

1. Right-click the project and select **BlackBerry** > **Package Project(s)**.

   This action packages the project.
2. Right-click the project and select **BlackBerry** > **Sign with Signature Tool**.

   This action signs the project.

The result is located in a generated directory called `deliverables`. This directory contains two subdirectories:

Standard
This directory contains the packaged application for uploading with USB cable to the device. This method is incompatible with the packaging required for the IBM Application Center server.

Web This directory contains the packaged application for uploading over the air. This method is compatible with the IBM Application Center. Therefore, use this directory and **not** the `Standard` directory. Place this directory into an archive (.zip) file.

Important: Make sure that the archive file does not contain the `Standard` directory.

Refer to the BlackBerry site for developers for more specific information that affects the mobile client application for BlackBerry projects.

For Experts

Look and feel and various features are controlled by a central property file called `appcenter.properties` in the directory `IBMAppCenterBlackBerry6/src/main/resources`. If you want to disable various features, you can adapt this property file before you build the project. For example, you can disable the feature for reverting the installation of an application to a previous version.

Deploying the mobile client in the Application Center

The Android, iOS, and BlackBerry versions of the mobile client must be deployed to the Application Center. To do so, you must upload the Android application package (.apk) files, iOS application (.ipa) files, and BlackBerry `Web` directory archive files to the Application Center.

Follow the steps described in “Adding a mobile application” on page 455 to add the mobile client application for Android, iOS, and BlackBerry. Make sure that you select the **`Installer`** application property to indicate that the application is an installer. Selecting this property enables mobile device users to install the mobile client application easily over the air. To install the mobile client, see “Installing the client on an Android mobile device” on page 473, “Installing the client on an iOS mobile device” on page 477, or “Installing the client on a BlackBerry mobile device” on page 478.

The Application Center console

With the Application Center console, you can manage the repository of the Application Center and your applications.

Experimental IBM Worklight offline user documentation
In case of issue, refer to the online user documentation.
This document is provided "as-is".

The Application Center console is a web application to manage the repository of the Application Center. The Application Center repository is the central location where you store the mobile applications that can be installed on mobile devices.

Use the Application Center console to:
- Upload Android or iOS applications.
- Manage several different versions of mobile applications.
- Review the feedback of testers of mobile applications.
- Define the users who have the rights to list and install an application on the mobile devices.
- Track which applications are installed on which devices.

Note:

Only users with the administrator role can log in to the Application Center console.

Starting the Application Center console

You can start the Application Center with your web browser and log in if you have the administrator role.

Procedure

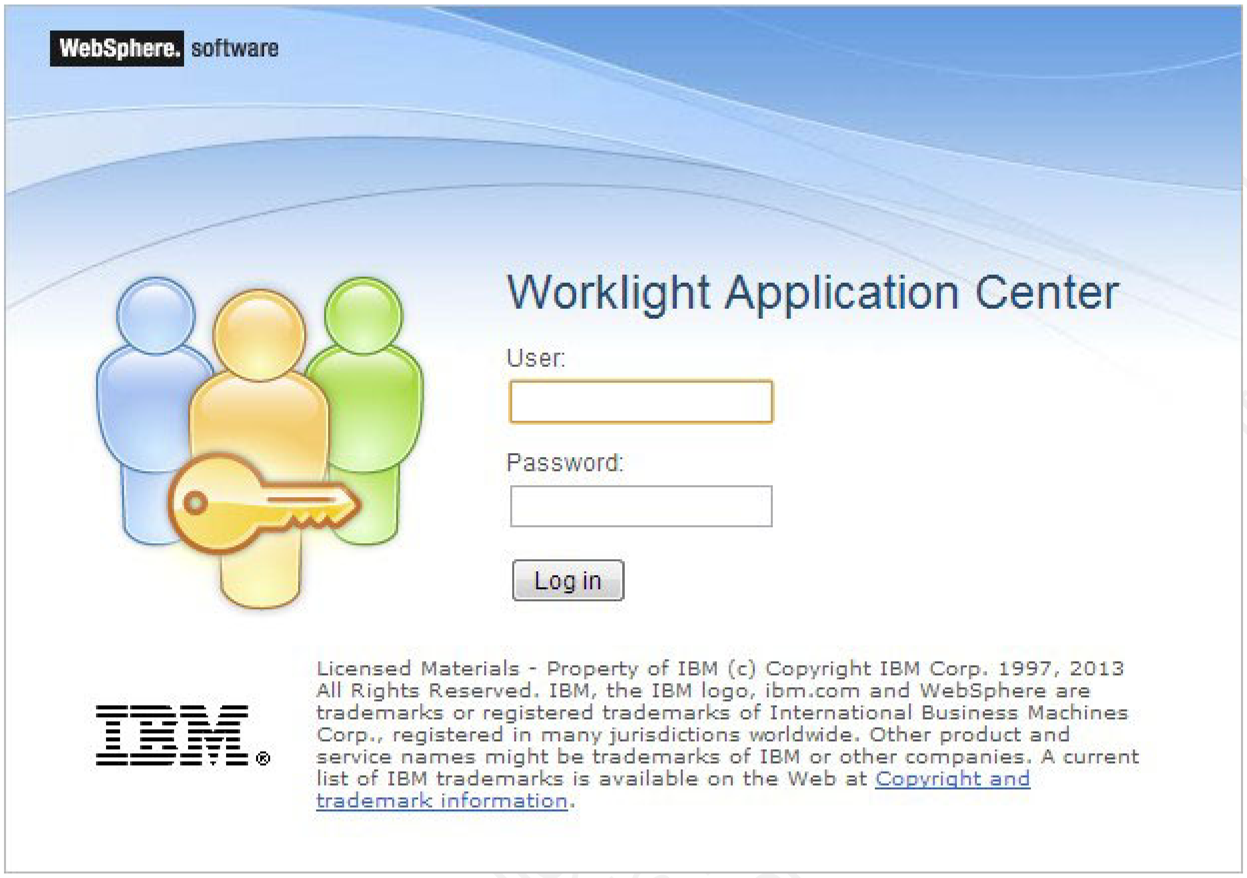
1. Start a web browser session on your desktop.
2. Contact your system administrator to obtain the address and port of the server where the Application Center is installed.
3. Enter the following URL: `http://server/appcenterconsole`

   Where *server* is the address and port of the server where the Application Center is installed.

   `http://localhost:9080/appcenterconsole`
4. Log in to the Application Center console

   Contact your system administrator to get your credentials so that you can log in to the Application Center console.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.



Licensed Materials - Property of IBM (c) Copyright IBM Corp. 1997, 2013 All Rights Reserved. IBM, the IBM logo, ibm.com and WebSphere are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. 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.

Figure 77. Login of the Application Center console

Note:

Only users with the administrator role can log in to the Application Center console.

Application Management

You can use Application Management to add new applications and versions and to manage those applications.

The Application Center enables you to add new applications and versions and to manage those applications.

Select **Applications** to access Application Management.

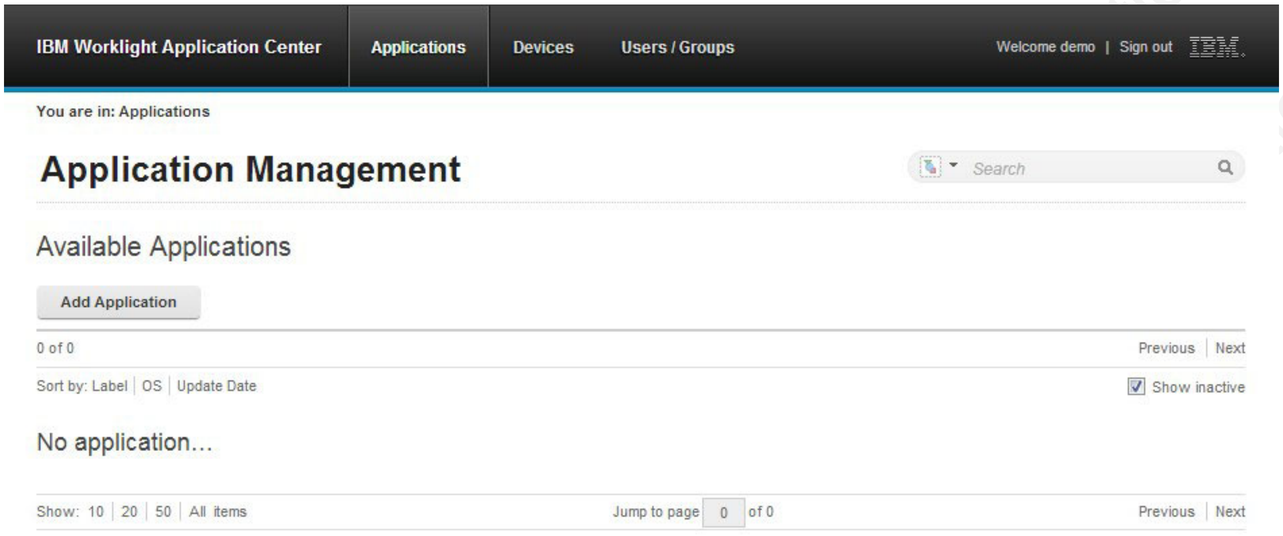
Application Center installed on WebSphere Application Server Liberty Profile or on Apache Tomcat

Installations of the Application Center on these application servers, during installation of Worklight with the Installation Manager package, have two different users defined that you can use to get started.

- User with login `demo` and password `demo`
- User with login `appcenteradmin` and password `admin`

WebSphere Application Server full profile

If you installed the Application Center on Websphere Application Server full profile, one user named `appcenteradmin` is created by default with the password indicated by the installer.



IBM Worklight Application Center | Applications | Devices | Users / Groups | Welcome demo | Sign out

You are in: Applications

Application Management

Search

Available Applications

Add Application

0 of 0 — Previous | Next

Sort by: Label | OS | Update Date — Show inactive

No application...

Show: 10 | 20 | 50 | All items — Jump to page 0 of 0 — Previous | Next

Figure 78. Empty application list

Adding a mobile application

Add applications to the repository on the server by using the Application Center console. These applications can then be installed on mobile devices by using the mobile client.

About this task

In the Applications view, you can add applications to the Application Center. Initially the list of applications is empty and you must upload an application file. Application files are described in this procedure.

Procedure

To add an application to make it available to be installed on mobile devices:

1. Click **Add Application**.
2. Click **Upload**.
3. Select the application file to upload to the Application Center repository.

   Android

   The application file extension is `apk`.

   iOS

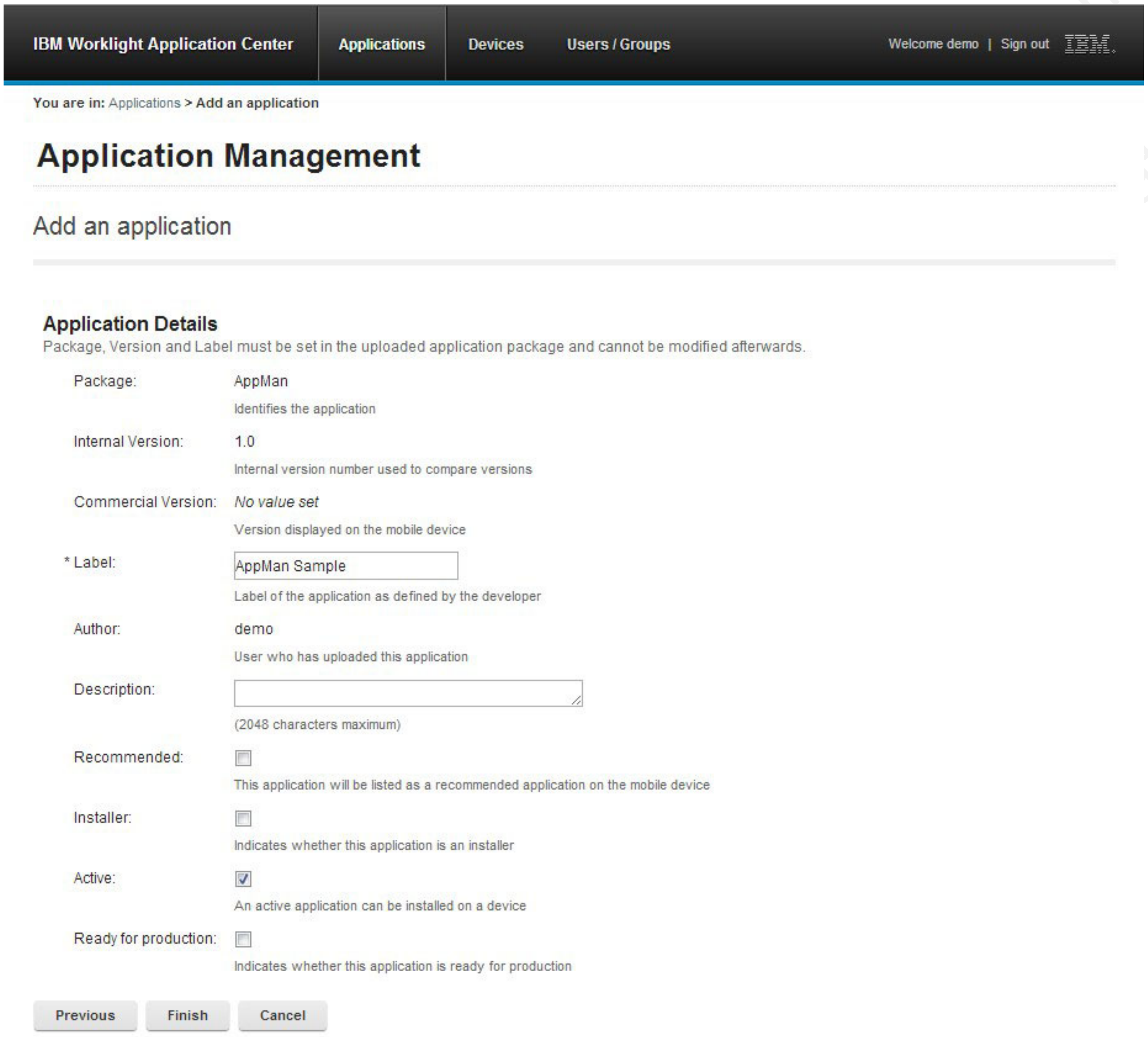
   The application file extension is `ipa`.

   BlackBerry

   The application file extension is `zip`. This archive file must contain a file with extension `jad` and all related files with extension `cod`.

Experimental IBM Worklight
This document is pro
In case of issues, refer to the online

4. Click **Next** to access the properties to complete the definition of the application.
5. Complete the properties to define the application. See Application properties for information about how to complete property values.
6. Click **Finish**.



You are in: Applications > Add an application

Application Management

Add an application

Application Details

Package, Version and Label must be set in the uploaded application package and cannot be modified afterwards.

Package:	AppMan Identifies the application
Internal Version:	1.0 Internal version number used to compare versions
Commercial Version:	*No value set* Version displayed on the mobile device
* Label:	AppMan Sample Label of the application as defined by the developer
Author:	demo User who has uploaded this application
Description:	 (2048 characters maximum)
Recommended:	☐ This application will be listed as a recommended application on the mobile device
Installer:	☐ Indicates whether this application is an installer
Active:	☑ An active application can be installed on a device
Ready for production:	☐ Indicates whether this application is ready for production

Figure 79. Application properties, adding an application

Application properties

Android applications and iOS applications have their own sets of properties that cannot be edited and common properties, and editable properties.

The values of the following fields are taken from the application and you cannot edit them.

- **`Package`**
- **`Internal Version`**
- **`Commercial Version`**

- **Label**

Properties of Android applications

- **Package** is the package name of the application; **package** attribute of the **manifest** element in the manifest file of the application. See the Android SDK documentation.
- **Internal Version** is the internal version identification of the application; **android:versionCode** attribute of the **manifest** element in the manifest file of the application. See the Android SDK documentation.
- **Commercial Version** is the published version of the application.
- **Label** is the label of the application; **android:label** attribute of the **application** element in the manifest file of the application. See the Android SDK documentation.

Properties of iOS applications

- **Package** is the company identifier and the product name; **CFBundleIdentifier** key. See the iOS SDK documentation.
- **Internal Version** is the build number of the application; **CFBundleVersion** key of the application. See the iOS SDK documentation.
- **Commercial Version** is the published version of the application.
- **Label** is the label of the application; **CFBundleDisplayName** key of the application. See the iOS SDK documentation.

Properties of BlackBerry applications

- **Package** is the name of the application project; **MIDlet-Name** entry of the `jad` file. See JSR-118 specification.
- **Internal Version** is the version of the application; **MIDlet-Version** entry of the `jad` file. See JSR-118 specification.
- **Commercial Version**, like **Internal Version**, is the version of the application.
- **Label** is the label of the application; **MIDlet-1** entry of the `jad` file. See JSR-118 specification. This property is optional. The label can be set or updated during the import of the application to the Application Center.
- **Vendor** is the vendor who created this application; **MIDlet-Vendor** entry of the `jad` file. See JSR-118 specification.

Common property

Author

The **Author** field is read only. It displays the user name of the user who uploads the application.

Editable properties

You can edit the following fields:

Description

Use this field to describe the application to the mobile user.

Recommended

Experimental IBM Worklight online documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Select **Recommended** to indicate that you recommend users to install this application. Recommended applications appear in a special list of recommended applications in the mobile client.

Installer

For the Administrator only: This property indicates that the application is used to install other applications on the mobile device and send feedback on an application from the mobile device to the Application Center. Usually only one application is qualified as **Installer** and is called the mobile client. This application is documented in "The mobile client" on page 473.

Active

Select **Active** to indicate that an application can be installed on a mobile device. If you do not select **Active**, the mobile user will not see the application in the list of available applications displayed on the device.

If you do not select **Active**, the application is inactive. In the list of available applications in Application Management, if **Show inactive** is selected, the application is disabled.

If **Show inactive** is not selected, the application does not appear in the list of available applications.

Ready for production

Select **Ready for production** to indicate that an application can be managed by the application store of Tivoli Endpoint Manager. Applications with this property selected are the only ones that are flagged to Tivoli Endpoint Manager. The property **Ready for production** indicates that an application is ready to be deployed in a production environment and is therefore suitable to be managed by Tivoli Endpoint Manager through its application store.

Editing application properties

You can edit the properties of an application in the list of uploaded applications.

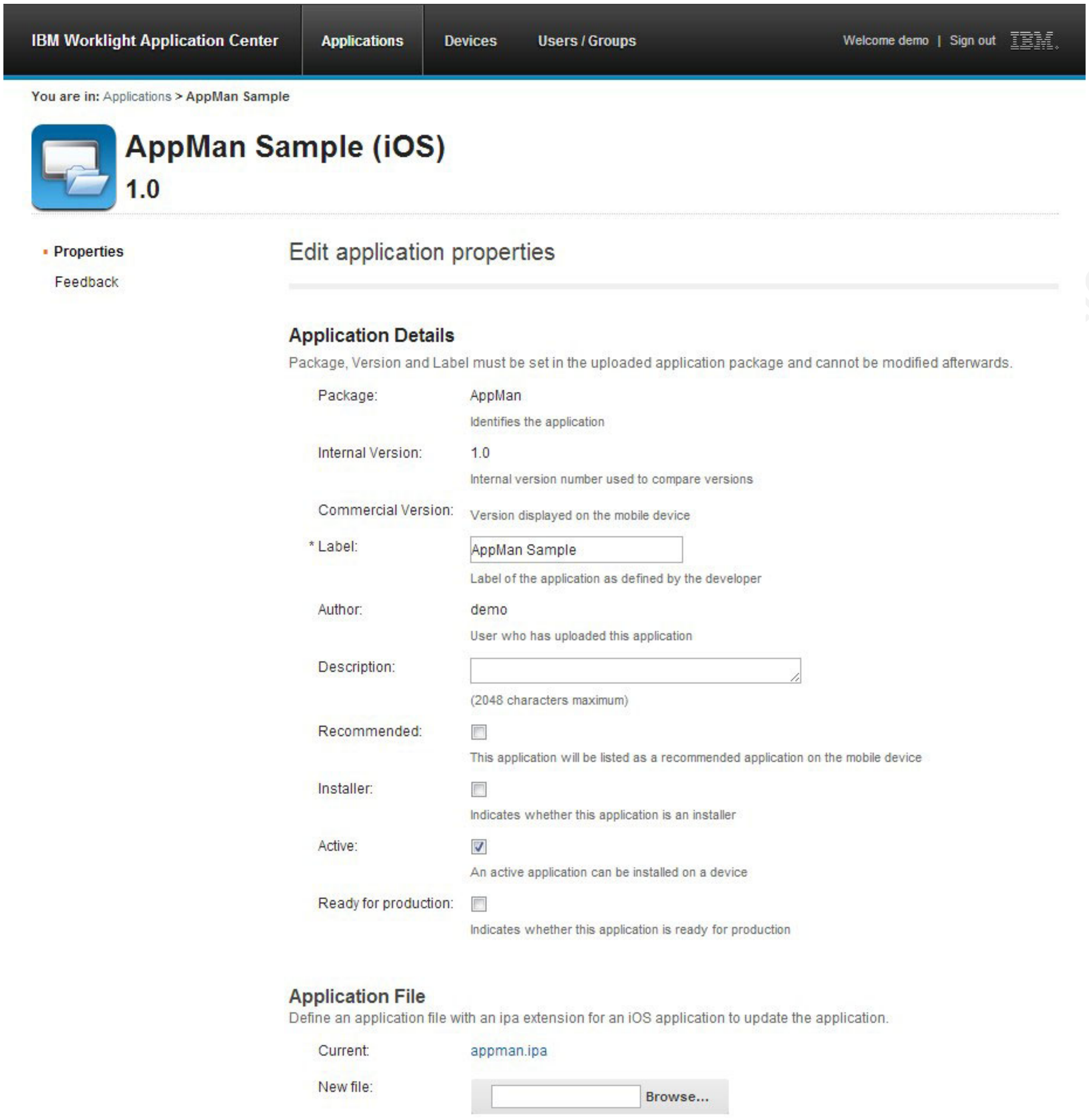
Procedure

To edit the properties of an uploaded application:

1. Select **Applications** to see the list of uploaded applications: Available Applications.
2. Click the version of the application to edit the properties: Application Details.
3. Edit any of the editable properties that you want. See "Application properties" on page 456 for details about these properties. The name of the current application file is shown below the properties.

   Important: If you want to update the file, it must belong to the same package and be the same version number. If either of these properties is not the same you must go back to the application list and add the new version first.
4. Click **OK** to save your changes and return to Available Applications or **Apply** to save and keep Application Details open.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.



Edit application properties

Application Details

Package, Version and Label must be set in the uploaded application package and cannot be modified afterwards.

Package:	AppMan Identifies the application
Internal Version:	1.0 Internal version number used to compare versions
Commercial Version:	Version displayed on the mobile device
* Label:	AppMan Sample Label of the application as defined by the developer
Author:	demo User who has uploaded this application
Description:	(2048 characters maximum)
Recommended:	☐ This application will be listed as a recommended application on the mobile device
Installer:	☐ Indicates whether this application is an installer
Active:	☑ An active application can be installed on a device
Ready for production:	☐ Indicates whether this application is ready for production

Application File

Define an application file with an ipa extension for an iOS application to update the application.

Current:	appman.ipa
New file:	

Figure 80. Application properties for editing

Downloading an application file

You can download the file of an application registered in the Application Center.

Procedure

1. Select **Applications** to see the list of uploaded applications: **Available Applications**.
2. Tap the version of the application under **Application Details**.
3. Tap the file name in the "Application File" section.

Viewing application feedback

In the Application Center console, you can see feedback about mobile applications sent by users.

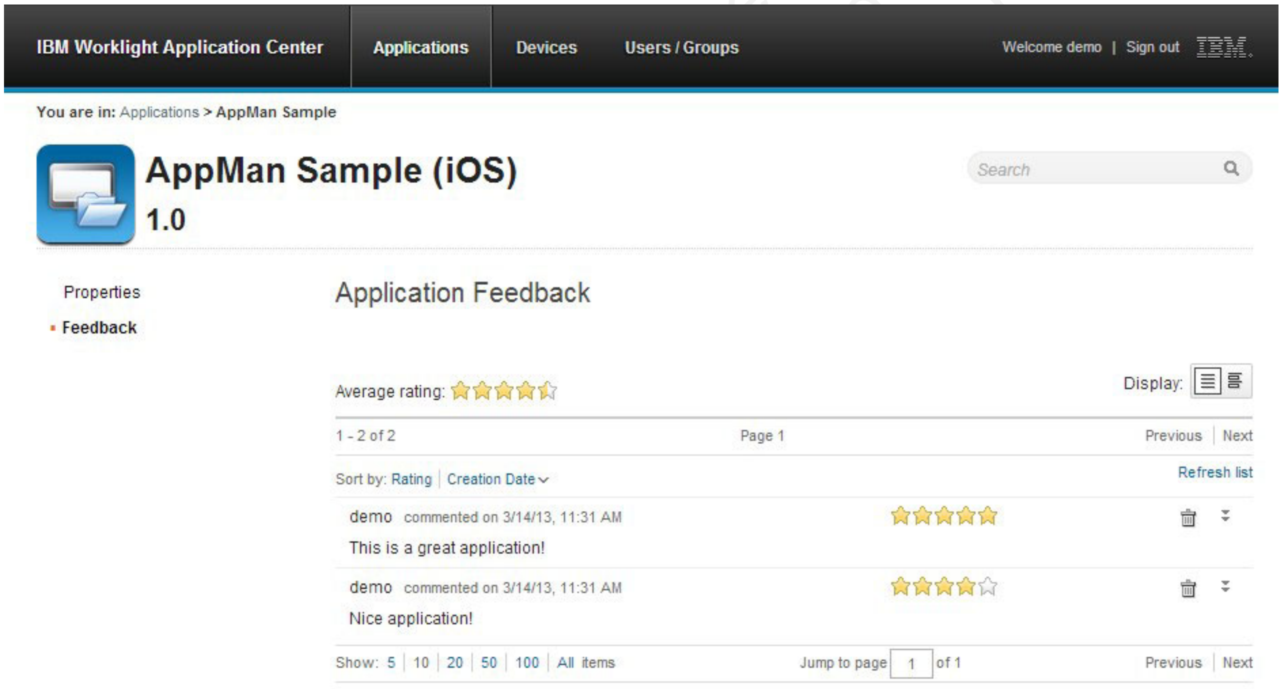
About this task

Users of mobile applications can rate an application and send feedback through the Application Center mobile client. The feedback is available in the Application Center console. Individual feedback is always associated with a particular version of an application.

Procedure

To view feedback from mobile users or testers about an application:

1. Select **Applications** to see the list of uploaded applications: Available Applications.
2. Select the version of the application.
3. Click the feedback rating.



AppMan Sample (iOS)
1.0

Application Feedback

Average rating: ★★★★☆

1 - 2 of 2 Page 1

Sort by: Rating | Creation Date

demo commented on 3/14/13, 11:31 AM ★★★★★
This is a great application!

demo commented on 3/14/13, 11:31 AM ★★★★☆
Nice application!

Show: 5 | 10 | 20 | 50 | 100 | All items Jump to page 1 of 1

Figure 81. Application feedback

The rating is an average of all recorded feedback. It consists of one to five stars, where one star represents the lowest level of appreciation and five stars represents the highest level of appreciation. If no stars are selected, no feedback is recorded.

The average rating gives an indication of how the application satisfies the intended use of the application.

4. Click the two arrow heads on the right to expand the comment that provides the feedback and to view the details of the mobile device where the feedback is generated.

For example, the comment can give the reason for feedback, such as failure to install.

If you want to delete the feedback, click the trash can on the right.

User and group management

You can use users and groups to define who has access to some features of the Application Center, such as installing applications on mobile devices.

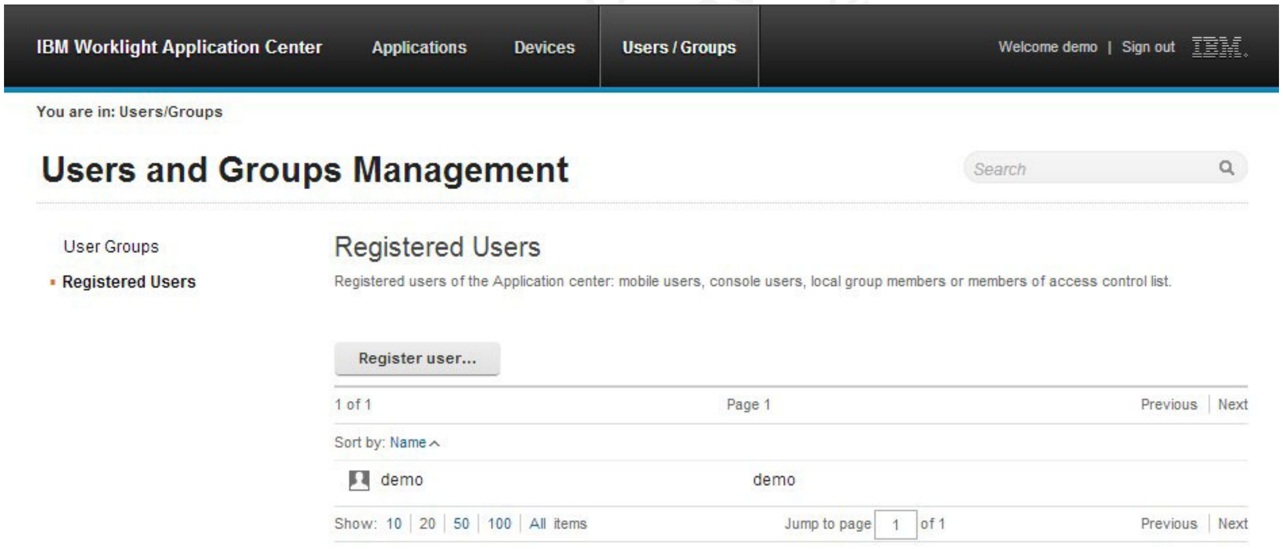
Purpose

Use users and groups in the definition of access control lists (ACL).

Managing registered users

You manage registered users by clicking the **Users/Groups** tab and selecting **Registered users**. You obtain a list of registered users of the Application Center that includes:

- Mobile client users
- Console users
- Local group members
- Members of an access control list



Registered Users

Registered users of the Application center: mobile users, console users, local group members or members of access control list.

1 of 1 Page 1

demo demo

Figure 82. List of registered users of the Application Center

If the Application Center is connected to an LDAP repository, you cannot edit the user display names. If the repository is not LDAP, you can change a user display name by selecting it and editing it.

You can register new users by clicking **Register User**, entering the login name and the display name, and clicking **OK**.

To unregister a user, click the trash icon next to the user name.

Unregistering a user from the Application Center has the effect of:

- Removing feedback given by the user
- Removing the user from the access control lists
- Removing the user from local groups

Note:

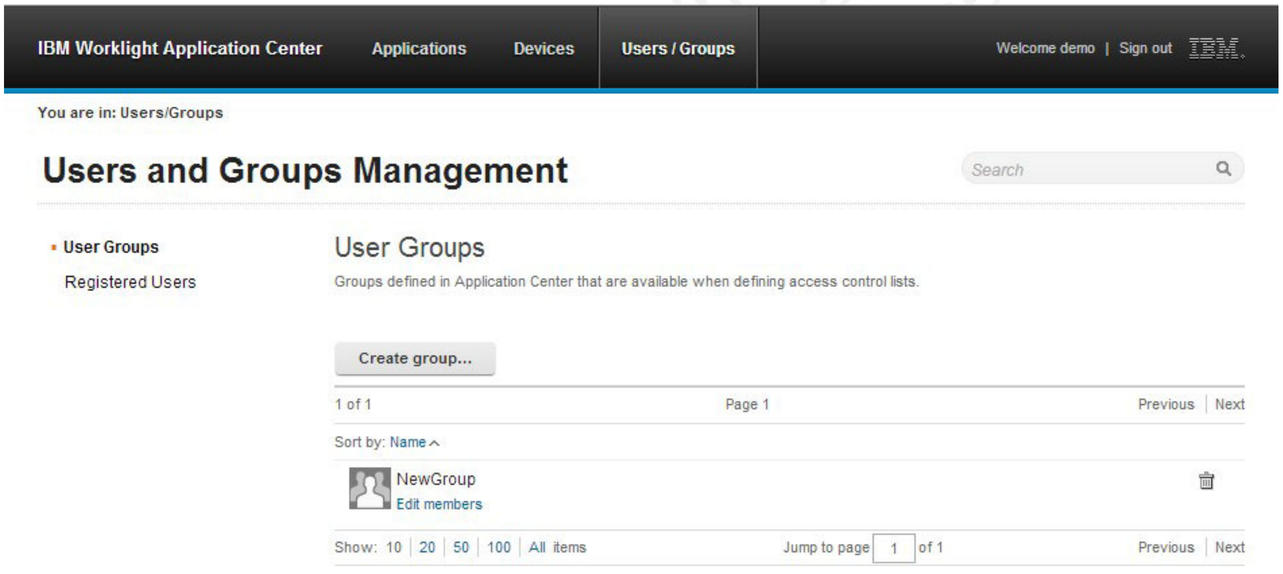
When you unregister a user, the user is not removed from the application server or the LDAP repository.

Managing local groups

You manage local groups by clicking the **Users/Groups** tab and selecting **User group**.

To create a local group, click **Create group**. Enter the name of the new group and click **OK**.

If the Application Center is connected to an LDAP repository, the search includes local groups as well as the groups defined in the LDAP repository. If the repository is not LDAP, only local groups are available to the search.



Groups defined in Application Center that are available when defining access control lists.

Figure 83. Local user groups

To delete a group, click the trash icon next to the group name. The group is also removed from the access control lists.

To add or remove members of a group, click the **Edit members** link of the group.

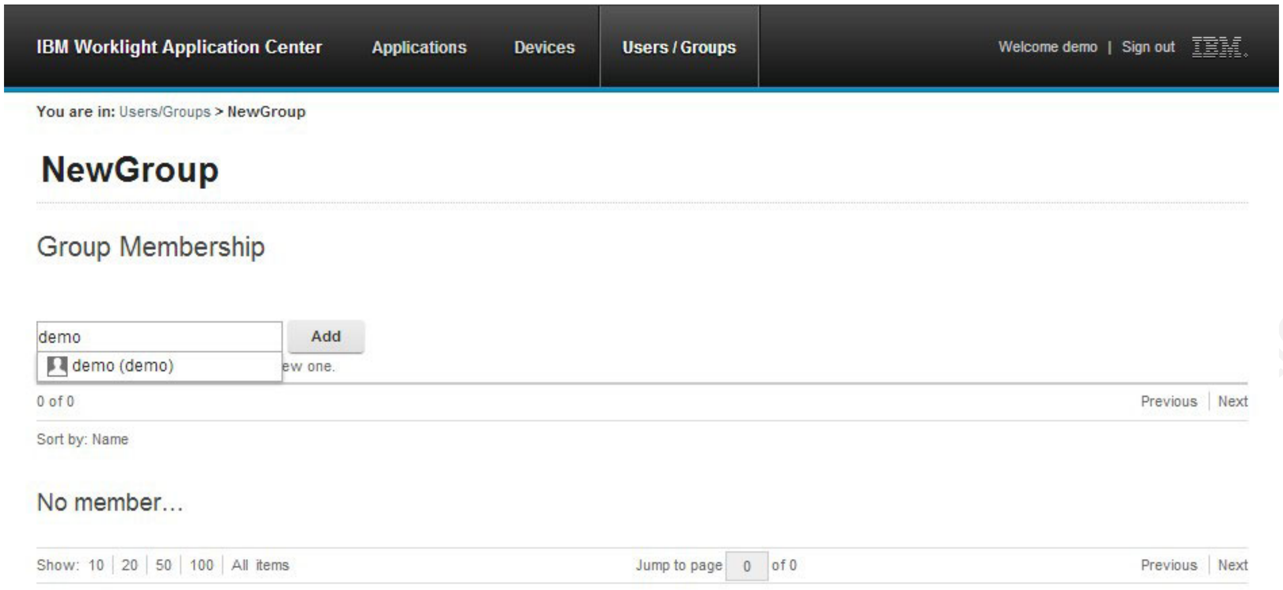


Figure 84. Managing group membership

To add a new member, search for the user by entering the user display name, select the user, and click **Add**.

If the Application Center is connected to an LDAP repository, the search for the user is performed in the LDAP repository. If the repository is not LDAP, the search is performed in the list of registered users.

To remove a member from a group, click the cross on the right of the user name.

Access control

You can decide whether installation of an application on mobile devices is open to any users or whether you want to restrict the ability to install an application.

Installation of applications on a mobile device can be limited to specific users or available to any users.

Access control is defined at the application level and not at the version level.

By default, after an application is uploaded, any user has the right to install the application on a mobile device.

The current access control for an application is displayed in Available Applications for each application. The unrestricted or restricted access status for installation is shown as a link to the page for editing access control.

Installation rights are only about the installation of the application on the mobile device. If access control is not enabled, everybody has access to the application.

Managing access control

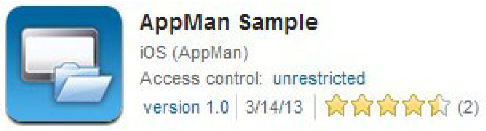
You can add or remove access for users or groups to install an application on mobile devices.

Experimental IBM Worklight online documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

Procedure

You can edit access control:

1. In Application Management under Available Applications, click the **unrestricted** or **restricted** state of Installation of an application.


AppMan Sample
iOS (AppMan)
Access control: unrestricted
version 1.0 | 3/14/13 | ★★★★☆ (2)

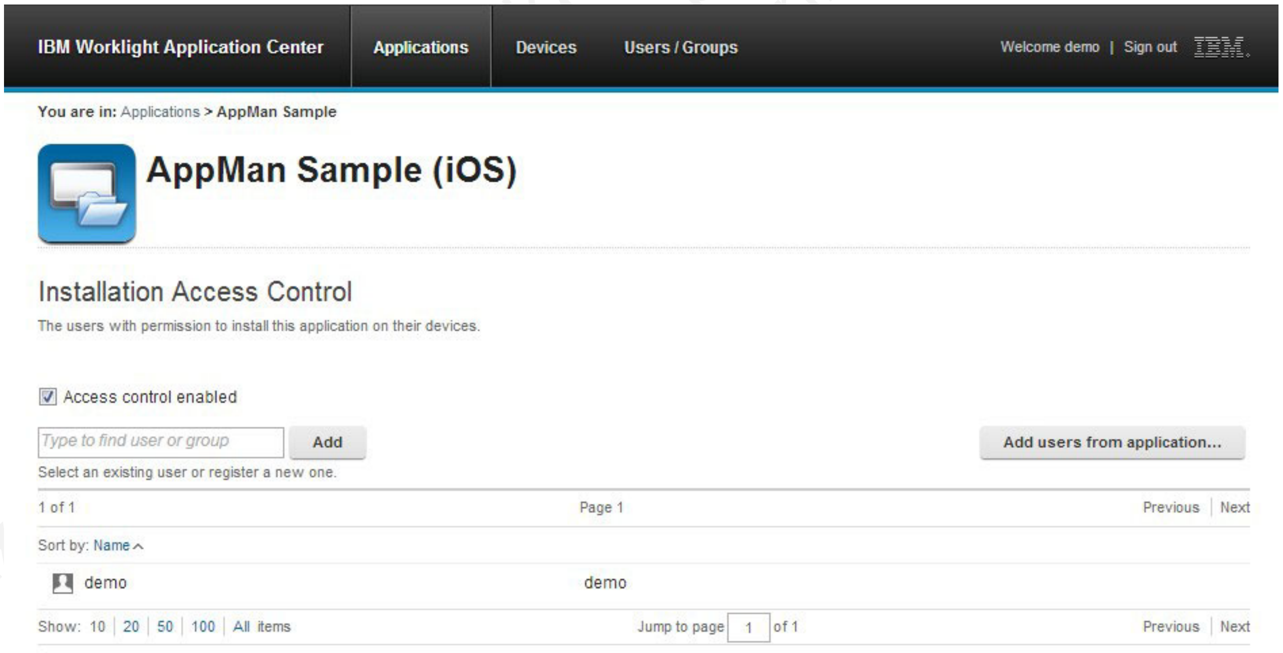
2. Select **Access control enabled** to enable access control.
3. Add users or groups to the access list.

   To add a single user or group, enter a name, select the entry in the matching entries found, and click **Add**.

   If the Application Center is connected to an LDAP repository, you can search for users and groups in the repository as well as locally defined groups. If the repository is not LDAP, you can search only local groups and registered users. Local groups are exclusively defined in the **Users/Groups** tab.

   You can register a user at the same time as adding the user to the access list by entering the name and clicking **Add**. Then you must specify the login name and the display name of the user.

   To add all the users of an application, click **Add users from application** and select the appropriate application.


IBM Worklight Application Center | Applications | Devices | Users / Groups | Welcome demo | Sign out

You are in: Applications > AppMan Sample

AppMan Sample (iOS)

Installation Access Control

The users with permission to install this application on their devices.

☑ Access control enabled

Type to find user or group [Add] [Add users from application...]

Select an existing user or register a new one.

1 of 1 Page 1 Previous | Next

Sort by: Name

demo demo

Show: 10 | 20 | 50 | 100 | All items Jump to page 1 of 1 Previous | Next

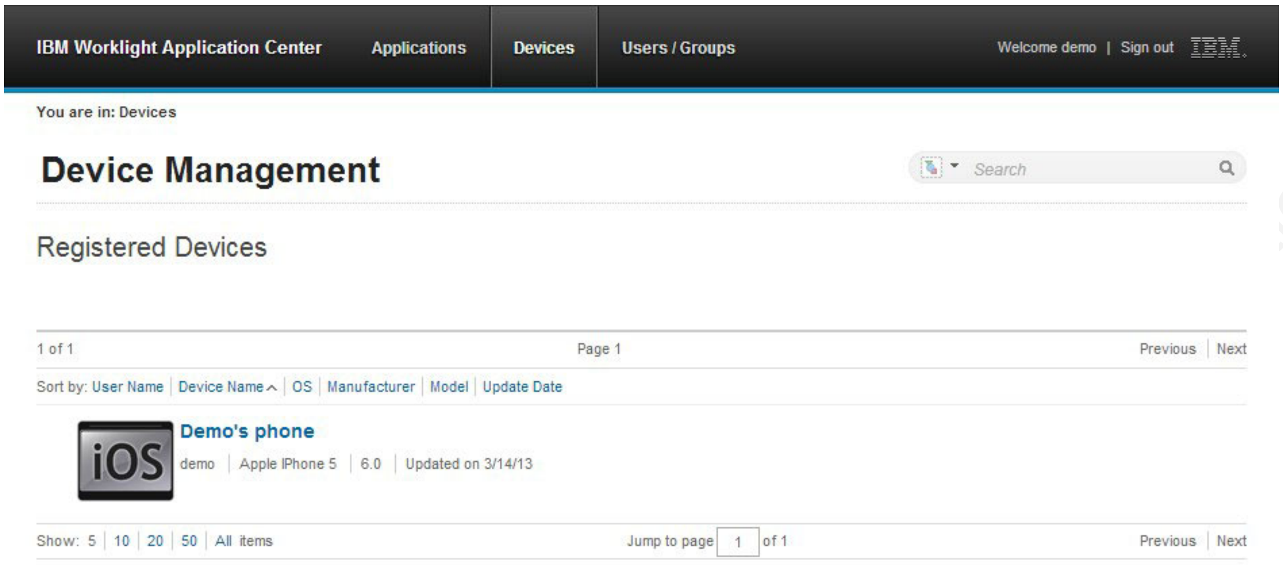
Figure 85. Adding users to the access list

To remove access from a user or group, click the cross on the right of the name.

Device Management

You can see the devices that connected to the Application Center from the Application Center mobile client and their properties.

Device Management shows under **Registered Devices** the list of devices that have connected to the Application Center at least once from the Application Center mobile client.



Device Management

Registered Devices

1 of 1 Page 1 Previous | Next

Sort by: User Name | Device Name | OS | Manufacturer | Model | Update Date

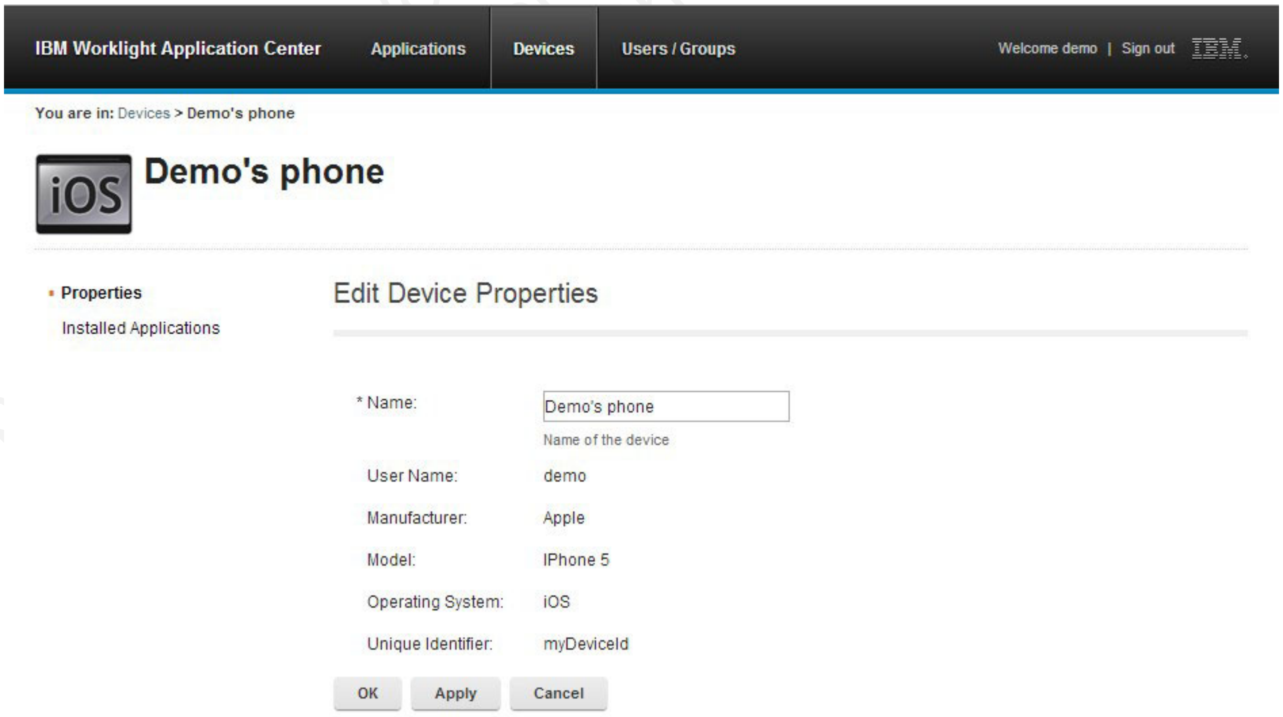
Demo's phone
demo | Apple IPhone 5 | 6.0 | Updated on 3/14/13

Show: 5 | 10 | 20 | 50 | All items Jump to page 1 of 1 Previous | Next

Figure 86. The device list

Device properties

Click a device in the list of devices to view the properties of the device or the applications installed on that device.



Demo's phone

Edit Device Properties

* Name:	Demo's phone Name of the device
User Name:	demo
Manufacturer:	Apple
Model:	IPhone 5
Operating System:	iOS
Unique Identifier:	myDeviceId

Figure 87. Device properties

Select **Properties** to view the device properties.

Name

The name of the device. You can edit this property.

Note: on iOS, the user can define this name in the settings of the device in **Settings** > **General** > **Information** > **Name**. The same name is displayed on iTunes.

User Name

The name of the first user who logged into the device.

Manufacturer

The manufacturer of the device.

Model

The model identifier.

Operating System

The operating system of the mobile device.

Unique identifier

The unique identifier of the mobile device.

If you edit the device name, click **OK** to save the name and return to Registered Devices or **Apply** to save and keep Edit Device Properties open.

Applications installed on device

Select **Applications installed on device** to list all the applications installed on the device.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

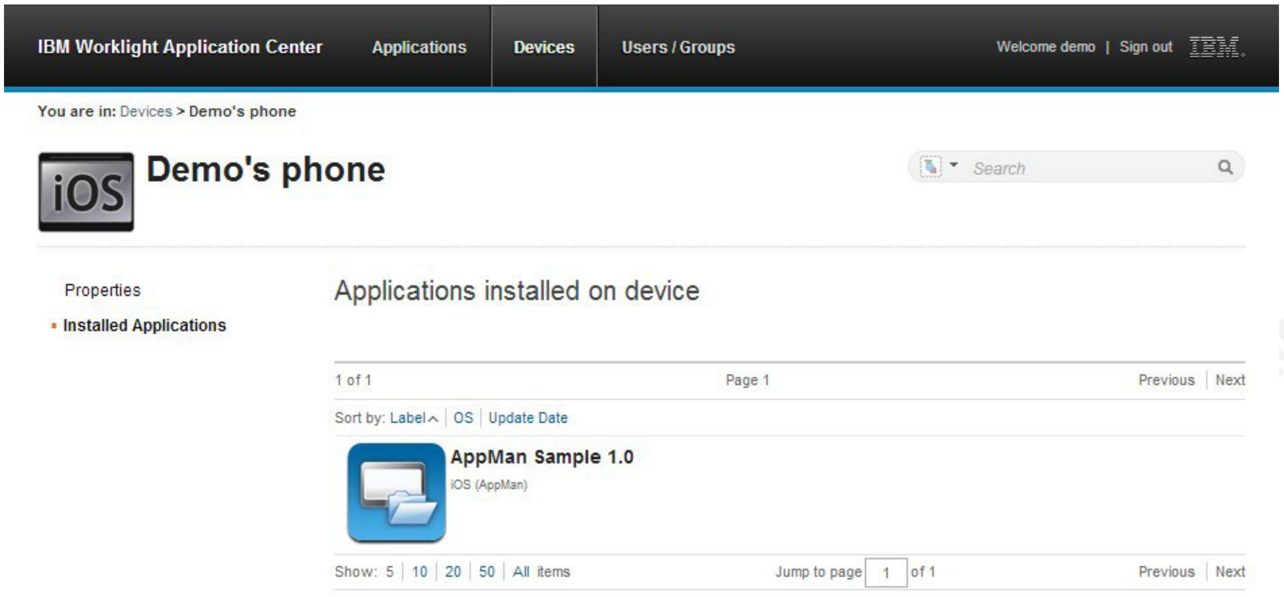


Figure 88. Applications installed on a device

Signing out of the Application Center console

For security purposes, you must sign out of the console when you have finished your administrative tasks.

Purpose

To log out of the secure sign-on to the Application Center console..

To sign out of the Application Center console, click **Sign out** next to the Welcome message that is displayed in the banner of every page.

Command-line tool for uploading an application

To deploy applications to the Application Center through a build process, use the command-line tool.

You can upload an application to the Application Center by using the web interface of the Application Center console. You can also upload a new application by using a command-line tool.

This is particularly useful when you want to incorporate the deployment of an application to the Application Center into a build process. This tool is located at:

installDir/ApplicationCenter/tools/applicationcenterdeploytool.jar

The tool can be used for application files with extension APK or IPA. It can be used stand alone or as an ant task.

The `tools` directory contains all the files required to support the use of the tool.

- `applicationcenterdeploytool.jar`: the upload tool.
- `json4j.jar`: the library for the JSON format required by the upload tool.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

- `build.xml`: a sample ant script that you can use to upload a single file or a sequence of files to the Application Center.

Using the stand-alone tool to upload an application

To upload an application, call the stand-alone tool from the command line.

Procedure

Use the stand-alone tool by following these steps.

1. Add `applicationcenterdeploytool.jar` and `json4j.jar` to the java classpath environment variable.
2. Call the upload tool from the command line:

   ```
   java com.ibm.appcenter.Upload [options] [files]
   ```

   You can pass any of the available options in the command line.

Option	Content indicated by	Description
-s	**serverpath**	The path to the Application Center server.
-c	**context**	The context of the Application Center web application.
-u	**user**	The user credentials to access the Application Center.
-p	**password**	The password of the user.
-f		Force uploading of applications, even if they exist already.

   The **files** parameter can specify files of type Android application package (.apk) files or iOS application (.ipa) files.

   In this example user `demo` has the password `demopassword`. Use this command line.

   ```
   java com.ibm.appcenter.Upload -s http://localhost:8080 -c applicationcenter -u demo -p demopasswo
   ```

Ant task for uploading an application

You can use the upload tool as an Ant task and use the Ant task in your own Ant script.

When you use the upload tool as an ant task, the **classname** of the ant task is `com.ibm.appcenter.ant.UploadApps`.

Parameters of ant task	Description
serverPath	To connect to the Application Center. The default value is `http://localhost:8080`.
context	The context of the Application Center. The default value is `/applicationcenter`.
loginUser	The user name with permissions to upload an application.
loginPass	The password of the user with permissions to upload an application.

Experimental IBM Worklight offline user documentation.
This document is provided "as-is".
In case of issues, refer to the online user documentation.

forceOverwrite	If set to true, the ant task attempts to overwrite applications in the Application Center when it uploads an application that is already present.
file	The `.apk` or `.ipa` file to be uploaded. This parameter has no default value.
fileset	To upload multiple files.

Example

This example shows how to use the ant task in your own ant script.

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="PureMeapAntDeployTask" basedir="." default="upload.AllApps">

 <property name="install.dir" value="../../" />
  <property name="workspace.root" value="../../" />

<!-- Server Properties -->
  <property name="server.path" value="http://localhost:9080/" />
 <property name="context.path" value="applicationcenter" />
 <property name="upload.file" value="" />
 <property name="force" value="true" />

 <!--  Authentication Properties -->
  <property name="login.user" value="appcenteradmin" />
  <property name="login.pass" value="admin" />
  <path id="classpath.run">
    <fileset dir="${install.dir}/ApplicationCenter/tools/">
      <include name="applicationcenterdeploytool.jar" />
      <include name="json4j.jar"/>
   </fileset>
  </path>
 <target name="upload.init">
  <taskdef name="uploadapps" classname="com.ibm.appcenter.ant.UploadApps">
   <classpath refid="classpath.run" />
  </taskdef>
 </target>
 <target name="upload.App" description="Uploads a single application" depends="upload.init">
  <uploadapps serverPath="${server.path}"
   context="${context.path}"
   loginUser="${login.user}"
   loginPass="${login.pass}"
   forceOverwrite="${force}"
   file="${upload.file}"
  </target>
  <target name="upload.AllApps" description="Uploads all found APK and IPA files" depends="upload.
  <uploadapps serverPath="${server.path}"
     loginUser="${login.user}"
   loginPass="${login.pass}"
   forceOverwrite="${force}"
    context="${context.path}"
   <fileset dir="${workspace.root}">
     <include name="**/*.ipa" />
    <include name="**/*.apk" />
   </fileset>
  </uploadapps>
 </target>
</project>
```

This sample ant script is in the `tools` directory. You can use it to upload a single application to the Application Center.

```
ant upload.App -Dupload.file=sample.apk
```

Experimental IBM Worklight online user documentation.
This document is provided "as is".
In case of issue, refer to the online user documentation.

You can also use it to upload all applications found in a directory hierarchy.

```
ant upload.AllApps -Dworkspace.root=myDirectory
```

Properties of the sample ant script

Property	Comment
install.dir	Defaults to `../../`
server.path	The default value is `http://localhost:9080`.
context.path	The default value is `applicationcenter`.
upload.file	This property has no default value. It must include the exact file path.
workspace.root	Defaults to `../../`
login.user	The default value is `appcenteradmin`.
login.pass	The default value is `admin`.
force	The default value is `true`..

To specify these parameters by command line when you call ant, add `-D` before the property name. For example:

```
-Dserver.path=http://localhost:8888/
```

Publishing Worklight applications to the Application Center

You can use the application management plug-in to publish native applications to the IBM Application Center.

About this task

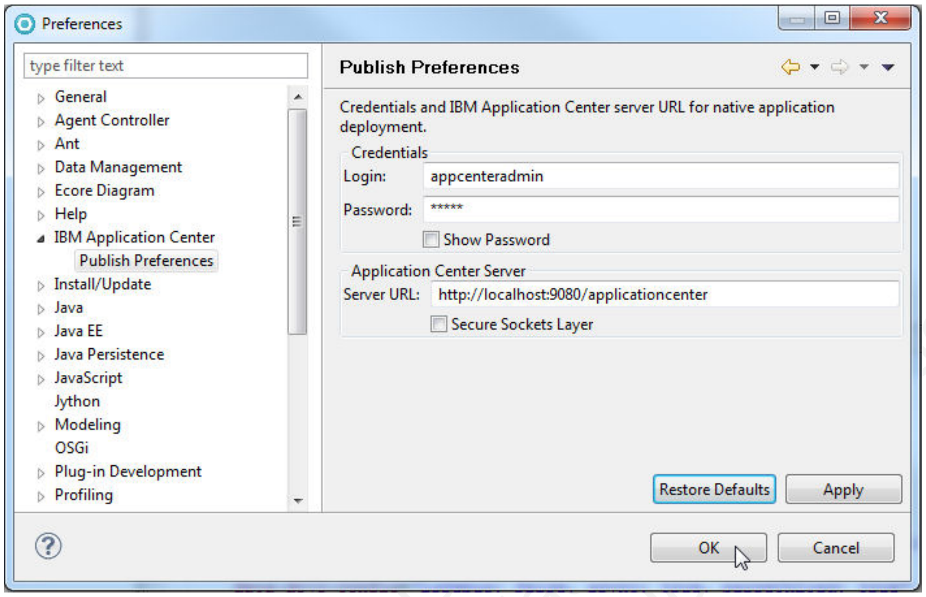
You can deploy applications for Android and iOS operating systems to the Application Center directly from the IBM Worklight Studio IDE. In Worklight Studio, you can deploy Android application package (.apk) files and iOS application (.ipa) files that you choose from your file system. You can right-click an application (.apk or .ipa) file to deploy it to the Application Center.

Procedure

To publish an application to the Application Center, complete the following steps:

1. Specify the publish preferences for the Application Center:
   a. In the main menu, click **Window** > **Preferences**.
   b. Expand **IBM Application Center** > **Publish Preferences**.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.


Preferences

type filter text

- General
- Agent Controller
- Ant
- Data Management
- Ecore Diagram
- Help
- IBM Application Center
  - Publish Preferences
- Install/Update
- Java
- Java EE
- Java Persistence
- JavaScript
- Jython
- Modeling
- OSGi
- Plug-in Development
- Profiling

Publish Preferences

Credentials and IBM Application Center server URL for native application deployment.

Credentials

Login: appcenteradmin

Password: *****

Show Password

Application Center Server

Server URL: http://localhost:9080/applicationcenter

Secure Sockets Layer

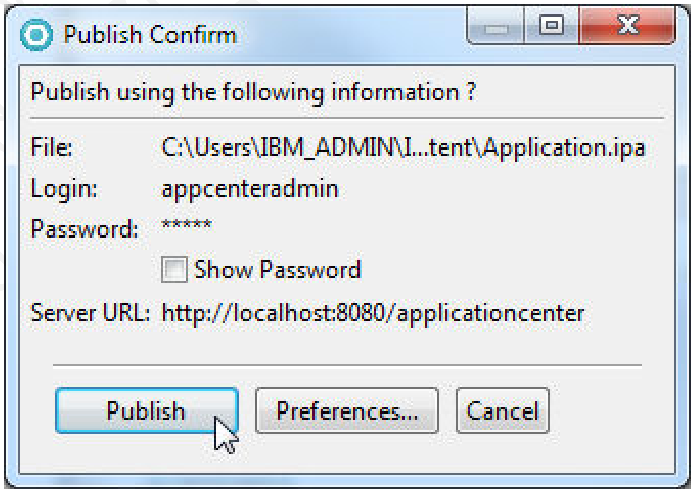
Restore Defaults | Apply

OK | Cancel

c. Specify the default publish preference settings for the Application Center:

Preference	Description
Credentials	Specify the login and password required to access the application repository.
Application Center Server	Specify the URL of the application center server to use when publishing applications.

2. Publish an application (.apk or .ipa file) from a Worklight project:
   a. Right-click the application and click **IBM Application Center** > **Publish on IBM Application Center**. The Publish Confirm dialog opens.


Publish Confirm

Publish using the following information ?

File: C:\Users\IBM_ADMIN\I...tent\Application.ipa

Login: appcenteradmin

Password: *****

Show Password

Server URL: http://localhost:8080/applicationcenter

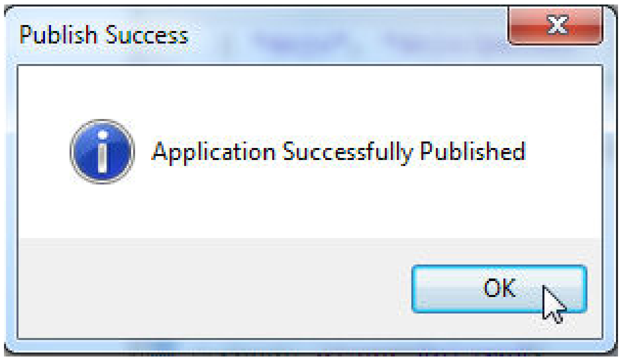
Publish | Preferences... | Cancel

   b. In the Publish Confirm dialog, choose one of the following options:

Option	Description
Publish the application by using the current preferences.	Click **Publish**.

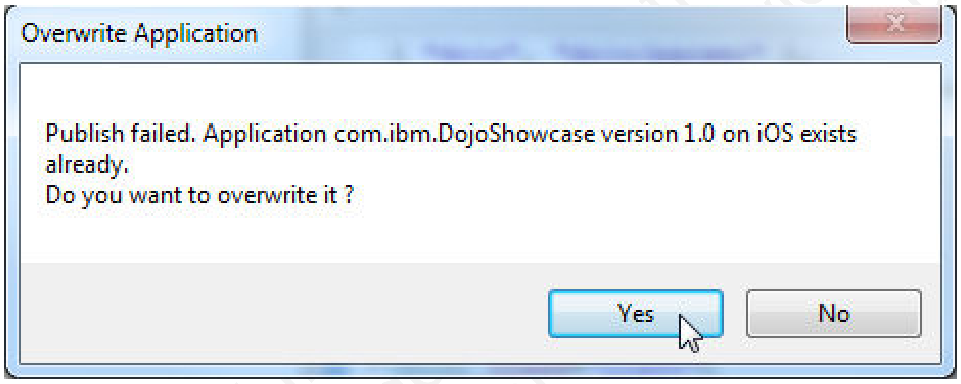
Option	Description
Change any of the preferences before publishing the application.	Click **Preferences** to open the Publish Preferences page and edit the preference settings.

You receive confirmation when publication is successful.


Publish Success

Application Successfully Published

If the application already exists, publication will fail. You are given the option to overwrite the existing version of the application.

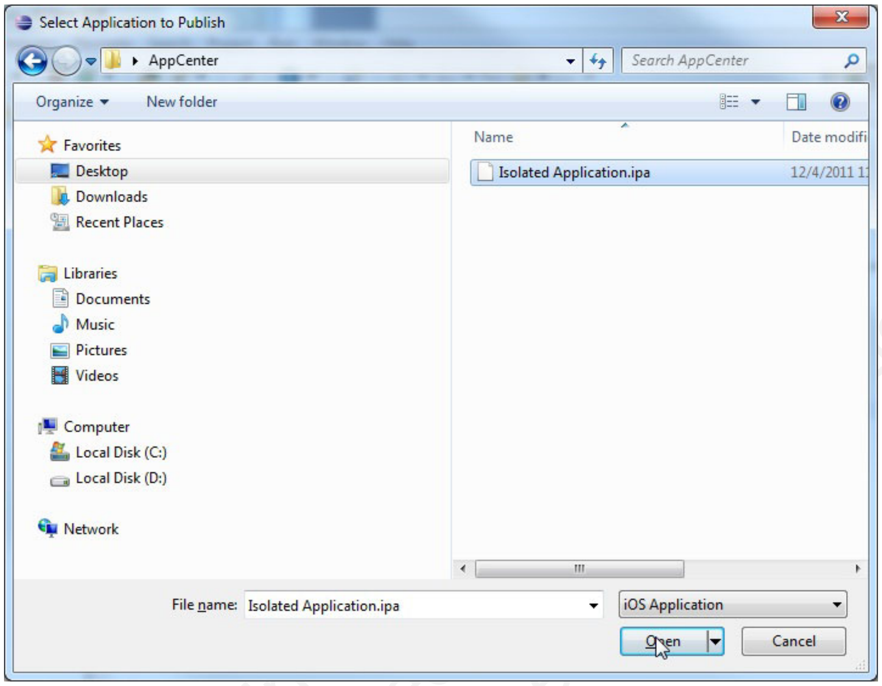

Overwrite Application

Publish failed. Application com.ibm.DojoShowcase version 1.0 on iOS exists already.
Do you want to overwrite it ?

Tip: To publish an application that is not part of the Worklight project:

1) Right-click the Worklight project and click **IBM Application Center** > **Publish on IBM Application Center**. The Select Application to Publish window opens.

Experimental offline user documentation
This documentation is provided "as is".
In case of issues, refer to online user documentation.



File name: Isolated Application.ipa

iOS Application

2) Navigate to the application (.apk or .ipa) file that you want to publish and click **Open** to open the Publish Confirm dialog.

The mobile client

You can install applications on your mobile device with the Application Center mobile client.

The Application Center mobile client is the application that runs on your Android, iOS, or BlackBerry device. (Only BlackBerry OS V6 and OS V7 are supported by the current version of the Application Center.) You use the mobile client to list the catalog of available applications in the Application Center. You can install these applications on your device. The mobile client is sometimes referred to as the Application Center installer. This application must be present on your device if you want to install on your device applications from your private application repository.

Prerequisites

Your system administrator must give you a user name and password before you can download and install the mobile client. This user name and password is required whenever you start the mobile client on your device. For security reasons, do not disseminate these credentials. These credentials are the same credentials used to log in to the Application Center console.

Installing the client on an Android mobile device

You can install the mobile client on your Android mobile device by entering the access URL in your browser, entering your credentials, and completing the required steps.

Procedure

1. Start the browser on your mobile device.

Experimental IBM Worklight online
This document is provided
In case of issues, refer to the online version.

2. Enter the following access URL in the address text field: `http://`*`hostname`*`:`*`portnumber`*`/appcenterconsole/installers.html`

   Where *hostname* is the address of the server and *portnumber* is the number of the port where the Application Center is installed. Your system administrator can provide this information.

The Android browser is not able to run pages when SSL communication and self-signed certificates are used. In this case, you must use a non self-signed certificate or use another browser on the Android device, such as Firefox, Chrome, or Opera.

3. Enter your user name and password. See Prerequisites in "The mobile client" on page 473.

   When your user name and password are validated, the list of compatible installer applications for your device is displayed in the browser. Normally, only one application, the mobile client, appears in this list.

Before you can see the mobile client in the list of available applications, the Application Center administrator must install the mobile client application. The administrator uploads the mobile client to the Application Center and sets the **`Installer`** property to **`true`**. See "Application properties" on page 456.

Experimental IBM Worklight offline user documentation
This document is provided "as-is"
In case of issues, refer to the online user documentation.

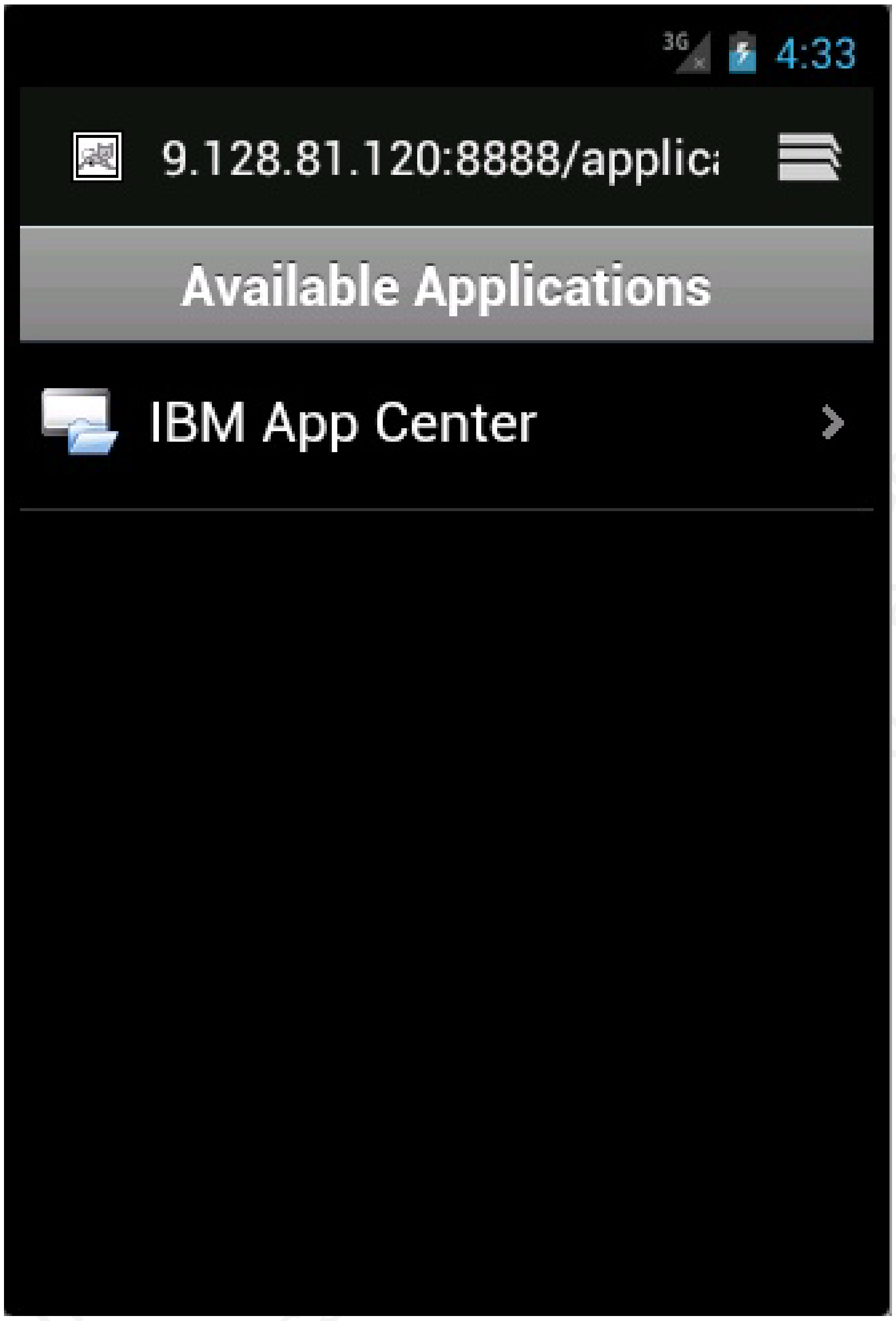
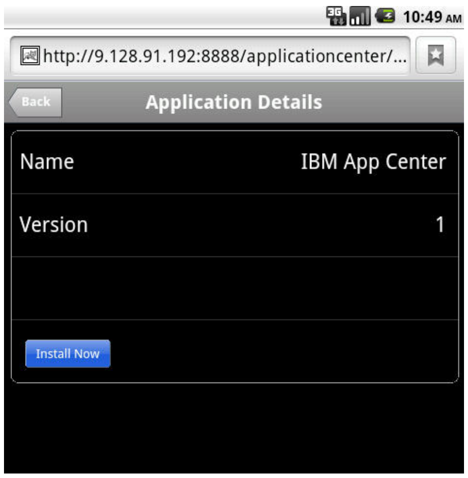


Figure 89. List of available mobile client applications to install

4. Select an item in the list to display the application details.
   Typically, these details include the application name and its version number.

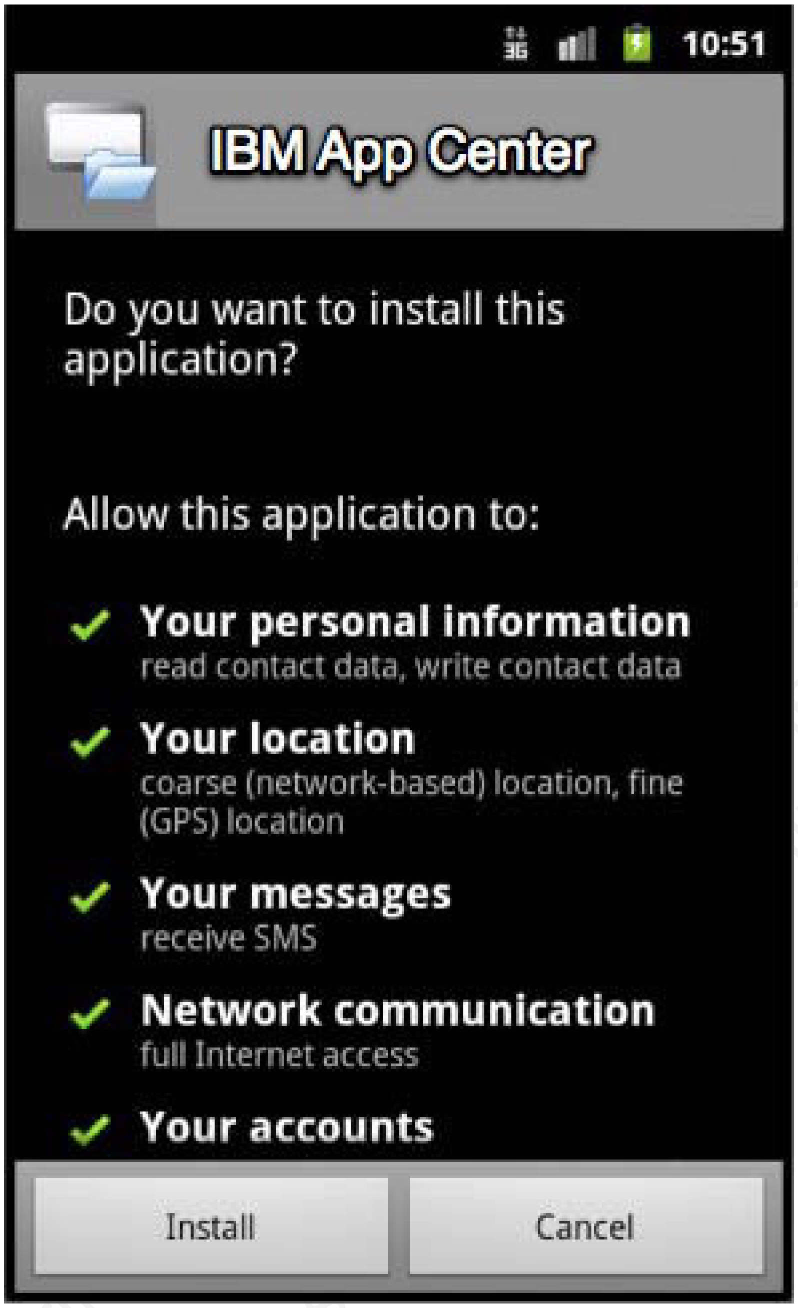


Name	IBM App Center
Version	1

Figure 90. Application details

5. Tap **Install Now** to download the mobile client.
6. Launch the **Android Download** applications.
7. Select the Application Center client installer.
   You can see the access granted to the application when you choose to install it.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.



Do you want to install this application?

Allow this application to:

✓ **Your personal information**
read contact data, write contact data

✓ **Your location**
coarse (network-based) location, fine (GPS) location

✓ **Your messages**
receive SMS

✓ **Network communication**
full Internet access

✓ **Your accounts**

Figure 91. Installation of the mobile client on Android

8. Select **Install** to install the mobile client.
9. When the application is installed, select **Open** to open the mobile client or **Done** to close the Downloads application.

Installing the client on an iOS mobile device

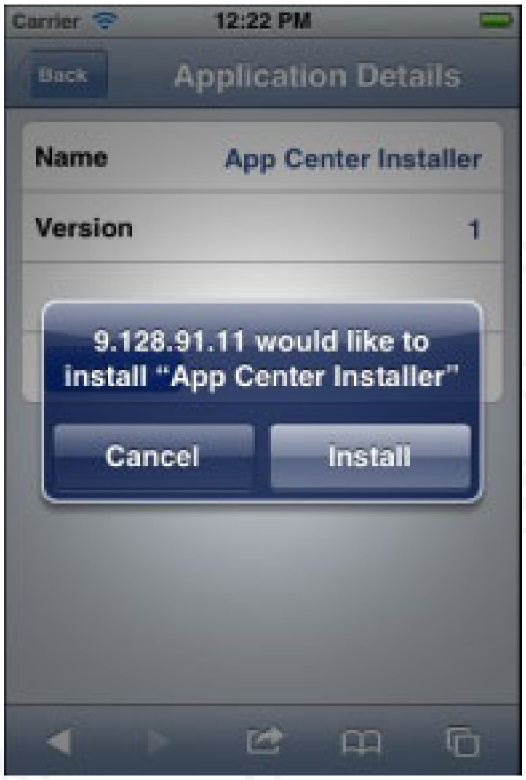
You can install the mobile client on your iOS mobile device by entering the access URL in your browser, entering your credentials, and completing the required steps.

Procedure

Installing the mobile client on an iOS device is similar to installing it on Android, but with some differences. The installer is automatically launched directly after download. Your user name and password credentials are requested for almost all the installation steps.

1. Start the browser on your mobile device.
2. Enter the following access URL in the address text field: `http://`*`hostname`*`:`*`portnumber`*`/appcenterconsole/installers.html`

   Where *hostname* is the address of the server and *portnumber* is the number of the port where the Application Center is installed. Your system administrator can provide this information.
3. Select an item in the list of available applications to display the application details.
4. Tap **Install Now** to download the mobile client.
5. Enter your credentials to authorize the downloader transaction.
6. To authorize the download, tap **Install**.



9.128.91.11 would like to install "App Center Installer"

Figure 92. Confirm app to be installed

7. Enter your credentials to authorize the installation.

   If you entered valid credentials, the browser will close and you can watch the download progress.

Installing the client on a BlackBerry mobile device

You can install the mobile client on your BlackBerry mobile device by entering the access URL in your browser, entering your credentials, and completing the required steps.

ight offline user documentation
ment is provided "as is".
In case efer to the online user documentation.

Procedure

The installer is automatically launched directly after download. Your user name and password credentials are requested for almost all the installation steps.

1. Start the browser on your mobile device.
2. Enter the following access URL in the address text field: `http://`*`hostname`*`:`*`portnumber`*`/appcenterconsole/installers.html`.

   Where *hostname* is the address of the server and *portnumber* is the number of the port where the Application Center is installed. Your system administrator can provide this information.
3. Enter your credentials to authorize access to the server.
4. Select an item in the list of available applications to display the application details.
5. Tap **Install Now** to download the mobile client.
6. In the BlackBerry Over The Air Installation Screen, tap **Download** to complete the installation.

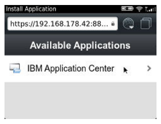


Figure 93. The installer in the BlackBerry browser

Views in the mobile client

The mobile client provides views that are adapted to the various tasks that you want to perform.

When you open the mobile client, you see the available views.

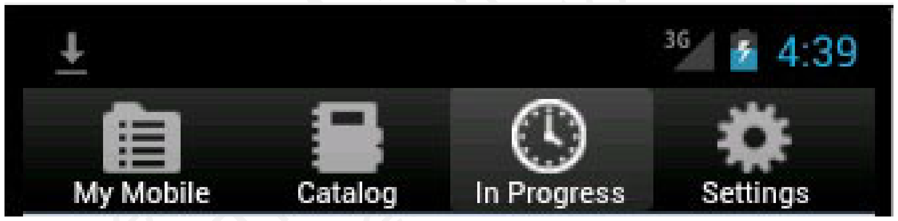


Figure 94. Views of the mobile client (Android and iOS operating systems)

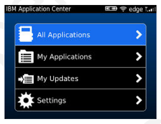


Figure 95. Views of the mobile client (BlackBerry devices)

These views enable you to communicate with a server to send or retrieve information about applications or to manage the applications located on your device. Here are descriptions of the different views.

My Mobile on Android and iOS versions.

My Applications on BlackBerry version.

Expiring ... Worklight offline user documentation
In case of issue, the document is provided "as is" ... the online user documentation.

This view shows the applications installed on your mobile device.

Catalog on Android and iOS versions.

All Applications on BlackBerry version.

This view shows the applications that can be installed.

In Progress on Android and iOS systems.

This view shows the progress of installing an application. This view is not required in the BlackBery version of the mobile client.

Settings

This view shows login information.

You can refresh a view by tapping the Refresh button.

When you first start the mobile client, it opens the **Settings** view for you to enter your user name, password, and the address of the Application Center server. This information is mandatory.

The layout of the views is specific to the Android, iOS, or BlackBerry environment, even though the functions you can perform in the views are the same for both operating systems. For example, the tabs are located in the upper part of the view on an Android device and in the lower part of the view on an IOS device. Devices might have quite different screen real estate, so all of the information a main tab view has to show might not be available at one time.

The Settings view

In the Settings view, you can access the fields that are required to connect to the server to view the list of applications available for your device.

Use the **Settings** view to enter your credentials to connect to the Application Center server to view the list of applications available for your device.

The **Settings** view presents all the mandatory fields for the information required to connect to the server.

User name and password

Enter your credentials for access to the server. These are the same user name and password granted by your system administrator for downloading and installing the mobile client.

Application Center server address

A preformatted example address fills this field. Use it as a model for entering the correct server address to avoid address format errors.

Secure Socket Layer (SSL)

Select SSL to turn on the SSL protocol for communications over the network. (Tapping this field again when SSL is selected switches SSL off.)

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

SSL selection is available for cases where the Application Center server is configured to run over an SSL connection. Selecting SSL when the server is not configured to handle an SSL layer prevents you from connecting to the server. Your system administrator can inform you whether the Application Center runs over an SSL connection.

Connect to the server.

1. Enter your user name and password.
2. Edit the example server address to make it correspond to your Application Center configuration.
3. If your configuration of the Application Center runs over the SSL protocol, select **SSL**.
4. Tap **Connect** to connect to the server.

When you start the Application Center on subsequent occasions, it opens in the **Settings** view. You are required to enter only your password. Your user name and the server address are retained.

Offline mode notification

Offline mode notification is shown in the Android and iOS versions of the mobile client only. If the connection to the server is lost while the mobile client is running, an unplugged icon is displayed in the current view, which could be the **My Mobile**, **Catalog**, or **In Progress** view. This icon disappears at the first successful communication attempt with the server.

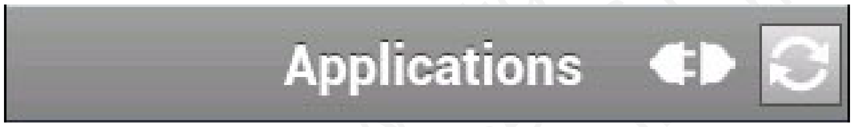


Figure 96. Connection lost indicated on Android devices

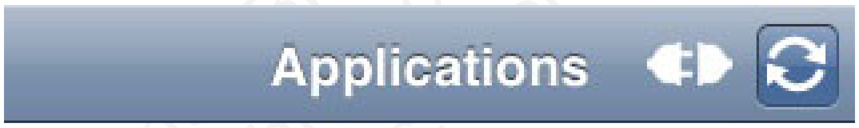


Figure 97. Connection lost indicated on iOS devices

The My Mobile and My Applications views

In the **My Mobile** and **My Applications** views, you can manage applications installed on your device; you can remove or update these applications and send feedback about them.

The view shows the installed applications and available updates for those applications. The location of this information in the screen real estate depends on the type of mobile device.

Depending on the type of device, you can display the entire content of the view at one time (on a tablet) or switch between different parts of the view (on a smartphone).

For example, to switch between the application list and the properties of an application, tap the application name. To switch back, tap **Back**.

Experimental IBM Worklight Information, user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

Installed

This list shows all the applications installed on your mobile device.

Update or Recommended

This list shows the updates that are available for the applications installed on your mobile device and applications that have been recommended by the administrator.

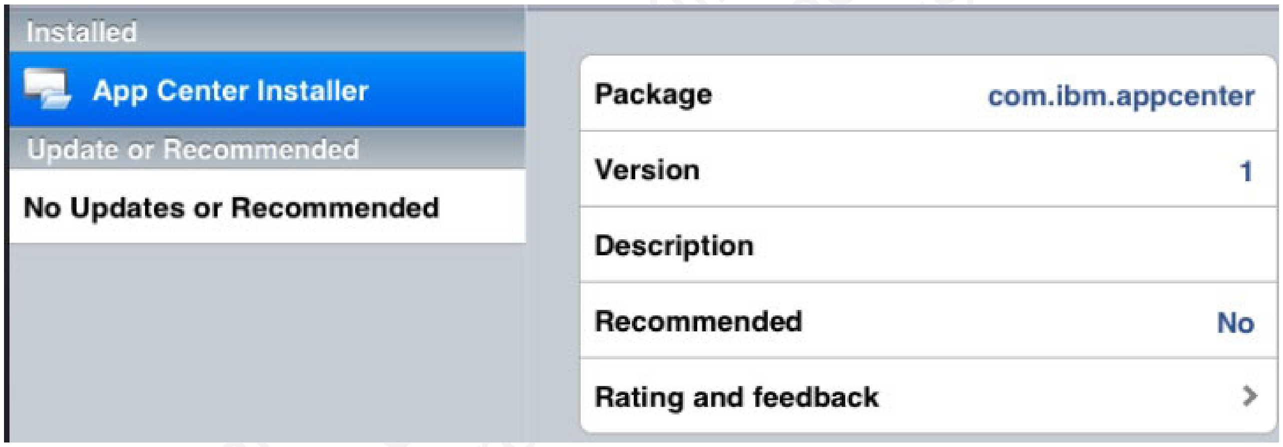
Application properties

The properties of an application are accessed by tapping the application name.

The properties shown are the package, version, description, and recommendation level of the installed application.

You can use this part of the **My Mobile** view to manage the application.

- Submit feedback about the application.
- Upgrade or downgrade the update state of the application.
- Remove the application from your mobile device.



Installed

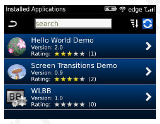
App Center Installer

Update or Recommended

No Updates or Recommended

Package	com.ibm.appcenter
Version	1
Description	
Recommended	No
Rating and feedback	>

Figure 98. Installed application list, updates or recommended versions, and application properties



Installed Applications

search

Hello World Demo
Version: 2.0
Rating: ★★★★★ (1)

Screen Transitions Demo
Version: 0.9
Rating: ★★★★★ (2)

WLBB
Version: 1.0
Rating: ★★★★★ (0)

Figure 99. Installed application list on BlackBerry devices

Removing an installed application

You can remove an application that is installed on your mobile device.

Procedure

1. Open the **My Mobile** or **My Applications** view.
2. Remove an application:
   - Android: In the installed applications list, select the application you want to remove and tap **Remove Now** in the application properties part of the view.

See “The My Mobile and My Applications views” on page 481 for how to switch between different parts of the view.

The mobile client is one of the installed applications. If you select it, you can shut it down and remove it from your mobile device by tapping **Remove Now**.

- iOS: Use the normal iOS procedure for removing an application and then click **Report Not Install**.
- BlackBerry: Follow steps a. and b.

a. Select the application to see the application detail screen. If the application is installed, an **Uninstall** button is displayed.

b. Tap the **Uninstall** button. Removing an installed application often results in a reboot request. If you choose to reboot later, the list of installed applications displayed in the mobile client might temporarily become unsynchronized until the next reboot.

Rating an installed application

You can rate your appreciation of an application and give feedback about it.

About this task

You can rate an application in the **My Mobile** or **My Applications** view.

Procedure

1. Tap the name of the application that you want to rate in the list of installed applications.
2. Make the next gestures according to the kind of device.
   - On Android or iOS operating systems: Expand the **`Rating and feedback`** property.
   - On BlackBerry devices: Scroll horizontally to the **Feedback** pane and tap **Submit Feedback**.
3. Tap a star, from 0 to 5, to represent your approval rating of the version of the application. On a BlackBerry device without touch screen, use the trackpad to slide horizontally to select the number of stars.

   One star represents the lowest level of appreciation and five stars represents the highest level of appreciation.
4. Enter a comment about this version of the application.
5. Tap **Submit** to send your feedback to the server; tap **Back** or **Cancel** to return to the application properties without submitting your feedback.

Updating an installed application (Android and iOS)

You can update an application that is installed on your iOS or Android device.

About this task

You can update an application in the My Mobile view.

If a more up-to-date version of an installed application is available on the server, it is listed under **Update or Recommended**.

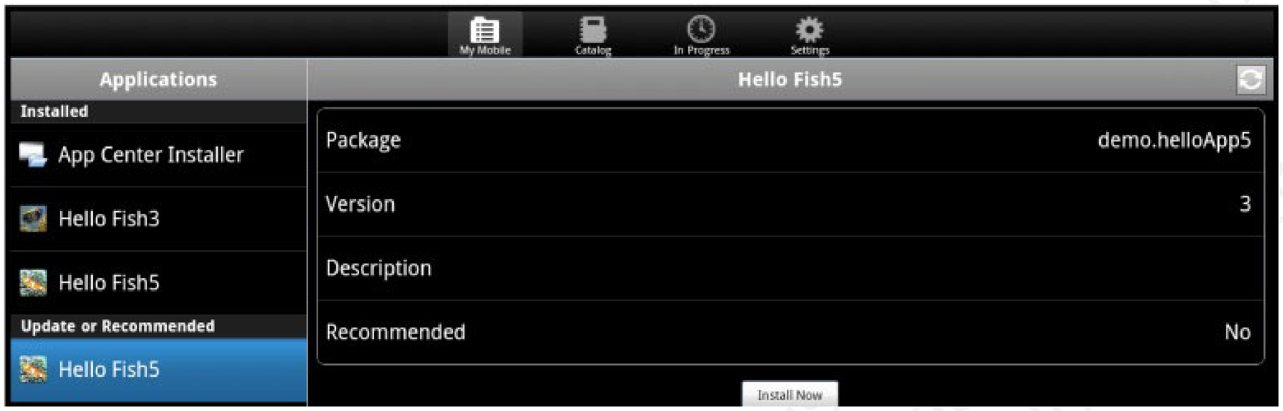
You can update the version of the installed application.

Experimental IBM copyright offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

Procedure

1. Tap the version of the application that you want to update to.

   The version number of the application is updated in the application properties.
2. Tap **Install Now** to start the update process.



Applications

Installed

App Center Installer

Hello Fish3

Hello Fish5

Update or Recommended

Hello Fish5

Hello Fish5

Package	demo.helloApp5
Version	3
Description	
Recommended	No

Figure 100. Updating the version of an installed application

3. Confirm that you want to replace the currently installed version with the updated version or cancel the update.
4. To confirm or cancel version installation on an Android or iOS device, see related tasks.

Related tasks:

"Confirming or canceling version installation on an Android device"
On an Android device, you can confirm the version installation and wait for it to install, or you can cancel the version installation and complete cancellation of the upgrade.

"Confirming or canceling installation on an iOS device" on page 485
On an iOS device, you can confirm or cancel the version installation by entering your credentials and choosing the appropriate option.

Confirming or canceling version installation on an Android device

On an Android device, you can confirm the version installation and wait for it to install, or you can cancel the version installation and complete cancellation of the upgrade.

Procedure

You can see the rights that are granted to the upgraded application when you choose to install it.

1. Tap **Install** to confirm upgrade of the application version or **Cancel** to cancel the upgrade.
2. Optional: If you tap **Install**, wait for the installation progress to complete.
   a. Open the new version of the application or remain in the **In Progress** view.
3. Optional: If you tap **Cancel**, you must perform the following steps to complete cancellation of the upgrade.
   a. Go to the **In Progress** view.
   b. Select the version of the application to be canceled and tap **Cancel Now**.

      The cancellation process might take several minutes.

Experimental IBM Worklight offline user documentation.
This document is provided "as-is".
In case of issues, refer to the online user documentation.

c. Tap **Dismiss** to exit the information message.

   You are informed when the cancellation is complete.

Confirming or canceling installation on an iOS device

On an iOS device, you can confirm or cancel the version installation by entering your credentials and choosing the appropriate option.

Procedure

You must enter your credentials to continue the upgrade.

1. Enter your user name and password.

   When your credentials are validated, the **In Progress** view is displayed.
2. Tap **Install** to confirm upgrade of the application version or **Cancel** to cancel the upgrade.
3. Optional: If you tap **Install**, the application upgrade is managed by the operating system. The mobile client is closed and the upgrade progress is shown on the mobile device desktop.
   a. When the upgrade is complete, start the mobile client.
   b. Tap the **My Mobile** view to see that the version of the selected installed application is upgraded.
4. Optional: If you tap **Cancel**, you must complete the cancelation from the **In Progress** view.
   a. In the **In Progress** view, select the canceled item.
   b. Tap **Report Not in Progress** to complete the cancelation process for the same application.

Reverting an installed application (Android and iOS)

You can revert the version of an installed application if an earlier version exists on the server.

About this task

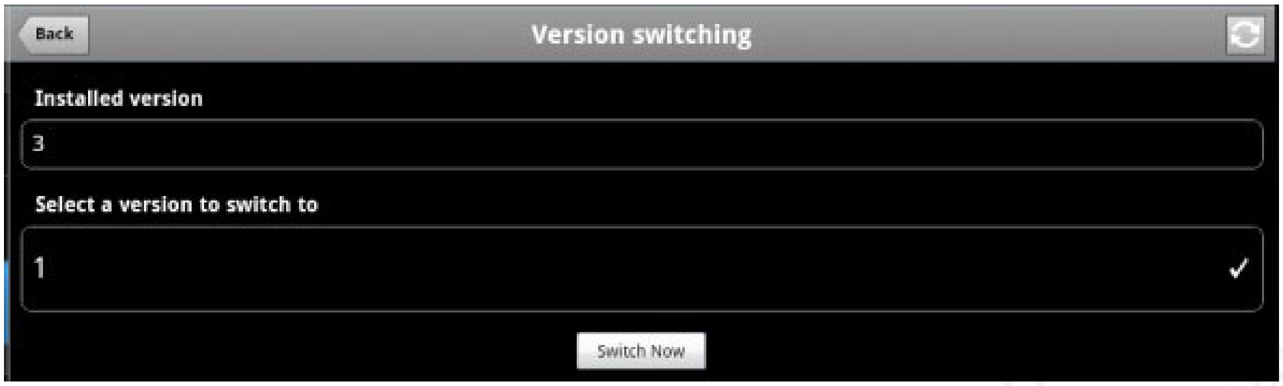
In the **My Mobile** view, you can revert the version of an installed application, provided that an earlier version is available on the server.

Procedure

1. Tap the name of the application you want to downgrade in the installed applications list.
2. Tap **Switch Version** in the application properties part of the view.

   Below the currently installed version is a list of versions available for downgrade.
3. Select the version that you want to revert.
4. Tap **Switch Now** to install the downgraded version.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.



Installed version
3
Select a version to switch to
1 ✓

Figure 101. Downgrading the version of an installed application

5. Confirm that you want to replace the existing installed version with the downgrade or cancel the downgrade.

   See related tasks.

   Note: Upgrading an application always works, but downgrading an application does not work on Android V4.2 and later.

Related tasks:

"Confirming or canceling version installation on an Android device" on page 484
On an Android device, you can confirm the version installation and wait for it to install, or you can cancel the version installation and complete cancellation of the upgrade.

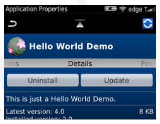
"Confirming or canceling installation on an iOS device" on page 485
On an iOS device, you can confirm or cancel the version installation by entering your credentials and choosing the appropriate option.

Updating and reverting an installed application (BlackBerry)

On BlackBerry, you can update or revert the version of an application. You can revert the version if an earlier version exists on the server.

About this task

In the **Application Properties** view, you can update or revert the version of an installed application. You access the **Application Properties** view after you select an application in the **All Applications**, **My Applications**, or **My Updates** view.



This is just a Hello World Demo.
Latest version: 4.0 8 KB

Figure 102. The Application Properties view on BlackBerry devices

Procedure

1. Select the application in the **All Applications** view, the **My Applications** view, or the **My Updates** view. The **Application Properties** view shows the information about the latest available version and the installed version of an application. In this view you can:
   - Install the latest version, if the application is not installed.

Experimental IBM Worklight offline user documentation.
This document is provided as is.
In case of issues, refer to the online user documentation.

- Update to the latest version, if an earlier version of the application is installed.
- Uninstall the application, if it is installed.

2. If you want to update or revert to a specific version, slide horizontally to the **Versions** pane. In the **Versions** pane, you see all available versions of the application.

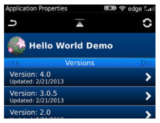


Figure 103. Selecting a version of an application

3. Select a version that you want to inspect. The **Application Properties** view is updated with information about this specific version. You can:
   - Update or revert to that specific version.
   - Uninstall that version, if it is installed.

   During the installation, a progress bar is displayed. While you are downloading the application, the installation can be canceled by tapping the red cross next to the progress bar. When the download of the application is complete, the installation can no longer be canceled.

   Updating or reverting the version of an application often results in a request for a reboot. If you choose to reboot later, the list of installed applications displayed in the mobile client might temporarily become unsynchronized until the next reboot.

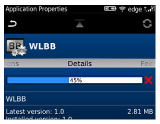


Figure 104. The progress bar during installation of an application version

The Catalog or All Applications view

In the **Catalog** or **All Applications** view, you can see a list of applications that can be installed on your device.

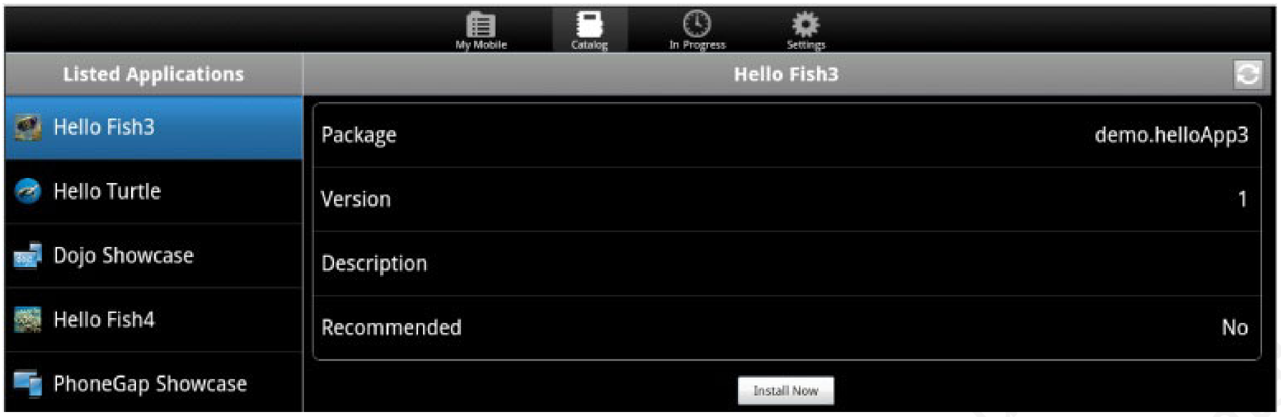
The **Catalog** or **All Applications** view has a list of applications that can be installed from the server and the properties and actions that are relevant to the selected application. The location of this information in the screen real estate depends on the type of mobile device.

Depending on the type of device, you can display the entire content of the view at one time (on a tablet) or switch between different parts of the view (on a smartphone).

For example, to switch between the application list and the properties and actions for an application, tap the application name. To switch back, tap **Back**.

Selecting an application retrieves its related properties and actions from the server.

Experimental IBM Worklight for user documentation.
This document is provided as is.
In case of issues, refer to the online user documentation.



Hello Fish3

Package	demo.helloApp3
Version	1
Description	
Recommended	No

Figure 105. Selecting an application to install on your device

Installing an application on an Android device

From the **Catalog** view, you can install an application on your Android device.

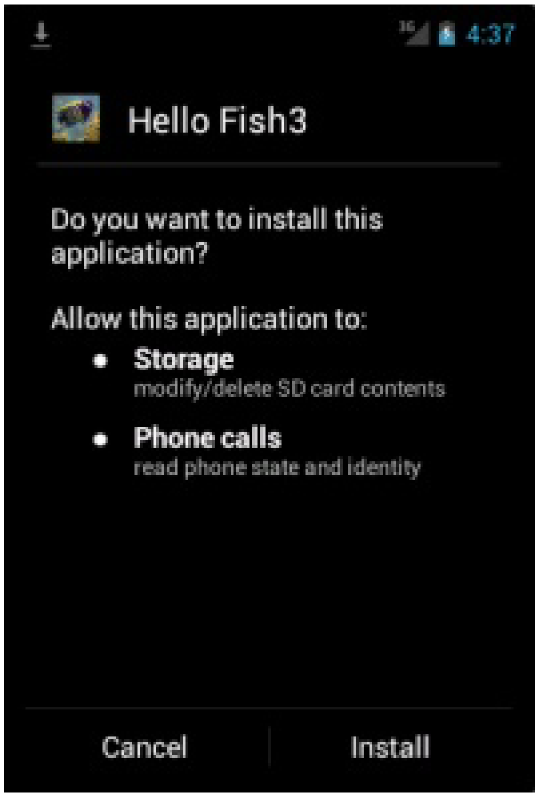
About this task

Installation is initiated from the **Catalog** view.

Procedure

1. Tap the name of the application you want to install.
   The application properties and action data are retrieved from the server.
2. Tap **Install Now**.
   You can see the rights that are granted to the upgraded application.

Experimental IBM Worklight offline user documentation.
This document is provided "as is".
In case of issues, refer to the online user documentation.


Hello Fish3

Do you want to install this application?

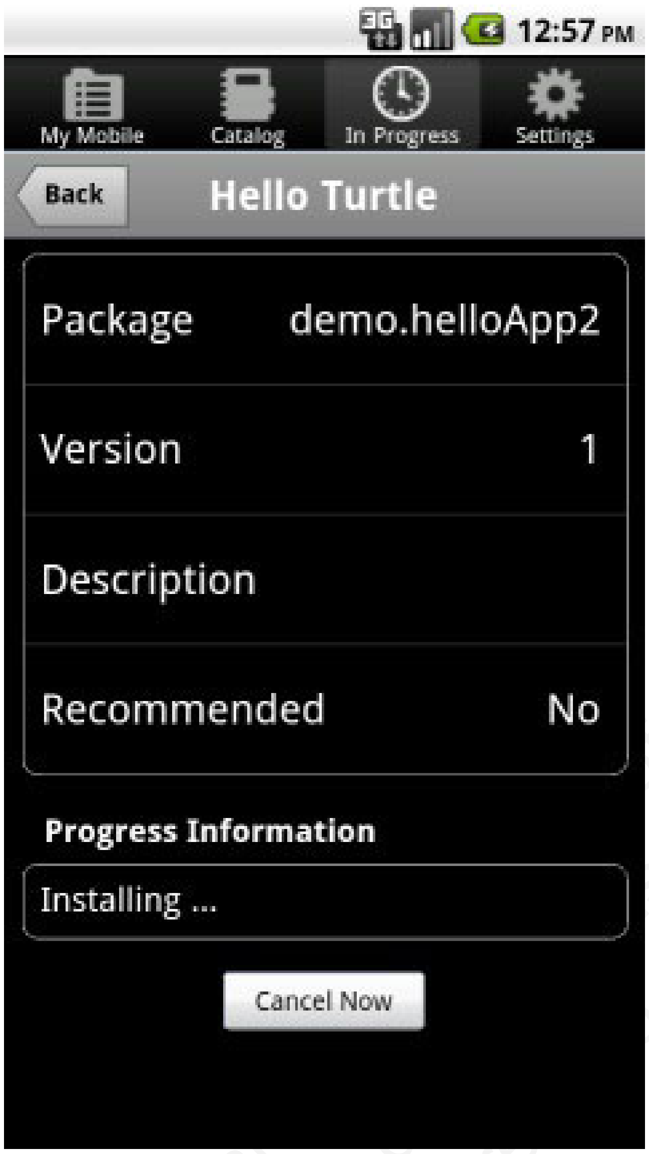
Allow this application to:

- **Storage**
  modify/delete SD card contents
- **Phone calls**
  read phone state and identity

Figure 106. Application rights on your Android device

3. Tap **Install** to confirm installation of the application or **Cancel** to cancel installation.
4. Optional: If you tap **Install**, wait for the installation progress to complete.
   a. Open the installed application or return to the mobile client.
5. Optional: If you tap **Cancel**, you must perform the following steps to complete cancellation of the installation.
   a. Go to the **In Progress** view.
   b. Select the application to be canceled and tap **Cancel Now**.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.



Hello Turtle

Package	demo.helloApp2
Version	1
Description	
Recommended	No

Progress Information

Installing ...

Figure 107. In Progress view, canceling application installation on your Android device

The cancellation process might take several minutes.

c. Tap **Dismiss** to exit the information message.
You are informed when the cancellation is complete.

Experimental offline user documentation. This documentation is provided "as is". In case of issues, refer to the online user documentation.

Installing an application on an iOS device

From the **Catalog** view, you can install an application on your iOS mobile device.

About this task

Look under Listed Applications in the **Catalog** view.

Procedure

1. Tap the name of the application you want to install.
   The application properties and action data are retrieved from the server.
2. Tap **Install Now**.

The **In Progress** view is displayed.

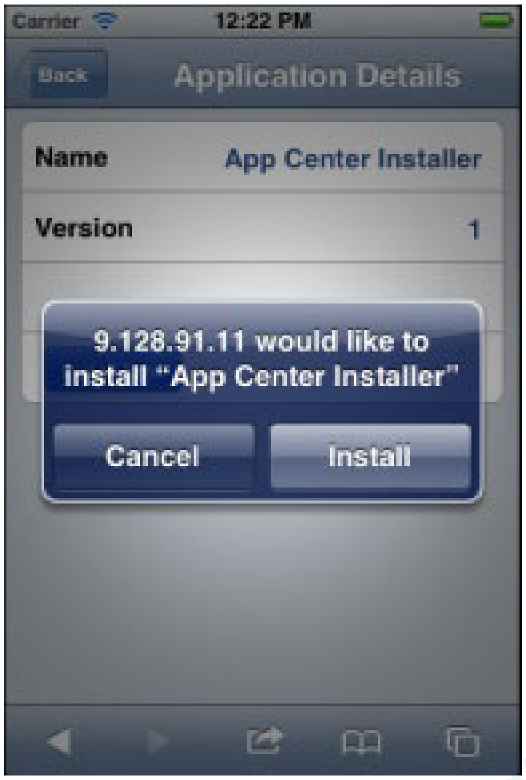


Figure 108. In Progress view, application installation on your iOS device

3. Tap **Install** to confirm installation of the application version or **Cancel** to cancel the installation.
4. Optional: If you tap **Install**, the installation of the application is managed by the operating system. The mobile client is closed and the installation progress is shown on the mobile device desktop.
   a. When the installation is complete, start the mobile client.
   b. Tap the **My Mobile** view to see that the selected application is included in the list of installed applications.
5. Optional: If you tap **Cancel**, you must complete the cancelation from the **In Progress** view.
   a. In the **In Progress** view, select the canceled item.
   b. Tap **Report Not in Progress** to complete the cancelation process for the same application.

Experimental IBM MobileFirst offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

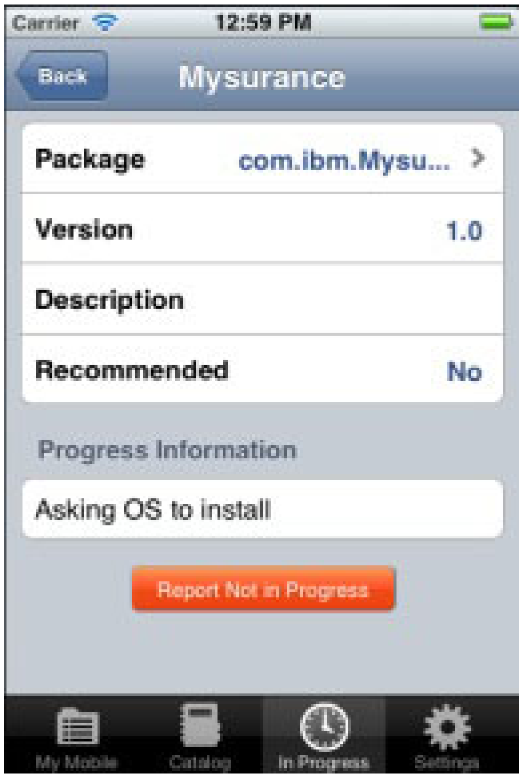


Figure 109. Canceling application installation on your iOS device

Installing an application on a BlackBerry device

From the **My Applications** view, you can install an application on your BlackBerry device.

About this task

On a BlackBerry device, you install an application from the **My Applications** view.

If the list of applications is too long, you can use the search field to find an application that contains the search string it its name.

You can also sort the application list by using the sort icon in the upper right corner.

Procedure

1. Tap or click the name of the application you want to install. The application properties and action data of the latest version of this application are retrieved from the server.
2. Tap or click **Install**. During the installation, a progress bar is displayed. While you are downloading the application, the installation can be canceled by tapping the red cross next to the progress bar. When the download of the application is complete, the installation can no longer be canceled.

   Updating or reverting an application often results in a request for a reboot. If you choose to reboot later, the list of installed applications displayed in the mobile client might temporarily become unsynchronized until the next reboot.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

The In Progress view (Android and iOS)

In the In Progress view, you can see the progress of the installation of an application or of an upgrade or downgrade of an application, or the cancelation of these actions.

The **In Progress** view shows applications that are at some stage of the installation or cancellation of the installation process. It also shows the progress of the upgrade or downgrade of a version of an application, or of the cancellation of such an upgrade or downgrade.

The BlackBerry version of the mobile client dooes not have this view, because progress is shown in the **Application Properties** view.

This view is automatically opened when an installation is in progress. You can also select the **In Progress** tab manually to access the view.

The **In Progress** view has a list of applications that are currently in the process of installation called **In Progress Applications**. The view also shows the status of the installation of the selected application. The location of this information in the screen real estate depends on the type of mobile device.

Depending on the type of device, you can display the entire content of the view at one time (on a tablet) or switch between different parts of the view (on a smartphone).

For example, to switch between the application list and the properties and actions for an application, tap the application name. To switch back, tap **Back**.

When no installation is in progress, the application list shows **Nothing to report** and you cannot switch between different parts of the view.

Installation

If you follow the normal procedure to install an application, the selected application is removed from the **Catalog** view (which shows applications that can be installed on your mobile device).

When you tap **Install Now**, you start the process to open the **In Progress** view. This view includes details of the application you selected to install; the package name, the application version, description, and recommendation level. It shows the current state of the installation of the application.

Cancellation

If you cancel an installation, the cancellation interrupts asynchronous server transactions. Facilities in the **In Progress** view enable you to terminate the installation request and to restore the list of applications that can be installed on your mobile device to a real and accurate state. For example, see "Installing an application on an Android device" on page 488 or "Installing an application on an iOS device" on page 490.

The installation of the application is considered to be in progress and it is removed from the **Catalog** view.

Experimental IBM Worklight offline documentation. This document is provided "as-is". In case of issues, refer to the online user documentation.

Details specific to the Android environment

If you accept installation of the application, you can follow the progress of the installation. When installation is complete, you can open the application or return to the **In Progress** view in the mobile client. The installed application is no longer shown under **In Progress Applications**.

The part of the view showing the application details and installation progress exists primarily to provide the opportunity to cancel the installation (see "Installing an application on an Android device" on page 488) and to interrupt repeated requests to confirm the installation.

Details specific to the iOS environment

The installation is performed by the operating system. For this reason, you must tap **Report Not in Progress** to actually cancel the installation. See "Installing an application on an iOS device" on page 490.

If you close the mobile client before the installation or cancellation operation is complete, this facility is available for the selected application in the **My Mobile** view. You can then tap **Report Not in Progress** in the **My Mobile** view.

Advanced information for BlackBerry users

You have a choice of connection suffixes for manual connection between the mobile client and BlackBerry.

Purpose

Sometimes you might have to set up the connection between the Application Center mobile client and BlackBerry service manually. This information helps you to set the correct connection.

The mobile client connects to the Application Center Server through HTTP. BlackBerry offers a wide range of HTTP connection modes that can be controlled by a connection suffix. The Application Center mobile client tries to detect the connection mode automatically. By default, the mobile client tries Wifi, then WAP 2.0, and then direct TCP over the mobile carrier (GPRS, 3G, and so on).

Setup of a manual connection

In rare cases, it might be necessary to set up the connection suffix manually.

On the BlackBerry home screen:
1. Open **Options**.
2. Open **Third Party Application**.
3. Open **IBM Application Center**.

   You can then specify the connection suffix and the connection timeout parameter.

The table shows the possible connection suffixes. For corporate-owned devices, you might need to contact your network administrator for the correct connection suffix. Corporate-owned devices might disallow certain connection modes in the service book of the device.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

See http://supportforums.blackberry.com/t5/Java-Development/Network-Transports/ta-p/482457 for more details.

Table 115. Details for manual connection

Connection suffix	User type	Conditions	Connection path
interface=wifi	All users	Wifi must be enabled. The device service book must allow Wifi. The device must be connected to a Wifi access point.	Device > Wifi access point > Internet > IBM Application Center Server
deviceside=true	All users	The mobile carrier must allow data connections. The device service book must allow direct TCP. The mobile carrier's APN must be set up.	Device > Mobile carrier > Internet > IBM Application Center Server
deviceside=true;apn=xyz		Similar to deviceside=true, but uses the specified APN.	
deviceside=true;apn=xyz;TunnelAuthUsername=user;TunnelAuthPassword=password		Similar to deviceside=true, but uses the specified APN and user name and password.	
deviceside=true;ConnectionUID=xyz	All users	The mobile carrier must allow data connections. The device service book must allow WAP 2.0. The WAP 2.0 connection details for the UID must be set up in the service book.	Device > Mobile carrier > WAP 2.0 Gateway > Internet > IBM Application Center Server
deviceside=true;WapGatewayIP=127.0.0.1;WapGatewayPort=9201; WapGatewayAPN=xyz	All users	The mobile carrier must allow data connections. The device service book must allow WAP 1.0/1.1.	Device > Mobile carrier > WAP 1.0 /1.1 Gateway > Internet > IBM Application Center Server

Experimental IBM Worklight offline user documentation.
This document is provided as-is.
In case of issues, refer to the online user documentation.

Table 115. Details for manual connection (continued)

Connection suffix	User type	Conditions	Connection path
deviceside=false	Corporate users	Your corporate entity must set up a BlackBerry Enterprise Server (BES) for mobile device services (MDS). The MDS connection UID must be set up in the service book.	Device > Wifi or Mobile carrier > Blackberry Infrastructure Network Operation Center (NOC) > Corporate BES > IBM Application Center Server
deviceside=false;ConnectionUID=xyz		Similar to deviceside=false, but uses the specified UID. This setting is useful when your corporate entity has set up multiple BES.	
A secret connection suffix.	BlackBerry Alliance members	You must be a member of the BlackBerry Alliance. In this case, you have received your own connection suffix. Instead of a corporate BES, you connect to the central Internet Service Browsing Server (BIS-B) of BlackBerry.	Device > Wifi or Mobile carrier > Blackberry Infrastructure Network Operation Center (NOC) > BIS-B > IBM Application Center Server
EndToEndRequired	All users	SSL connections only; use this suffix in combination with the other connection suffixes.	Device > ... > IBM Application Center Server is fully SSL encrypted
EndToEndDesired	All users	SSL connections only; use this suffix in combination with the other connection suffixes.	Device > ... > (BES or BIS-B does not necessarily use SSL) BES or BIS-B > ... > IBM Application Center Server uses SSL

Application rating feature called from another application (advanced feature)

You can send feedback to the server from another IBM Worklight application.

Experimental IBM Worklight offline user documentation. This document is provided as-is. In case of issues, refer to the online documentation.

The application rating feature described in “Rating an installed application” on page 483 can be called from another IBM Worklight application. The Worklight application must be installed on the mobile device by using the Application Center mobile client.

The rating and feedback form of the Application Center mobile client for the Worklight application that issues the call is shown when the API is called. The form contains any rating and comment that were passed through the API. The user can edit the feedback and tap **Submit** to send the feedback to the server.

API parameters

The same API parameters are required on both Android and iOS operating systems.

Parameters:	
pkg	- The application package ID. Mandatory.
version	- The application package version; if unspecified, the version installed. Optional.
stars	- A value between 0 and 5 representing the number of stars of the rating. Optional.
comment	- The comment string. Optional.

On Android systems, the API is implemented with an Android intent class, with an action and extras corresponding to the parameters.

On iOS, the API object is part of a URL that is completed by the values of the parameters.

API for Android operating system

Start an intent with the action `com.ibm.appcenter.FEEDBACK_REQUEST`. The intent must contain an extra with the package ID named **pkg**. Optionally, it can also contain three more extras: **version**, **stars**, and **comment**.

Example of use of the API on Android

```
Intent intent = new Intent("com.ibm.appcenter.FEEDBACK_REQUEST");
intent.putExtra("pkg", "com.TestFeedbackApp");
//optional
intent.putExtra("version", 2);
//optional
intent.putExtra("stars", 1);
//optional
intent.putExtra("comment", "Good");
//optional
```

API for iOS

Open the URL:

`ibmappctrfeedback://`*`pkg`*`/`*`version`*`?stars=`*`number`*`&comment=`*`text`*.

Where the parameters in the path:

pkg	is mandatory.

Experimental IBM Worklight offline user documentation.
In case of issues, refer to the online user documentation. This document is provided "as is".

`version`	is optional.
`stars`	is optional.
`comment`	is optional.

Example of use of the API on iOS

```
ibmappctrfeedback://com.TestFeedbackApp/2?stars=1&comment=Good
```

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Chapter 8. Deploying to the cloud by using IBM PureApplication System

IBM Worklight provides the capability to deploy IBM Worklight Servers and applications to the cloud by using IBM PureApplication System.

Using Worklight in combination with PureApplication System provides a simple and intuitive environment for developers and administrators to develop mobile applications, test them, and deploy them to the cloud. The following components are available:

IBM Mobile Application Platform Pattern Type
Provides Worklight runtime and artifacts support for PureApplication System.

IBM Worklight PureApplication System Extension for Worklight Studio
Provides PureApplication System tooling support for Worklight Studio.

Ant command line interface
Provides an alternative method to build and deploy Worklight Virtual Application.

Installation of IBM Worklight support for PureApplication System

You must install the IBM Mobile Application Platform Pattern Type and IBM Worklight PureApplication System Extension for Worklight Studio.

Installing the IBM Mobile Application Platform Pattern Type

You use the PureApplication System Workload Console to install the IBM Mobile Application Platform Pattern Type.

Before you begin

You can find the `worklight.ptype-5.0.6.0.tgz` file in the `worklight-pattern-offering.zip` file. Make sure you extract it before you start this procedure.

Procedure

1. Log in to IBM PureApplication System with an account that has permission to create new pattern types.
2. Go to **Workload Console** > **Cloud** > **Pattern Types**.
3. Upload the IBM Mobile Application Platform Pattern Type `.tgz` file.
4. On the toolbar, click the plus (+) button. The "Install a pattern type" window opens.
5. On the Local tab, click **Browse**, select the IBM Mobile Application Platform Pattern Type `.tgz` file, and then wait for the upload process to complete. The pattern type is displayed in the list and is marked as not enabled.
6. In the list of pattern types, click the uploaded pattern type. Details of the pattern type are displayed.
7. In the License Agreement row, click **License**. The License window is displayed stating the terms of the license agreement.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

8. To accept the license, click the **Accept** button. Details of the pattern type now show that the license is accepted.
9. In the Status row, click **Enable**. The pattern type is now listed as being enabled.

Installation of IBM Worklight PureApplication System Extension for Worklight Studio

IBM Worklight PureApplication System Extension is included with IBM Worklight Studio Enterprise Edition and Consumer Edition. When IBM Worklight Studio is installed in the Eclipse development environment, the Worklight PureApplication System Extension is also installed.

For more information about installation and configuration of IBM Worklight Studio, see "Setting up IBM Worklight Studio" on page 13.

Working with the IBM Mobile Application Platform Pattern Type

Working with the IBM Mobile Application Platform Pattern Type involves creating an IBM Mobile Application Platform Pattern, integrating with Tivoli Directory Server, connecting to a Tivoli Directory Server, and managing Worklight VAP instances.

Composition and components

The IBM Mobile Application Platform Pattern Type is composed of the IBM Web Application Pattern and the IBM Mobile Application Platform Pattern. The IBM Mobile Application Platform Pattern provides a number of components.

Composition

IBM Mobile Application Platform Pattern Type is composed of the following patterns:
- IBM Web Application Pattern
- IBM Mobile Application Platform Pattern

Components

In addition to all components provided by IBM Web Application Pattern, IBM Mobile Application Platform Pattern provides the following components:
- Worklight application component
- Worklight adapter component
- Worklight configuration component
- Worklight application component link to enterprise application (Websphere Application Server) component
- Worklight adapter component link to enterprise application (Websphere Application Server) component
- Enterprise application (Websphere Application Server) component link to Worklight configuration component
- Worklight configuration link to user registry (Tivoli Directory Server)

Creating an IBM Mobile Application Platform Pattern

You create an IBM Mobile Application Platform Pattern by creating and configuring an IBM Worklight Server and database, configuring database connectivity, and uploading applications and adapters.

Experimental IBM Worklight online user documentation
In case of issues, refer to the online user documentation.
This document is provided "as-is".

Before you begin

This procedure involves uploading certain artifacts to IBM PureApplication System such as the IBM Worklight Server. Before you begin, ensure that the artifacts are available for upload.

Procedure

1. Create an IBM Worklight Server.
   a. If necessary, use the Worklight Studio PureApplication System Extension or the command line interface to package up the IBM Worklight Server into an EAR file.
   b. In IBM PureApplication System, in the Virtual Application Builder, click the **Diagram** tab.
   c. From the Assets list, expand **Application Components**, and then drag and drop an Enterprise Application WebSphere Application Server component onto the canvas.
   d. Supply the following information in the fields provided:

Table 116. IBM Worklight Server component properties

Property	Description
Name	Name for the IBM Worklight Server.
EAR file	Worklight EAR file that contains the IBM Worklight Server package to be uploaded.

2. Create a Worklight database.
   a. From the Assets list, expand **Database Components**, and then drag and drop a Database DB2 component onto the canvas.
   b. Supply the following information in the fields provided:

Table 117. Worklight database component properties

Property	Description
Name	Name for the Worklight database component.
Database name	Name for the database.
Schema file	Select the `create-worklight-db2.sql` or `create-worklightreports-db2.sql` file. These files can be found in the IBM Worklight Server installation directory.

3. Configure database connectivity.
   a. Drag a connection from the IBM Worklight Server component to the database component.
   b. In the **JNDI Name of Data Source** field, enter the JNDI name of the Worklight datasource; for example, `jdbc/WorklightDS`.
   c. Repeat the previous steps to create a Worklight reports database component and a link from the IBM Worklight Server component to the reports database component.
4. Configure the IBM Worklight Server.
   a. From the Assets list, expand **Worklight Components**, and then drag and drop a Worklight Configuration component onto the canvas.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

b. Create a link from the IBM Worklight Server component to the Worklight configuration component.

c. Supply the following information in the fields provided:

Table 118. Worklight configuration component properties

Property	Description
Name	Name for the Worklight configuration component.
Worklight Console Protection	Select this check box to enable security protection for the Worklight console. Clear the check box to disable security protection.
Worklight Console Username	User name for Worklight console protection.
Worklight Console Password	Password for Worklight console protection.

5. Create Worklight applications and adapters.

   a. From the Assets list, expand **Worklight Components**, and then drag and drop a Worklight adapter component and a Worklight application component onto the canvas.

   b. For the Worklight application component, supply the following information in the fields provided:

Table 119. Worklight application component properties

Property	Description
Name	Name for the Worklight application.
Worklight Application Files	Worklight application files to upload. Supported formats are `*.wlapp` and `*.zip`.

   c. For the Worklight Adapter component, supply the following information in the fields provided:

Table 120. Worklight adapter component properties

Property	Description
Name	Name for the Worklight adapter.
Worklight Adapter Files	Worklight adapter files to upload. Supported formats are `*.wlapp` and `*.zip`.

   d. Create links from the Worklight application component to the IBM Worklight Server component, and from the Worklight adapter component to the IBM Worklight Server component.

Integrating with Tivoli Directory Server

Tivoli Directory Server is supported as a directory server in IBM Mobile Application Platform Pattern and can be used in conjunction with **`LdapLoginModule`** provided by IBM Worklight.

To use Tivoli Directory Server, **`LDAPLoginModuleIPAS`** must be defined in your Worklight application. For more information, see “Integration with Tivoli Directory Server” on page 506

Experimental IBM Worklight offline user documentation. This document is provided "as-is". In case of issues, refer to the online user documentation.

Connecting to a new Tivoli Directory Server

You connect to a new Tivoli Directory Server by dragging and dropping a new User Registry TDS component onto the PureApplication System canvas, linking the Worklight configuration component to it, and then uploading an LDIF file for Tivoli Directory Server.

Procedure

1. In IBM PureApplication System, in the Virtual Application Builder, click the **Diagram** tab.
2. From the Assets list, expand **User Registry Components**, and then drag and drop an User Registry Tivoli Directory Server component onto the canvas.
3. Supply the following information in the fields provided:

Table 121. Tivoli Directory Server component properties

Property	Description
Name	Name for the directory server.
LDIF file	LDIF file to be uploaded for the Tivoli Directory Server.
Base DN	Effective only when the LDAP login module has the parameter **`baseDN`**.
User filter	Effective only when the LDAP login module has the parameter **`userFilter`**.
Group filter	Effective only when the LDAP login module has the parameter **`groupFilter`**.

Connecting to an existing Tivoli Directory Server

You connect to an existing Tivoli Directory Server by dragging and dropping a Connect Out component onto the PureApplication System canvas, specifying the IP address and port number of your existing Tivoli Directory Server, and linking the IBM Worklight Server component to the Connect Out component.

Procedure

1. In IBM PureApplication System, in the Virtual Application Builder, click the **Diagram** tab.
2. From the Assets list, drag and drop a Connect Out component onto the canvas.
3. Specify the IP address and port number of your existing Tivoli Directory Server.
4. Drag a link from the IBM Worklight Server component to the Connect Out component.

Performing operations on running IBM Worklight VAP instances

Use the PureApplication System Workload Console to perform management tasks on a running IBM Worklight VAP instance.

Procedure

1. In IBM PureApplication System, in the Workload Console, click the **Instances** tab.
2. From the Virtual Application Instances list, click the required instance, and then click **Manage**.

Experimental IBM Worklight offline user documentation. This document is provided "as-is". In case of issues, refer to the online user documentation.

3. Click the **Operations** tab, and then from the Operations list, click **WORKLIGHT**.
4. In the details panel, you can perform the following operations:

Table 122. Operations on VAP instances

Operation	Description
Worklight Application/Adapter	Install or update Worklight applications and adapters. Supported file types: `*.wlapp`, `*.adapter`, `*.zip`.
Worklight Console Protection	Enable and disable security protection for the Worklight Console.
Worklight Console Username	Username for Worklight Console protection.
Worklight Console Password	Password for Worklight Console protection.

5. To submit the changes you have made, click the **Submit** button.
6. Navigate back to the **Instances** tab and verify that the status of the instance is displayed as "Running".

Upgrading IBM Mobile Application Platform Pattern

To upgrade IBM Mobile Application Platform Pattern, upload the `.tgz` file that contains the latest updates.

Procedure

1. Log into IBM PureApplication System with an account that is allowed to upload new system plugins.
2. Navigate to **Workload Console** > **Cloud** > **System Plug-ins**.
3. Upload the IBM Mobile Application Platform Pattern `.tgz` file that contains the updates.
4. Enable the plugins you have uploaded.
5. Redeploy the pattern.

Working with IBM Worklight PureApplication System Extension for Worklight Studio

Working with IBM Worklight PureApplication System Extension for Worklight Studio involves setting up PureApplication System preferences, deploying your Worklight project to PureApplication System, and integrating with Tivoli Directory Server.

Setting up PureApplication System preferences in IBM Worklight Studio

Set up PureApplication System preferences in Worklight Studio before you deploy IBM Worklight projects to PureApplication System.

Procedure

1. In Eclipse, click **Windows** > **Preferences** > **IBM Worklight For IPAS**.
2. Supply the following information in the fields provided:

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

Table 123. PureApplication System preferences

Property	Description
IPAS Host	IP address of the PureApplication System host.
User name	Account user name for accessing the PureApplication System host.
Password	Password for accessing the PureApplication System host.

3. Click the **Fetch Environment Profiles** button. Details of retrieved environment profiles are displayed in the Preferences panel.
4. From the **Profiles** list, select the correct profile for cloud deployment.
5. Click **Apply** to save the settings, and then click **OK** to close the Preferences panel.

Deploying an IBM Worklight project to PureApplication System

You deploy an IBM Worklight project to PureApplication System by running the project in Eclipse.

Before you begin

Before deploying to PureApplication System, write your Worklight application and test it in the local development environment. Since you are deploying to an environment outside Eclipse, make sure you have applied the correct settings for the Worklight Server location in the `worklight.properties` file. For more information, see "Configuring the IBM Worklight Server location" on page 406.

Procedure

1. In Eclipse, navigate to the Project Explorer view.
2. Right-click your IBM Worklight project, and then click **Run As** > **Deploy project to IPAS**.
3. Select Worklight applications and adapters to be deployed on PureApplication System, and then click **Run**.
4. In the Worklight Console, check the status and wait for the project to be deployed on PureApplication System. When the project has been deployed, a window opens displaying the Worklight Console URL.

Fetching the Worklight Console URL for a deployed IBM Worklight project

You can fetch the Worklight Console URL for a deployed IBM Worklight project by using a command available in the Worklight Console.

Procedure

In the Worklight Console, right-click the IBM Worklight project, and then click **Fetch Worklight Console URL on IPAS**. A window opens displaying the Worklight Console URL.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues, refer to the online user documentation.

Integration with Tivoli Directory Server

To use Tivoli Directory Server as a user registry on PureApplication System for your Worklight application, you need to implement an LDAP login module.

You need to implement the LDAP login module as follows:

- The name attribute of LoginModule must be set to LDAPLoginModuleIPAS.
- The module must include a parameter with a name attribute set to ldapProviderURL.

  This is an example of a suitable LDAP login module:

  ```
  <loginModule name="LDAPLoginModuleIPAS">
   <className>com.worklight.core.auth.ext.LdapLoginModule</className>
   <parameter name="ldapProviderURL" value="ldaps://192.0.2.123:636"/>
   ...
   ...
  </loginModule>
  ```
- If **Connect to a new TDS** is enabled in your Worklight project configuration, you need to specify a .ldif file.
- If **Connect to existing TDS** is enabled, the value of the **ldapProviderURL** parameter is taken as the Tivoli Directory Server address.

Building and deploying IBM Worklight virtual applications by using the command line interface

IBM Mobile Application Platform Pattern includes a set of Ant tasks to help you build Worklight virtual applications and artifacts and deploy to IBM PureApplication System.

Building an IBM Worklight virtual application

You can use an Ant task to build a Worklight virtual application.

Before you begin

The Ant tasks are contained in the worklight-ant.jar file, which you can find in the worklight-pattern-offering.zip file. Make sure you extract it before you build and deploy Worklight virtual applications with the command line interface.

About this task

The Ant task for building a Worklight virtual application has the following structure:

```
<taskdef resource="com/worklight/ant/defaults.properties"
    classpath="${taskdefClasspath}"/>
<target name="buildIPAS_VAP"
    depends="buildAll" >
    <vap-builder
        worklightWar="${worklightWar}"
        destinationFolder="${wlProjectDestDir}"
        artifactsFolder="${artifactsFolder}"
        elbHost="${elbHost}"/>
 </target>
```

The following table describes the attributes.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues refer to the online user documentation.

Table 124. Ant task build attributes

Attributes	Description
worklightWar	Required. The Worklight Console WAR file including the full file path.
destinationFolder	Optional. Default value: `${projectfolder}/bin`.
artifactsFolder	Optional. Folder for adapters and applications.
elbHost	Optional. Host name for elastic load balancer.
createVAPFlag	Optional. Whether to generate a VAP `.zip` file. Default value: `true`.
isConnectNewTDS	Optional. Whether to connect a new Tivoli Directory Service.
ldifFile	Optional. When **isConnectNewTDS** is true, you must set this attribute.
ipasModel	Optional. Default value is `W1500`; in this case, it works on Intel. You can set its value to `W1700`; in this case, IPAS® runs on Power® system.

Deploying an IBM Worklight virtual application

You can use an Ant task to deploy a Worklight virtual application.

Before you begin

The Ant tasks are contained in the `worklight-ant.jar` file, which you can find in the `worklight-pattern-offering.zip` file. Make sure you extract it before you build and deploy Worklight virtual applications with the command line interface.

About this task

The Ant task for deploying a Worklight virtual application has the following structure:

```
<taskdef resource="com/worklight/ant/defaults.properties" classpath="${taskdefClasspath}"/>
<target name="deployVAP" depends="buildVap4IPAS">
    <ipas-deployer
        vapZipFile="${vapFile}"
        ipasHost="${ipasHost}"
        username="${username}"
        password="${password}"
        profileName="${profileName}"
        cloudGroupName="${cloudGroupName}"
        ipGroupName="${ipGroupName}"
        priority="${ipasPriority}"
    />
</target>
```

The following table describes the attributes.

Table 125. Ant task deployment attributes

Attributes	Description
vapZipFile	Required. Path to the zip file built by vap-builder.

Experimental IBM Worklight offline user documentation
This document is provided as-is.
In case of issue, refer to the online user documentation.

Table 125. Ant task deployment attributes (continued)

Attributes	Description
ipasHost	Required. The URL of IBM PureApplication System.
username	Required. Username to access PureApplication System console.
password	Required. Password to access PureApplication System console.
profileName	Required. Profile name for deploying VAP.
cloudGroupName	Required. Cloud group name for deploying VAP.
ipGroupName	Required. IP group name for deploying VAP.
priority	Required. Priority for deploying VAP.

Deployment of the Application Center on IBM PureApplication System

You must configure and connect the operational components of the Application Center to deploy the enterprise application on PureApplication System.

The operational model of the Application Center is composed of:

- An application server that hosts the administration console and services.
- A user authentication system; here, an LDAP server that handles user and group authentication and user management, but the basic authentication mechanism of the application server can be used.
- The database, a repository for tracking users, applications, and feedback.

Experimental IBM Worklight offline user documentation
This document is provided as is.
In case of issues, refer to the online user documentation.

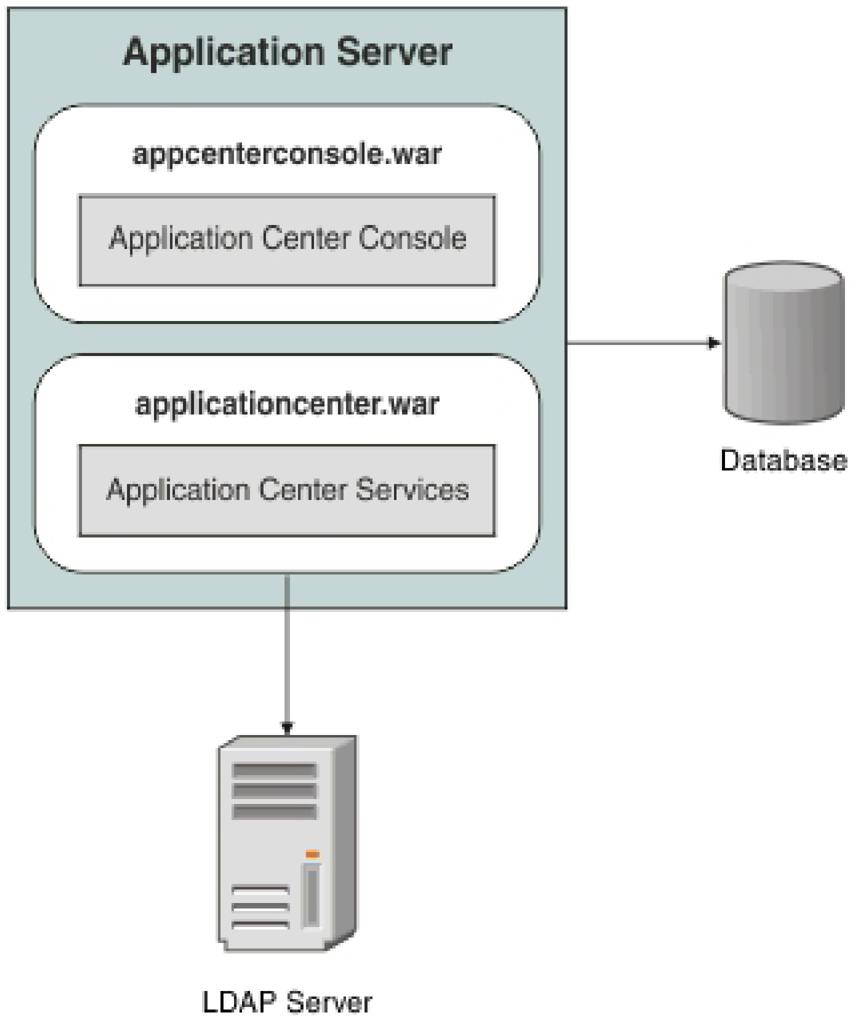


Figure 110. Typical operational model of the Application Center

Related concepts:

"Introduction to the Application Center" on page 423
Tells you about the Application Center: what it is for, the different components, how to get started, and the files in the distribution.

"Configuration of the Application Center after installation" on page 429
You configure user authentication and choose an authentication method; configuration procedure depends on the web application server that you use.

"Managing users with LDAP" on page 436
Use the Lightweight Directory Access Protocol (LDAP) registry to manage users..

Deploying the Application Center on IBM PureApplication System

Configure the enterprise application, the database, the user registry, and map the security roles before you deploy the Application Center on PureApplication System.

Experimental IBM. This documentation is provided "as is". In case of issue, refer to the user documentation.

Before you begin

Install the Application Center. The Application Center is part of IBM Worklight Server. You can install it through IBM Installation Manager or manually, See "Installation" on page 314 under IBM Worklight Server administration.

Make a note of the path of the installation folder, because later in the procedure you will need some assets that are located in it. When you install the Worklight Server through the IBM Installation Manager, the Application Center artifacts are installed in the `{worklight_install_folder }/ApplicationCenter`.

You must have an IBM PureApplication System environment and the privilege to create Virtual Application Pattern (VAP) and to run Virtual Application instances.

About this task

By following this procedure you prepare the operational components of the Application Center for deployment of the enterprise application on PureApplication System. You connect the operational components to each other and then you can save the configuration and deploy the Application Center on PureApplication System as a web application.

Procedure

1. Get the enterprise archive (EAR) file for the Application Center. This file is located in `{worklight_install_folder }/ApplicationCenter/console`. As of V5.0.6, the Application Center has two web archive (WAR) files, one for the console and one for the services. An EAR file containing them is supplied to simplify deployment on PureApplication System. The context roots of the WAR files within the EAR file are:
   - `/appcenterconsole` for the console
   - `/applicationcenter` for the services

   If you choose to build the EAR file manually, you must remember the context roots of the WAR files.
2. Create the Virtual Application Pattern.
   a. Log in to IBM PureApplication System
   b. Select **Workload Console** > **Patterns** > **Virtual Application Patterns**.
   c. Select **Web Application Pattern Type 2.0**.
   d. Click the plus (+) button.
   e. Select a template to start from and then click **Start Building**. You can select any template that conforms with the operational model used in this documentation. You must create one web application component, one database component, and one user registry component. The example is based on selection of "Blank application".
3. Add an Enterprise Application component.
   a. Expand **Application Components**.
   b. Drag the **Enterprise Application** component into the pane on the right.
   c. Select the component in this property pane and specify the path of the Application Center EAR file.
4. Add routing policy.
   a. Move the mouse over the Enterprise Application component and click the plus sign (+).
   b. Select **Routing Policy**.

Experimental IBM Worklight offline user documentation. This document is provided "as-is". In case of issues refer to the online user documentation.

c. In the property pane, click **Routing Policy** and specify **Virtual Host name**. Take a note of the host name, because you will use it later.

5. **Optional**: Add JVM policy. If you use the supplied EAR file or have defined the context root of the services WAR file as `/applicationcenter`, this step is optional.
   a. Select **JVM Policy** in the same way as you selected Routing Policy.
   b. In the property pane, specify **Generic JVM arguments**: `-Dibm.appcenter.services.endpoint=http://{virtual_host}/{services_context_root}` where:

      `virtual_host` is the virtual host name that you specified in Routing Policy.

      `services_context_root` is the context root of the services WAR file.
6. Add a database component.
   a. In the left pane, expand **Database Components**.
   b. Drag a database into the property pane on the right. The database used in the example is DB2.
   c. In the property panel, click the Database component and specify the schema file. You can find `create-appcenter-{db}.sql`, used in the example, in `{worklight_install_folder}/ApplicationCenter/database`.
7. Connect enterprise application and database.
   a. On the Enterprise Application component, click and drag the connection point on the right edge to the Database component. This gesture creates the connection between the web application and the database.
   b. Click the connector and specify the data source as `jdbc/AppCenterDS`.
8. Add a user registry component.
   a. In the left pane, expand **User Registry Components**.
   b. Drag the user registry component into the property pane.
   c. In the property pane, select the User Registry component and specify the "Base DN" and the "LDIF file".
9. Connect web application and user registry.
   a. Drag two connectors between the Enterprise Application component and the User Registry component.
   b. Specify the "Role name" `appcenteradmin`.
   c. Set "Mapping special subjects" to `AllAuthenticatedUsers`.
   d. Specify the next "Role name" `appcenteruser`.
   e. Set "Mapping special subjects" to `AllAuthenticatedUsers`.
10. Save the configuration and deploy the Application Center on PureApplication System.
   a. Save the virtual application; give it a name, for example, "Worklight Application Center".
   b. Return to **Virtual Application Patterns**. You should see the pattern that you created in this procedure.
   c. Click **Deploy** to deploy the Application Center on PureApplication System.

Experimental IBM Worklight user documentation. This document is provided as is. In case of issues, refer to the online user documentation.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Chapter 9. Troubleshooting and known limitations

You can find advice about how to troubleshoot problems, and more information about known limitations and Technote (Troubleshooting).

Troubleshooting

The following links point to troubleshooting topics in other parts of this user documentation. To navigate from there back to this topic, click the **Go Back** button in the menu bar above the topic, or click **Back** in your Web browser.

- "Troubleshooting Worklight Server" on page 404
- "Troubleshooting IBM Worklight Studio installation" on page 17
- "Troubleshooting a Cast Iron adapter – connectivity issues" on page 98
- "Troubleshooting information for synchronization" on page 157

Important: For more information about known limitations or issues in the product, see "Known limitations" on page 12 and the product Technote (Troubleshooting) information.

Experimental IBM Worklight offline user documentation.
This document is provided as-is.
In case of issues, refer to the online user documentation.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Chapter 10. Notices

Permission for the use of these publications is granted subject to these terms and conditions.

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.

Experimental IBM Worklight offline user documentation.
This document is provided as is.
In case of issues, refer to the online user documentation.

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.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues refer to the online user documentation.

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".

Copyright

© Copyright IBM Corp. 2006, 2014

US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Trademarks

IBM, the IBM logo, ibm.com®, Cast Iron, Cognos, DataPower, DB2, developerWorks, IPAS, Passport Advantage, Power, PureApplication, Rational, Rational Team Concert, Redbooks, Tealeaf, Tivoli, and WebSphere 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.

Experimental IBM Worklight Documentation
This document is provided as is.
In case of issues, refer to the online user documentation.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

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.

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

Experimental IBM Worklight offline user documentation.
This document is provided "as is".
In case of issues, refer to the online user documentation.

Chapter 11. 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 Developer Edition support community at:

https://www.ibm.com/developerworks/mobile/worklight/connect.html

If you would like a response from IBM, please provide the following information:

- Name
- Address
- Company or Organization
- Phone No.
- Email address

Experimental IBM Worklight online user documentation
This document is provided "as is".
In case of issues refer to the online documentation.

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Chapter 12. Terms and conditions for information centers

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein. IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed. You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. 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 www.ibm.com/legal/copytrade.shtml.

© Copyright IBM Corporation 2006, 2014.

Experimental IBM Worklight online user documentation.
This document is printed "as is".
In case of issues, refer to the online user documentation.

This information center is Built on Eclipse. (www.eclipse.org)

Experimental IBM Worklight offline user documentation
This document is provided "as is".
In case of issues, refer to the online user documentation.

Index

Experimental IBM Worklight online user documentation. This document is provided "as-is" regardless of issues, refer to the online user documentation.

Experimental IBM Worklight offline user documentation. This document is provided "as is". In case of issues refer to the online user documentation.

Experimental IBM Worklight offline user documentation. In case of issues, this document is provided "as is". refer to the online user documentation.

Experimental IBM Worklight offline user documentation.
This document is provided "as is".
In case of issues, refer to the online documentation.