



# IBM Worklight V6.0.0

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

## Chapter 1. Overview of IBM Worklight . . . 1

Introduction to mobile application development . . .	1
Overview of IBM Worklight main capabilities . . .	3
Introducing IBM Worklight components . . . . .	5
IBM Worklight editions . . . . .	8
System requirements for using IBM Worklight . . .	8
Matrix of features and platforms. . . . .	9
Known limitations . . . . .	9

## Chapter 2. What's new . . . . . 13

What's new in IBM Worklight V6.0.0.2 . . . . .	13
What's new in IBM Worklight V6.0.0.1 . . . . .	14
What's new in IBM Worklight V6.0.0 . . . . .	14
New functional testing capabilities . . . . .	14
New operational analytics feature . . . . .	15
Enhancing applications with location features . .	15
Enhanced rapid application development . . . .	16
Features to optimize IBM Worklight applications	17
Improved serviceability . . . . .	19
Enhanced IBM Worklight API . . . . .	20
Augmented support of mobile environments and	
operating systems . . . . .	22
Miscellaneous modifications . . . . .	22

## Chapter 3. Tutorials and samples . . . 25

## Chapter 4. Installing and configuring 35

IBM Worklight installation overview . . . . .	35
Installing Worklight Studio . . . . .	35
Running post-installation tasks . . . . .	36
Starting Worklight Studio. . . . .	37
Installing mobile specific tools . . . . .	37
Changing the port number of the internal	
application server . . . . .	39
Installing IBM Mobile Test Workbench for Worklight	40
Troubleshooting IBM Mobile Test Workbench for	
Worklight . . . . .	41
Installing Worklight Server . . . . .	41
Installation prerequisites . . . . .	42
Overview of the Worklight Server installation	
process . . . . .	43
Running IBM Installation Manager . . . . .	47
Distribution structure of Worklight Server . . .	62
Manually installing Application Center . . . . .	66
Configuring Worklight Server . . . . .	83
Backup and recovery . . . . .	83
Optimization and tuning of Worklight Server . .	84
Optimization and tuning of Worklight Server	
project databases . . . . .	86
Security configuration . . . . .	88
Transmitting IBM Worklight data on the	
BlackBerry Enterprise Server MDS channel . . .	94
Protecting your mobile application traffic by	
using IBM WebSphere DataPower as a security	
gateway . . . . .	95

Configuring SSL between IBM Worklight	
Servers and clients by using certificates that are	
not signed by a trusted certificate authority . .	106
Handling MySQL stale connections . . . . .	116
Installing the Application Center . . . . .	118
Configuring the Application Center after	
installation . . . . .	118
Configuring WebSphere Application Server full	
profile . . . . .	119
Configuring WebSphere Application Server	
Liberty Profile . . . . .	120
Configuring Apache Tomcat . . . . .	121
Managing users with LDAP . . . . .	121
Defining the endpoint of the application	
resources . . . . .	139
Configuring Secure Sockets Layer (SSL) . . . .	143
Typical topologies of an IBM Worklight instance	145
Setting up IBM Worklight in an IBM WebSphere	
Application Server Network Deployment V8.5	
cluster environment . . . . .	147
Setting up IBM Worklight in an IBM WebSphere	
Application Server Liberty Profile farm. . . . .	161
Installing and configuring the IBM WebSphere	
Analytics Platform. . . . .	174
Configuring Worklight Server for analytics . .	176
Troubleshooting Worklight Server . . . . .	177
Troubleshooting to find the cause of installation	
failure . . . . .	177
Troubleshooting failure to create the DB2	
database . . . . .	177
Troubleshooting an installation blocked by DB2	
connection errors . . . . .	178

## Chapter 5. Upgrading from one version of IBM Worklight to another . 181

Migrating from IBM Worklight V5.0.6 to V6.0.0 . .	181
Migrating an IBM Worklight project to use the	
Dojo library . . . . .	184
Manually migrating Facebook apps . . . . .	188
Migrating Worklight Studio to V6.0.0 . . . . .	189
Migrating projects to a new Worklight Studio	
instance . . . . .	190
Upgrading Worklight Server in a production	
environment. . . . .	191
Migrating from IBM Worklight V5.0.5 to V5.0.6 . .	224
Dojo iOS fixes . . . . .	228
Dojo 1.8.3 code migration . . . . .	228
Migrating from IBM Worklight V5.0.0.3 to V5.0.5	229

## Chapter 6. Developing IBM Worklight applications . . . . . 233

Worklight Studio overview . . . . .	233
IBM Worklight client-side API overview . . . . .	236
IBM Worklight server-side API overview . . . . .	237
Artifacts produced during development cycle . . .	238

IBM Worklight projects, environments, and skins . . . . .	239	Store internals . . . . .	398
Creating IBM Worklight projects . . . . .	240	JSONStore asynchronicity, callbacks, and promises . . . . .	399
Creating an application in an IBM Worklight project. . . . .	241	Chain JSONStore functions and concurrency . . . . .	399
Creating the client-side of an IBM Worklight application . . . . .	242	JSONStore events . . . . .	400
Integrating with source control systems . . . . .	244	JSONStore errors . . . . .	400
Developing hybrid and web applications . . . . .	246	JSONStore error codes . . . . .	401
Anatomy of an IBM Worklight project . . . . .	246	JSONStore support . . . . .	403
Anatomy of an IBM Worklight application . . . . .	247	JSONStore performance . . . . .	405
Setting up a new IBM Worklight environment for your application . . . . .	262	JSONStore multiple user support . . . . .	406
The Worklight Development Server and the Worklight Console. . . . .	266	JSONStore security . . . . .	407
Working with multiple Worklight Servers in Worklight Studio . . . . .	268	Worklight adapter integration for JSONStore	407
Developing user interface of hybrid applications	278	Troubleshooting JSONStore and data synchronization . . . . .	410
Using IBM Worklight Client API . . . . .	317	Push notification . . . . .	410
Connecting to Worklight Server . . . . .	317	Possible IBM Worklight push notification architectures. . . . .	411
Web and native code in iPhone, iPad, and Android . . . . .	318	Subscribe SMS servlet . . . . .	413
Developing hybrid applications for iOS. . . . .	321	Windows Phone 8 push notifications . . . . .	415
Developing hybrid applications for Android . . . . .	322	Sending push notifications from WebSphere Application Server – IBM DB2. . . . .	416
Developing hybrid applications for BlackBerry	327	IBM Worklight security framework . . . . .	417
Development guidelines for desktop and web environments . . . . .	328	IBM Worklight Security Overview . . . . .	417
Developing native applications . . . . .	331	Security Tests . . . . .	421
Development guidelines for using native API	331	Authentication realms . . . . .	423
Developing native applications for iOS. . . . .	333	Authenticators and Login Modules . . . . .	424
Developing native applications for Android . . . . .	336	The authentication configuration file . . . . .	424
Developing native applications for Java Platform, Micro Edition . . . . .	340	Configuring IBM Worklight web application authorization . . . . .	426
Optimizing IBM Worklight applications . . . . .	342	Configuring authenticators and realms . . . . .	426
Including and excluding application features	343	Basic authenticator . . . . .	427
Application cache management in Desktop Browser and Mobile Web apps . . . . .	346	Form-based authenticator . . . . .	428
IBM Worklight application build settings . . . . .	351	Header authenticator . . . . .	429
Minification of JS and CSS files . . . . .	354	Persistent cookie authenticator . . . . .	429
Concatenation of JS and CSS files. . . . .	356	Adapter authenticator . . . . .	429
Developing the server side of an IBM Worklight application . . . . .	360	LTPA authenticator . . . . .	430
Overview of IBM Worklight adapters . . . . .	360	Configuring login modules . . . . .	431
The adapter XML File . . . . .	364	Non-validating login module . . . . .	432
Creating an IBM Worklight adapter . . . . .	377	Database login module . . . . .	432
Adapter invocation service . . . . .	380	Single identity login module . . . . .	433
Implementing adapter procedures . . . . .	381	Header login module. . . . .	434
Encoding a SOAP XML envelope. . . . .	382	WASLTPAModule login module . . . . .	434
Calling Java code from a JavaScript adapter . . . . .	383	LDAP login module . . . . .	434
Features of Worklight Studio . . . . .	383	Mobile device provisioning. . . . .	436
Procedure invocation . . . . .	388	Configuring and implementing device provisioning. . . . .	439
Invoking a back-end service . . . . .	389	Device single sign-on (SSO) . . . . .	440
Deploying an adapter . . . . .	392	Configuring device single sign-on . . . . .	440
JSONStore overview . . . . .	393	IBM Worklight application authenticity overview . . . . .	441
JSONStore features comparison . . . . .	394	Developing globalized hybrid applications . . . . .	445
Enabling JSONStore . . . . .	395	Globalization in JavaScript frameworks. . . . .	445
JSONStore document . . . . .	396	Globalization mechanisms in IBM Worklight	456
JSONStore collection . . . . .	397	Globalization of web services . . . . .	465
JSONStore store . . . . .	397	Globalization of push notifications . . . . .	466
JSONStore search fields . . . . .	397	Developing accessible applications . . . . .	469
JSONStore queries. . . . .	398	Location services . . . . .	470
		Platform support for location services . . . . .	472
		Triggers . . . . .	472
		Setting an acquisition policy . . . . .	474
		Working with geofences and triggers . . . . .	475



Differentiating between indoor areas . . . . .	477
Securing server resources based on location . . . . .	480
Tracking the current location of devices . . . . .	481
Keeping the application running in the background . . . . .	483
Client-side log capture . . . . .	483
Server preparation for uploaded log data . . . . .	485
Client-side logging in client apps . . . . .	485

## Chapter 7. API reference . . . . . 489

IBM Worklight client-side API . . . . .	489
JavaScript client-side API . . . . .	489
Objective-C client-side API for native iOS apps . . . . .	620
Java client-side API for native Android apps . . . . .	620
Java client-side API for Java ME apps . . . . .	620
IBM Worklight server-side API . . . . .	620
JavaScript server-side API . . . . .	620
Java server-side API . . . . .	653
Internal IBM Worklight database tables . . . . .	653
HTTP Interface of the production server . . . . .	660

## Chapter 8. Deploying IBM Worklight projects . . . . . 663

Deploying IBM Worklight applications to test and production environments . . . . .	663
Deploying an application from development to a test or production environment . . . . .	663
Building a project WAR file with Ant . . . . .	665
Deploying an IBM Worklight project . . . . .	665
Configuration of IBM Worklight applications on the server . . . . .	714
Ant tasks for building and deploying applications and adapters . . . . .	730
Deploying applications and adapters to Worklight Server . . . . .	733
Administering adapters and apps in Worklight Console . . . . .	735
High availability . . . . .	738
Deploying to the cloud by using IBM PureApplication System . . . . .	740
Installing IBM Worklight support for PureApplication System . . . . .	740
Working with the IBM Mobile Application Platform Pattern Type . . . . .	742
Working with IBM Worklight PureApplication System Extension for Worklight Studio . . . . .	747
Building and deploying IBM Worklight virtual applications by using the command line interface . . . . .	748
Deployment of the Application Center on IBM PureApplication System . . . . .	750

## Chapter 9. Administering IBM Worklight applications . . . . . 755

Administering IBM Worklight applications with Worklight Console . . . . .	755
Direct updates of app versions to mobile devices . . . . .	755
Direct updates of app versions to desktop apps . . . . .	757
Locking an application . . . . .	757

Remotely disabling application connectivity . . . . .	758
Displaying a notification message on application startup . . . . .	761
Defining administrator messages from Worklight Console in multiple languages . . . . .	761
Controlling authenticity testing for an app . . . . .	765
Administering push notifications with the Worklight Console . . . . .	767
Application Center . . . . .	769
Concept of the Application Center . . . . .	769
General architecture . . . . .	770
Preliminary information . . . . .	772
Preparations for using the mobile client . . . . .	772
Push notifications of application updates . . . . .	779
The Application Center console . . . . .	782
Command-line tool for uploading or deleting an application . . . . .	797
Publishing Worklight applications to the Application Center . . . . .	802
The mobile client . . . . .	805
Advanced information for BlackBerry users . . . . .	826
Application review feature called from another application (advanced feature) . . . . .	828
Federal standards support in IBM Worklight . . . . .	830
FDCC and USGCB support . . . . .	830
FIPS 140-2 support . . . . .	830

## Chapter 10. Monitoring, reports, and operational analytics . . . . . 833

Logging and monitoring mechanisms . . . . .	833
Vitality queries for checking server health . . . . .	834
Configuring logging in the development server . . . . .	836
Routing logging to Windows event log . . . . .	838
Comparison of operational analytics and reports features . . . . .	839
Operational analytics . . . . .	840
IBM Worklight analytics components . . . . .	841
Analytics event types . . . . .	842
Analytics page . . . . .	843
Capturing client analytic data . . . . .	851
Analytics data flow . . . . .	852
Managing data throughput and accumulation . . . . .	853
Enabling analytics for existing applications . . . . .	855
Logging additional data . . . . .	855
Integrating with the IBM Tealeaf CX Mobile server . . . . .	856
Troubleshooting analytics . . . . .	858
Reports . . . . .	859
Using raw data reports . . . . .	861
Device usage reports . . . . .	865
Predefined BIRT Reports . . . . .	867
Installing BIRT on Apache Tomcat . . . . .	869
Installing BIRT on WebSphere Application Server Liberty Profile . . . . .	871
Installing BIRT on WebSphere Application Server Full Profile . . . . .	873
Configuring BIRT reports for your application server by using Ant . . . . .	874
Manually configuring BIRT Reports for your application server . . . . .	875
BIRT in Eclipse . . . . .	876

Notification reports database schema . . . . . 878

**Chapter 11. Integrating with other IBM Mobile Foundation products . . . . . 881**

Introduction to IBM Worklight integration options . . . . . 881  
Integration with Cast Iron . . . . . 882  
Integration with reverse proxy. . . . . 883  
    Authentication at the gateway. . . . . 884  
IBM Endpoint Manager for Mobile Devices  
overview . . . . . 885  
Managing end points with IBM Endpoint Manager . . . . . 887  
Useful links . . . . . 888

**Chapter 12. Migrating from the WebSphere Application Server Feature Pack . . . . . 889**

Migration scenarios . . . . . 889  
Migrating an application that uses the client programming model . . . . . 889  
Migrating an application that uses the server programming model . . . . . 890  
Considerations for applications that use JAX-RS, JSON-RPC, or proxying . . . . . 891  
Example: Migrating the Dojo showcase sample . . . . . 891

**Chapter 13. Troubleshooting and known limitations . . . . . 893**

**Chapter 14. Glossary . . . . . 895**

A . . . . . 895

B . . . . . 896  
C . . . . . 896  
D . . . . . 897  
E . . . . . 897  
F . . . . . 897  
G . . . . . 898  
H . . . . . 898  
I . . . . . 898  
K . . . . . 898  
L . . . . . 898  
M . . . . . 899  
N . . . . . 899  
P . . . . . 899  
R . . . . . 900  
S . . . . . 900  
T . . . . . 901  
V . . . . . 901  
W . . . . . 901

**Chapter 15. Notices. . . . . 903**

**Chapter 16. Support and comments . . . . . 907**

**Index . . . . . 911**

---

## Chapter 1. Overview of IBM Worklight

Start here to learn about IBM® Worklight® product.

With IBM Worklight, you can extend your business to mobile devices. IBM Worklight provides an open, comprehensive, and advanced mobile application platform for smartphones and tablets. It helps organizations of all sizes to efficiently develop, test, connect, run, and manage mobile and applications. Using standards-based technologies and tools, IBM Worklight provides an integrated platform that includes a comprehensive development environment, mobile-optimized runtime middleware, a private enterprise application store, and an integrated management and analytics console, all supported by various security mechanisms.

Read the following topics to learn more IBM Worklight.

---

### Introduction to mobile application development

With IBM Worklight, you can develop mobile applications by using any of four different approaches: web development, hybrid development, hybrid mixed development, and native development.

IBM Worklight provides capabilities to help you respond to the fast-paced development of mobile devices. IBM Worklight is based on open standards such as HTML, CSS, or JavaScript, and solutions such as Apache Cordova, Eclipse Foundation, or the Android and Apple SDKs, for delivering mobile solutions. This gives you more flexibility when you implement your mobile communication channel, or release a new version of your application. You can evaluate what the best approach is for each case, according to skills, time, and functionality, without being limited by a specific approach to mobile application development.

Four approaches exist to develop your mobile applications:

#### Web development

By using the web development approach, your application runs inside the browser of the mobile device, and uses standard technologies such as HTML5, CSS3, and JavaScript. Your application is platform independent, so you do not need to develop a new application to support a new mobile platform. Modifications to your application might be required to support a different browser engine. Mobile web applications cannot access the platform functions because they rely only on the browser, and the associated web standards. Mobile web applications are not distributed through an application store. They are accessed through a link on a website, or a bookmark in the mobile browser of the user.

#### Hybrid development

With the hybrid development approach, you can create applications that use parts of both the native development and web development approaches. Your hybrid application runs inside a native container and uses the browser engine to display the application interface, which is based on HTML and JavaScript. With the native container, your application can access device capabilities that are not accessible to web applications, such as the accelerometer, camera, and local storage on a

smartphone. Similar to native applications, hybrid applications are distributed through the application store of the platform.

## Hybrid mixed development

You can enhance the hybrid development approach with hybrid mixed development. With the hybrid mixed development approach, you can create applications that use a container to access device capabilities, but also use other native, platform-specific components such as libraries, or specific user-interface elements to enhance the mobile application.

## Native development

With the native development approach, you can create applications that are written for a specific platform and runs on that platform only. Your applications can fully use all platform functions such as accessing the camera or contact list, or interacting with other applications on the device. To support platforms such as Android, iOS, BlackBerry, and Windows Phone, you must develop separate applications with different programming languages, such as Objective-C for iOS, or Java™ for Android. Typically, native applications are distributed through an application store.

## Aspects of each development approach

Each of these development approaches has advantages and disadvantages. You must select the appropriate development approach according to the specific requirements for an individual mobile solution. This choice depends heavily on the specificities of your mobile application and its functional requirements. Mapping your requirements to select an appropriate development approach is the first step in a mobile development project. Table 1 outlines the major aspects of the four development approaches, and can help you decide which development approach is appropriate for your specific mobile application.

Table 1. Comparison of mobile development approaches

Aspect	Web development	Hybrid development	Hybrid mixed development	Native development
Easy to learn	Easy	Medium	Medium	Hard
Application performance	Slow	Moderate	Moderate	Fast
Device knowledge required	None	Some	Some	A lot
Development lifecycle (build/test/deploy)	Short	Medium	Medium	Long
Application portability to other platforms	High	High	Medium	None
Support for native device functionality	Some	Most	All	All

Table 1. Comparison of mobile development approaches (continued)

Aspect	Web development	Hybrid development	Hybrid mixed development	Native development
Distribution with built-in mechanisms	No	Yes	Yes	Yes
Ability to write extensions to device capabilities	No	Yes	Yes	Yes

## Overview of IBM Worklight main capabilities

With IBM Worklight, you can use capabilities such as development, testing, back-end connections, push notifications, offline mode, update, security, analytics, monitoring, and application publishing.

### Development

IBM Worklight provides a framework that enables the development, optimization, integration, and management of secure applications that run on smartphones and other consumer environments. IBM Worklight does not introduce a proprietary programming language or model that users must learn. You can develop apps by using HTML5, CSS3, and JavaScript. You can optionally write native code (Java or Objective-C), and IBM Worklight provides an SDK that includes libraries that you can access from native code.

### Testing

IBM Worklight includes IBM Mobile Test Workbench for Worklight for testing mobile applications. With the mobile testing capabilities of IBM Mobile Test Workbench for Worklight, you can automate the creation, execution, and analysis of functional tests for IBM Worklight native and hybrid applications on Android and iOS devices.

### Back-end connections

Some mobile applications run strictly offline with no connection to a back-end system, but most mobile applications connect to existing enterprise services to provide the critical user-related functions. For example, customers can use a mobile application to shop anywhere, at any time, independent of the operating hours of the store. Their orders must still be processed by using the existing e-commerce platform of the store. To integrate a mobile application with enterprise services, you must use middleware such as a mobile gateway. IBM Worklight can act as this middleware solution and make communication with back-end services easier.

### Push notifications

With push notifications, mobile applications can send information to mobile devices, even when the application is not being used. IBM Worklight includes a unified notification framework that provides a consistent mechanism for such push notifications. With this unified notification framework, you can send push notifications without having to know the details of each targeted device or

platform, because each mobile platform has a different mechanism for these push notifications.

## Offline

In terms of connectivity, mobile applications can operate offline, online, or in a mixed mode. IBM Worklight uses a client/server architecture that can detect whether a device has network connectivity, and the quality of the connection. Acting as a client, mobile applications periodically attempt to connect to the server and to assess the strength of the connection. An offline-enabled mobile application can be used when a mobile device lacks connectivity but some functions can be limited. When you create an offline-enabled mobile application, it is useful to store information about the mobile device that can help preserve its functionality in offline mode. This information typically comes from a back-end system, and you must consider data synchronization with the back end as part of the application architecture. IBM Worklight includes a feature called JSONStore for data exchange and storage, that provides the ability to create, read, update, and delete data records from a data source. Each operation is queued when operating offline. When a connection is available, the operation is transferred to the server and each operation is then performed against the source data.

## Update

IBM Worklight simplifies version management and mobile application compatibility. Whenever a user starts a mobile application, the application communicates with a server. By using this server, IBM Worklight can determine whether a newer version of the application is available, and if so, give information to the user about it, or push an application update to the device. The server can also force an upgrade to the latest version of an application to prevent continued use of an outdated version.

## Security

Protecting confidential and private information is critical for all applications within an enterprise, including mobile applications. Mobile security applies at various levels, such as mobile application, mobile application services, or back-end service. You must ensure customer privacy and protect confidential data from being accessed by unauthorized users. Dealing with privately owned mobile devices means giving up control on certain lower levels of security, such as the mobile operating system.

IBM Worklight provides secure, end-to-end communication by positioning a server that oversees the flow of data between the mobile application and your back-end systems. IBM Worklight gives you the possibility to define custom security handlers for any access to this flow of data. Because any access to data of a mobile application has to go through this server instance, you can define different security handlers for mobile applications, web applications, and back-end access. With this kind of granular security, you can define separate levels of authentication for different functions of your mobile application or avoid sensitive information being accessed from a mobile application entirely.

## Analytics

The operational analytics feature enables searching across apps, services, devices, and other sources to collect data about usage or detect problems.



In addition to reports summarizing app activity, IBM Worklight includes a scalable operational analytics platform accessible in the Worklight Console. The analytics feature enables enterprises to search across logs and events that are collected from devices, apps, and servers for patterns, problems, and platform usage statistics. You can enable analytics, reports, or both, depending on your needs.

## Application publishing

IBM Worklight Application Center is an enterprise application store. With the Application Center, you can install, configure, and administer a repository of mobile applications for use by individuals and groups across your enterprise. You can control who in your organization can access the Application Center and upload applications to the Application Center repository, and who can download and install these applications on to a mobile device. You can also use the Application Center to collect feedback from users and access information about devices on which applications are installed.

The concept of the Application Center is similar to the concept of the Apple public App Store or the Google Play store, except that it targets the development process. It provides a repository for storing the mobile application files and a web-based console for managing that repository. It also provides a mobile client application to allow users to browse the catalog of applications that are stored by the Application Center, install applications, leave feedback for the development team, and expose production applications to IBM Endpoint Manager. Access to download and install applications from the Application Center is controlled by using access control lists (ACLs).

---

## Introducing IBM Worklight components

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

### 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 an integrated development environment for mobile applications, 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.



- 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.

## Client-side 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 client-side runtime components provide the following functions:

- Mobile data integration: connectivity and authentication APIs
- Security features: on-device encryption, offline authentication, and remote disablement of the ability to connect to Worklight Server
- 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

- 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 applications.

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 the ability to connect to Worklight Server 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 use 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, the Apple iOS platform, and the BlackBerry OS 6 and 7 platform, but does not target mobile web applications. BlackBerry OS 10 is not supported by the current version of the Application Center.

## 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

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.

---

## IBM Worklight editions

IBM Worklight and IBM Mobile Foundation are available to you in different editions.

### IBM Worklight Developer Edition

IBM Worklight Developer Edition is a free, 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 the same Worklight Studio functions that are available in the IBM Worklight Consumer Edition and the IBM Worklight Enterprise Edition, except for some security-related features. Support is only provided as a best-effort service. For full product support, choose the IBM Worklight Consumer Edition or the IBM Worklight Enterprise Edition.

The P2 Eclipse update version of IBM Mobile Test Workbench for Worklight is also downloadable for free as a separate plug-in from the developerWorks website. After you install the IBM Worklight Developer Edition, you can then optionally install the test workbench in Eclipse.

### IBM Worklight Consumer Edition and IBM Worklight Enterprise Edition

IBM Worklight Consumer Edition and IBM Worklight Enterprise Edition are identical programs that differ in license only. These programs are supported through an IBM International License Agreement and available from IBM Passport Advantage®. The IBM Worklight Consumer Edition and the IBM Worklight Enterprise Edition contain:

- A separate Worklight Studio component, which is available as an Eclipse plug-in (Eclipse P2 install)
- A separate Worklight Server component, which is available as an Installation Manager package
- A separate, optional, IBM Mobile Test Workbench for Worklight component, which is available as an Eclipse plug-in (Eclipse P2 install)

**Note:** When you install and upgrade IBM Worklight Consumer Edition or IBM Worklight Enterprise Edition, you must make sure that the version numbers of your Worklight Studio and Worklight Server components stay in sync. You must not mix components across releases.

---

## System requirements for using IBM Worklight

Identify the operating systems, Eclipse versions, SDKs, and other software that are required or supported by IBM Worklight, and features in IBM Worklight that are available for each supported platform.

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 needed hardware configuration
- 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 application servers, database management systems, and other software that are required or supported by IBM Worklight

---

## Matrix of features and platforms

IBM Worklight provides many features and supports many platforms.

The technote at <http://www.ibm.com/support/docview.wss?uid=swg27039422> lists the IBM Worklight features that are available on each of the platforms that IBM Worklight supports.

---

## Known limitations

General limitations apply to IBM Worklight as detailed here. Limitations that apply to specific features are explained in the topics that describe these features.

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.

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

### Globalization

If you are developing globalized apps, notice 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.

The analytics platform has the following limitations in terms of globalization:

- In reports, the format for dates and times do not follow the International Components for Unicode (ICU) rules.
- In reports, the format for numbers do not follow the International Components for Unicode (ICU) rules.
- In reports, the numbers do not use the user's preferred number script.
- In reports, searching for Chinese, Japanese, and Korean characters (CJK) returns no results.
- Messages that include non-ASCII characters and that are created with "WL.Analytics.log" on page 492 or with "WL.Logger.error" on page 588 are not always logged successfully.
- The Analytics page of the Worklight Console does not work in the following browsers:
  - Microsoft Internet Explorer version 8 or earlier
  - Apple Safari on iOS version 4.3 or earlier
- On Mozilla Firefox browser and Google Chrome browser, the locale that is used to display dates and time might differ from the locale that is set for the browser.
- The dates on the X-axis are not localized.

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

For example:

- You must define the user name and password of the Application Center with ASCII characters only. This limitation exists because IBM WebSphere® Application Server (full or Liberty profiles) does not support non-ASCII passwords and user names. See [http://pic.dhe.ibm.com/infocenter/wasinfo/v8r0/topic/com.ibm.websphere.nd.doc/info/ae/ae/csec\\_chars.html](http://pic.dhe.ibm.com/infocenter/wasinfo/v8r0/topic/com.ibm.websphere.nd.doc/info/ae/ae/csec_chars.html).
- To deploy a virtual application, you must define the user name and ID in the corresponding Worklight configuration component with ASCII characters only. This limitation exists because IBM PureApplication System does not support non-ASCII passwords and user names.
- On Windows:
  - To see any localized messages in the log file that the Worklight Test Server that is embedded in Worklight Studio creates, you must open this log file with the UTF8 encoding.
  - In the Worklight Test Server console in Eclipse, the localized messages are not properly displayed.

These limitations exist because of the following causes:

- The Worklight Test Server is installed on IBM WebSphere Application Server Liberty Profile, which creates log file with the ANSI encoding except for its localized messages for which it uses the UTF8 encoding.
- The Worklight Test Server console in Eclipse displays the content by using the ANSI encoding, not the UTF8 encoding.

## Application Center mobile client

The Application Center mobile client follows the cultural conventions of the running device, such as the date formatting. It does not always follow the stricter International Components for Unicode (ICU) rules.

The Application Center mobile client for BlackBerry has the following known limitations:

- It supports only a limited set of languages. In particular, it does not fully support right-to-left languages, such as Arabic and Hebrew.
- It does not support Unicode characters outside the Basic Multilingual Plane.
- It supports Unicode characters inside the Basic Multilingual Plane but how these characters are displayed depends on the fonts that are available on the device

## Application Center mobile client: refresh issues on Android 4.0.x

Android 4.0.x WebView component is known to have several refresh issues. Updating devices to Android 4.1.x should provide a better user experience.

If you build the Application Center client from sources, disabling the hardware acceleration at the application level in the Android manifest should improve the situation for Android 4.0.x. In that case, the application must be built with Android SDK 11 or later.

## 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.

## JSONStore resources for iPhone and iPad

When you develop apps for iPhone and iPad, the JSONStore resources are always packaged in the application, regardless of whether you enabled JSONStore or not in the application descriptor. The application size is not reduced even if JSONStore is not enabled.

## Analytics page of the Worklight Console

Response times in the Analytics page of the Worklight Console are dependent on several factors, such as hardware (RAM, CPUs), quantity of accumulated analytics data, and IBM Websphere Analytics Platform (IWAP) clustering. Consider testing



your load prior to integrating IWAP into production.

## Deployment of an app from Worklight Studio to Tomcat

If you use Tomcat as an external server in Eclipse (for example to test and debug the applications directly in Worklight Studio), the following restrictions apply:

- The context path that you set to your project is ignored. When you deploy your app from Worklight Studio to Tomcat, the default context path, which is the project name, is used instead of the context path. The URL of the Worklight Console for your app similarly uses the project name.
- When you deploy your app from Worklight Studio to Tomcat, the deployed WAR file is not visible in the **Server** view of Eclipse (in Worklight Studio), even if the application is correctly deployed.

To avoid these issues, keep the default value of the context path of your project, which is the project name.

## Installation on a cluster of IBM WebSphere Application Servers Liberty Profile, that you administer with a collective controller

The following limitations apply if you install Worklight Server on a cluster of IBM WebSphere Application Servers, Liberty Profile, that you administer with a collective controller:

- The Application Center installation with the Worklight Server installer does not use the collective controller. You must install Worklight Server on each server separately.
- The Worklight console installation with the `<configureApplicationServer>` Ant task does not use the collective controller. You must run the `<configureApplicationServer>` Ant task for each server separately.

## Support for Android Emulator 2.3.x

IBM Worklight does not support Android Emulator 2.3.x because of known issues, as detailed in the Android list of issues at <https://code.google.com/p/android/issues/list> (search for issue 12987).



---

## Chapter 2. What's new

This section details the new features and changes in IBM Worklight V6.0.0 and subsequent fix packs.

---

### What's new in IBM Worklight V6.0.0.2

IBM Worklight V6.0.0.2 fixes many problems that were identified in previous versions.

#### Client-side log collection

Starting with IBM Worklight V6.0.0.2, you can now capture and receive uploaded client-side logs. For more information, see “Client-side log capture” on page 483.

#### 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.
- 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 509 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 758.

## 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 600.

## 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 495.

## List of fixes for IBM Worklight V6.0.0.2

For a complete list of issues that are fixed in IBM Worklight V6.0.0.2, see Version 6.0.0 Fix Pack 2.

---

## What's new in IBM Worklight V6.0.0.1

IBM Worklight V6.0.0.1 fixes many problems that were identified in previous versions.

For more information about the fixed issues, see Version 6.0.0 Fix Pack 1.

---

## What's new in IBM Worklight V6.0.0

This section details the new features and changes in IBM Worklight V6.0.0 compared to the previous version of this product.

### New functional testing capabilities

IBM Worklight V6.0.0 offers improved ability to create, run, and automate tests on mobile applications.

The accelerated delivery cycles of mobile applications requires fast and effective test cycles. IBM Worklight V6.0.0 now includes IBM Mobile Test Workbench for Worklight (IMTWW), which is an automated functional testing feature for Android and iOS native and hybrid applications.

Designed for use by developers and testers, IBM Mobile Test Workbench for Worklight automates functional testing. First, developers or testers record a sequence of actions on a mobile device, using a recording-ready app to generate a test script. Next, they edit and enhance the script using natural language syntax to add verification points and other instructions. Then, they can run the enhanced test script on a real device, simulator, or emulator. Results can be viewed and shared via a generated HTML report.

With IBM Mobile Test Workbench for Worklight, you can create, run, and automate tests on mobile applications. The main new key features are as follows:

- Seamless experience: You create, run, and automate tests on mobile applications in Worklight Studio, that is, the same Eclipse environment that you also use to develop your apps.
- Support for the most common needs: You can test native or hybrid apps for the two leading mobile platforms: Android and iOS.

- Complete: You can record your tests, edit your test definitions, and run your tests on mobile devices and on simulators.
- Support for all JavaScript frameworks (with an additional support for JQuery Mobile components that are detected and registered as such in the test script) for building and for testing apps.
- Natural scripting language: You can define test scripts with a natural language that non-IT people can understand.

As a developer, you can use Worklight Studio to test your apps with IBM Mobile Test Workbench for Worklight. For more information about functional testing within Worklight Studio, see *Testing with IBM Worklight*

As a developer, you can share apps that are ready for tests with IBM Mobile Test Workbench for Worklight with the testing team by using Application Center. For more information, see *Using the Application Center and the Mobile Test Workbench to share applications*.

## New operational analytics feature

With IBM Worklight V6.0.0, you can now use the operational analytics feature to search across apps, services, devices, and other sources to collect data about usage or to detect problems.

In addition to BIRT reports, which are still supported, IBM Worklight V6.0.0 now also includes a scalable operational analytics platform that you can access from the Worklight Console. The analytics feature enables enterprises to search across logs and events that are collected from devices, apps, and servers for patterns, problems, and platform usage statistics. You can enable analytics, reports, or both, depending on your needs.

Information that is accessible for operational analytics includes data from the following sources:

- Crash events of an app on iOS and Android devices.
- Interactions of any app-to-server activity. This data includes anything that is supported by the IBM Worklight client/server protocol, including push notifications.
- Server-side logs that are captured in existing IBM Worklight log files.

**Note:** If location services are enabled in your IBM Worklight application, geolocation and WiFi data is automatically included in captured operational analytics data. For more information about location services, see “Location services” on page 470

For more information, see “Operational analytics” on page 840.

To get started with the operational analytics feature of IBM Worklight, see the module *Operational Analytics*, under category 10, *Moving to production*, in Chapter 3, “Tutorials and samples,” on page 25.

## Enhancing applications with location features

IBM Worklight V6.0.0, contains new location capabilities to enhance your apps.

Location is a powerful differentiator of mobile apps. Yet because location coordinates must be constantly polled to understand where a mobile device is

located, the resulting stream of geographic information can be difficult to manage without exhausting resources such as battery and network.

IBM Worklight V6.0.0 now defines location services to respond to geo events. These services handle multiple geo modalities such as GPS, WiFi sampling and interpolation, with policies for acquiring geo data and sending it in batch to optimize battery and network usage.

With the geolocation toolkit, business actions can be triggered when users reach a point of interest, or enter or exit a region (geo-fencing), and server-side logic can be run to react to important geo events.

For more information about the new location features in IBM Worklight V6.0.0, see “Location services” on page 470 and “Location services API” on page 553 for the corresponding API.

To get started with the location features of IBM Worklight, see the module *Location services*, under category 9, *Advanced topics*, in Chapter 3, “Tutorials and samples,” on page 25.

## Enhanced rapid application development

With IBM Worklight V6.0.0, you can use new features to accelerate the development of your hybrid applications, including templates, and improved Dojo and JavaScript tools.

### Screen templates

Delivering an outstanding mobile UI experience requires conformance with continuously evolving mobile patterns of behavior specific to each OS family. IBM Worklight V6.0.0 now includes screen templates that automate the creation of mobile screens that reflect design best practices. Developers can choose from templates in four categories: Lists, Authentication, Navigation & Search, and Configuration. Each screen template can be previewed, used as provided, or further refined by using any combination of web and native technologies

For more information about screen template, see “Mobile patterns” on page 301.

### Improved Dojo tools

IBM Worklight V6.0.0 provides tools to improve the build of IBM Worklight applications with Dojo. In particular, here are the changes that are now available:

- Dojo source files are moved out of the IBM Worklight project into a separate Dojo Library project, which is on an internal Jetty server. Moving the Dojo files to a separate project keeps the IBM Worklight project light and easy to work with. The Dojo Library feature is where access to the full distribution of Dojo is initiated by linking to the internal server.

An IBM Worklight project that uses Dojo is paired with a Dojo Library project. An IBM Worklight project that you created with a previous version of IBM Worklight has a mobile version of Dojo that is placed directly in it. Starting with IBM Worklight V6.0.0, your projects now embed a subset of Dojo resources, and also have a separate Dojo Library project.

- Dojo Mobile applications that are built by using Worklight are created by using an optimized set of layers to help the quick development and deployment of your application. The pre-built layers are configured to support numerous locales; however, the application includes the locale for US English only, upon

project creation. On some mobile devices, if the locale is configured to anything other than US English, the application can crash due to a defect in Dojo. To correct this, supported locales must be added to the application. The NLS files can be found in your Dojo Library project's `toolkit/dojo/dojo/nls` folder, and they must be copied into your mobile application project's `www/dojo/nls` folder. After this change, the application can then be deployed to the device with various locale support.

- Developers can focus on coding without having to evaluate which JavaScript modules they must bring into the project from the Dojo Library project before production deployment. The development environment provides full access to the Dojo widgets and API. Access is provided even if the JavaScript modules are not included in the IBM Worklight project. Developers can test the application without having to copy JavaScript modules into the IBM Worklight project, but rely on the Dojo Library project to supply the necessary modules. When developers test on an external device or emulator, IDE must be running and must have Internet connectivity. If the Dojo Library project's IP changes, then the app cannot access resources from the Dojo Library until it is rebuilt.

For more information about improved Dojo tools, see “Using JavaScript toolkits” on page 279.

### AMD module locator

IBM Worklight V6.0.0 provides improved JavaScript tools for you to locate the Asynchronous Module Definition (AMD) module that contains the API that you need. Type a few characters of the API that you require, and the locator identifies the AMD module that you must add to `require()`.

## Features to optimize IBM Worklight applications

With IBM Worklight V6.0.0, you can use a number of new features to reduce the size of your apps or enable them to start more quickly.

During development, the applications you develop with Worklight Studio can demonstrate good performance. But when these apps are run on mobile devices, performance can be impacted by a number of factors. Worklight Studio V6.0.0 includes a number of features that can reduce the size of your IBM Worklight apps, or reduce the number of resources that must be loaded at start time. Using these features can improve both client-side performance and user satisfaction.

To get started with these features of IBM Worklight to optimize your apps, see the module *General information when developing Mobile Web applications* and the module *General information when developing desktop applications*, under category 3, *Worklight client-side development basics*, in Chapter 3, “Tutorials and samples,” on page 25.

### Including and excluding application features

Application features such as JSONStore offer many benefits, if used in the application. If it is not used, however, the JSONStore resources greatly increase the size of your applications, in particular if they are simple, and thus increase their initial download time and start time.

In IBM Worklight V6.0.0, you can choose which features to include in the build by editing the `application-descriptor.xml` file of your application with the Worklight Studio Application Descriptor editor. This selection is reflected in the new `<features>` element in the Application Descriptor. This element enables users to



control the inclusion or exclusion of certain application resources. For more information about this feature, see “Including and excluding application features” on page 343.

**Note:** In IBM Worklight V6.0.0, only JSONStore resources on Android applications can be excluded with the <features> element.

## Controlling the application cache

Ideally, developers want mobile web applications to work when the user is offline. The release of HTML5 addressed this need with the introduction of the *application cache*, which provides improvements in offline browsing, speed, and reduced server load. The *Cache Manifest* is a text file that lists the resources that the browser is to cache for offline access. It contains a list of resources that are explicitly cached after the first time they are downloaded.

In IBM Worklight V6.0.0, users can view and edit the Cache Manifest for web environments (Desktop Browser and Mobile Web applications). For more information about this feature, see “Application cache management in Desktop Browser and Mobile Web apps” on page 346.

## Minification of application resources

*Minification* is a process that minifies web resources to make them smaller. The smaller size of the resources means less traffic between the Worklight application and Worklight Server, and can improve initial download time and application start time.

In Worklight V6.0.0, you can minify certain Mobile Web and Desktop Browser resources (JavaScript and CSS files) at build time, with the Google Closure Compiler. For more information about this feature, see “Minification of JS and CSS files” on page 354.

## Concatenation of application resources

In IBM Worklight V6.0.0, a new feature enables users to concatenate multiple web resources that are used by Mobile Web and Desktop Browser applications into a smaller number of files. By reducing the total number of JavaScript and Cascading Style Sheet files that are referenced by the application HTML, fewer browser requests are required. As a result, the application can start more quickly. For more information about this feature, see “Concatenation of JS and CSS files” on page 356.

## Controlling your build settings to improve the app start time

In IBM Worklight V6.0.0, a new file is added to Worklight applications when they are first created: `build-settings.xml`. The purpose of the file is to prepare minification and concatenation configurations for each Mobile Web and Desktop Browser environment. These configurations are then used by the minify and concatenation engines during the build process.

You can view and edit these build settings from within Worklight Studio, and use them to create appropriate configurations for different stages of the development cycle. For example, during development you can choose not to minimize or concatenate your web resources in each build. But when you move the application to production, you can enable minification and concatenation in the builds to

reduce application size or improve start time. For more information about this feature, see “IBM Worklight application build settings” on page 351.

## Improved serviceability

IBM Worklight V6.0.0 improves how you deploy or configure your applications

### No IBM Worklight project installed by default, no Worklight Console available by default

**Important:** In IBM Worklight V6.0.0, the installation process no longer installs a default IBM Worklight project, and so no longer creates the corresponding IBM Worklight Console by default.

Before you can open a Worklight Console, you must therefore first install IBM Worklight Server, which copies the deployment tools and production libraries on your computer, and then deploy a Worklight project WAR file as detailed in Chapter 8, “Deploying IBM Worklight projects,” on page 663.

### Improved deployment process

With IBM Worklight V6.0.0, you can now perform the following actions:

- Deploy several project WAR files to the same application server.  
Each IBM Worklight project can be connected to a different database or to the same database with distinct data storage area.  
To get started with the deployment of several IBM Worklight project WAR files to the same server, see the module *Creating your first Worklight application*, under category 2, *Hello Worklight*, in Chapter 3, “Tutorials and samples,” on page 25.
- Create a single IBM Worklight project WAR file and use it for test or production environment without the need to repackage it. You only have to customize the project WAR file by using JNDI properties.  
For more information about how to configure a project WAR file with JNDI properties, see “Configuring an IBM Worklight project in production by using JNDI environment entries” on page 723.
- Use Ant tasks to create and configure the IBM Worklight databases on the supported database types.  
For more information about these Ant tasks, see “Creating and configuring the databases with Ant tasks” on page 669.
- Use Ant tasks to deploy the IBM Worklight project WAR file to the application server and configure the application server.  
For more information about these Ant tasks, see “Deploying a project WAR file and configuring the application server with Ant tasks” on page 694.

### Improved Application Center

With IBM Worklight V6.0.0, Application Center contains the following improvements:

- The Application Center mobile client for iOS and Android is new, with additional functions:
  - Ability to manage a list of favorite applications per user.
  - Ability to sort applications by name, popularity, or last update date.
  - Ability to see more details on the application, including rating and reviews.



For more information about the Application Center mobile client, see “The mobile client” on page 805.

- Users can be notified of new application update through push notification.
- The list of application on the catalog is filtered according to the capabilities of the device (OS level, iPad versus iPhone).
- The Application Center now supports federated repositories on IBM WebSphere Application Server 7 for the Access Control List (ACL) management. When installing Application Center on WebSphere Application Server 7, you can now configure the Access Control List by using the Virtual Member Manager (VMM)
- You can configure LDAP through VMM for WebSphere Application Server (full profile version 7)
- You can use Application Center to share applications that are instrumented by the IBM Mobile Test Workbench for Worklight for testing purpose, as introduced in “New functional testing capabilities” on page 14.
- The Ant task to deploy applications to Application Center has been improved with the ability to provide more metadata on the app such as the description. You can also remove apps from the catalog.

For more information about the Application Center, see “Application Center” on page 769.

## Enhanced IBM Worklight API

IBM Worklight V6.0.0 enhances and extends the API that you can use to develop mobile applications.

**Important:** Some of the changes that are listed here 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 from IBM Worklight V5.0.6 to V6.0.0” on page 181.

### Support of updated libraries and toolkits

IBM Worklight V6.0.0 supports recent versions of the following libraries, which you can use during the development of your apps:

- Apache Cordova platform
- Dojo toolkit
- jQuery library
- jQuery Mobile library
- Sencha Touch framework

For more information, see “System requirements for using IBM Worklight” on page 8.

To get started with using Apache Cordova in IBM Worklight, see the module *Apache Cordova overview*, under category 6, *Adding native functionality to hybrid applications with Apache Cordova*, in Chapter 3, “Tutorials and samples,” on page 25.

### Updated JavaScript client-side API

IBM Worklight V6.0.0 updated its JavaScript client-side API. Notice, in particular, the following changes:

- JSONStore now provides an API-compatible implementation that runs in supported browsers. With this feature, developers can create applications that use the JSONStore API on web-only environments, before they move to the production environments on iOS and Android.  
For more information, see “JSONStore overview” on page 393.
- The new Analytics API is defined for you to enable and disable data collection, reset the analytics feature configuration, and log analytics data during IBM Worklight application run time.  
For more information, see “WL.Analytics” on page 491.
- The WL.Logger API is redesigned with new features, such as packages, log levels, filtering, string output, pretty print objects, level and package tags, optional callback, stack traces.  
For more information, see “WL.Logger methods” on page 584 and “Configuring the Worklight Logger” on page 574.
- The new client-side Location API is defined for you to use Geo and WiFi locations to perform various actions in your apps.  
For more information, see “Location services API” on page 553.

### Updated JavaScript server-side API

IBM Worklight V6.0.0 complements its JavaScript server-side API with further elements that you can use to extend the Worklight Server. Notice, in particular, the following changes:

- The new server-side Location API is defined, as a complement to the client-side Location API:
  - WL.Server.createEventHandler
  - WL.Server.getClientDeviceContext
  - WL.Server.logActivity
  - WL.Server.setApplicationContext
  - WL.Server.setEventHandlers

For more information, see “JavaScript server-side API” on page 620.

### Updated Objective-C client-side API for iOS

IBM Worklight V6.0.0 adds the following method to its Objective-C client-side API to develop native apps on iOS:

- The method `setHeartbeatInterval` that sets the interval at which the client (device) sends a heartbeat signal to the server. You use this heartbeat signal to prevent a session with the server from timing out because of inactivity.

For more information, see “Objective-C client-side API for native iOS apps” on page 620.

### Updated Java client-side API for Android

IBM Worklight V6.0.0 adds the following method to its Java client-side API to develop native apps on Android:

- The method `setHeartBeatInterval` that sets the interval at which the client (device) sends a heartbeat signal to the server. You use this heartbeat signal to prevent a session with the server from timing out because of inactivity.

For more information, see “Java client-side API for native Android apps” on page 620.

**Note:** IBM Worklight V6.0.0 includes no changes for the following APIs:

- Java client-side API for Java Platform, Micro Edition (Java ME)
- Java server-side API

## **Augmented support of mobile environments and operating systems**

IBM Worklight V6.0.0 offers extended capabilities in terms of supported mobile environments and operating systems

### **Updated operating systems, hardware, and software support**

The list of operating systems, hardware, and software supported by IBM Worklight V6.0.0 has changed. For more information about this list, see “System requirements for using IBM Worklight” on page 8.

### **Deprecated or removed environments**

With IBM Worklight V6.0.0, the following environment is now deprecated:

- Windows Phone 7.5

With IBM Worklight V6.0.0, the following environments are now removed:

- iGoogle
- Mac OS/X Dashboard
- Windows 7 / Vista Gadgets
- Facebook

### **Alignment of the Push function between mobile environments**

With IBM Worklight V6.0.0, you can now use the Push function also on the Windows Phone 8 environment, while it was available only on other environments in previous versions.

For more information about Push for Windows Phone 8, see “Windows Phone 8 push notifications” on page 415.

### **Support for more versions of hosts**

With IBM Worklight V6.0.0, you can now run Worklight Server on the following hosts

- WebSphere Application Server 8.5.5
- WebSphere Application Server ND 8.5.5

## **Miscellaneous modifications**

IBM Worklight V6.0.0 includes other miscellaneous modifications, including changes in project files and behaviors, changes in the development test server and the Worklight Console.

## Changes in project files and behaviors

In IBM Worklight V6.0.0, 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 from IBM Worklight V5.0.6 to V6.0.0” on page 181.

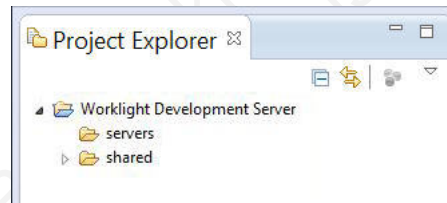
## Changes in the installation process of Worklight Studio

Before IBM Worklight V6.0.0, you installed Worklight Studio Consumer Edition and Worklight Studio Enterprise Edition either with IBM Installation Manager, or with P2 Eclipse update. Since IBM Worklight V6.0.0, IBM Installation Manager for Worklight Studio is no longer provided. All editions are installed with P2 Eclipse update only.

## Changes to the development server and Worklight Console

In previous versions of IBM Worklight, the embedded server that is used during development was provided by Jetty. In IBM Worklight V6.0.0, the Jetty server is replaced with an embedded instance of WebSphere Application Server Liberty Profile. This Liberty profile server is installed with Worklight Studio, and becomes the default test server.

As a result, you see a new **Worklight Development Server** element in your Eclipse Project Explorer view, even before you begin creating new projects and working with them.



A new menu option named **Open Worklight Console** opens an IBM Worklight Console window in your default web browser, in which you can view projects that are deployed to this test server. There are also changes to the URL used to start this default Worklight Console in the browser itself because this URL now must include the Worklight project name to define a *root context*. For more information about these changes, see “The Worklight Development Server and the Worklight Console” on page 266.

## Changed location of PureApplication configuration

In IBM Worklight V6.0.0, IBM Worklight PureApplication System Extension for Worklight Studio copies the PureApplication configuration into the `WorklightServerConfig` folder under your Eclipse workspace. In previous releases, the PureApplication configuration was copied into the `WorklightServerHome` folder.

## Integration with Firebug in Worklight Studio no longer available

In IBM Worklight V6.0.0, Worklight Studio no longer integrates with Firebug. The following functions are removed:

- Connecting to a Firebug JavaScript debugger from an Eclipse debug session.
- Viewing the JavaScript debug stack, controlling breakpoints, and stepping through JavaScript code from within Eclipse.

You might still debug JavaScript code within your apps by using Firebug in Mozilla Firefox or by using other JavaScript debuggers.

## Enhanced tutorials and samples

In IBM Worklight V6.0.0, the list of tutorials and samples is augmented and enhanced for you to learn how to get started with the new features of IBM Worklight.

For more information about the new or highly modified tutorials and samples, and how you can use to get started with IBM Worklight, see Chapter 3, “Tutorials and samples,” on page 25.

## Specifying the remote server

Since IBM Worklight V6.0.0, the **WorklightServerRootURL** element is deprecated and no longer part of the `application-descriptor.xml` file. To specify the URL of the Worklight Server that your application uses, build your application by clicking **Run As > Build for Remote Server**. For more information about how to specify the remote server, see “Deploying an application from development to a test or production environment” on page 663.

---

## Chapter 3. Tutorials and samples

Tutorials and samples help you get started with and 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 33
- “JavaScript framework-based application samples” on page 33

You can find links to download compressed files that contain the materials for the tutorials and samples in “Additional resources” on page 33.

**Important:** These materials were 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 34, 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 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 2. Getting Started modules and samples

Module	Sample (if any)	Description
<b>1. Setting up your development environment</b>		
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.



Table 2. 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 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.
<b>2. Hello Worklight</b>		
<i>Highly revised:</i> 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.
<b>3. Worklight client-side development basics</b>		
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.
General information when developing for iOS		This module gives some general information that you must know to develop apps for the iOS environment.

Table 2. Getting Started modules and samples (continued)

Module	Sample (if any)	Description
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.
<i>Highly revised:</i> General information when developing Mobile Web applications		This module gives some general information that you must know to develop mobile web applications.
<i>Highly revised:</i> General information when developing desktop applications		This module gives some general information that you must know to develop desktop applications.
<b>4. Worklight server-side development</b>		
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. <b>Note:</b> This module has the same code sample as the module <i>HTTP adapter - Communicating with HTTP back-end systems</i> .
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	Exercise and code sample	This module explains how to work with adapters to communicate by using Java Message Service (JMS). <b>Note:</b> This module has the same code sample as the module <i>HTTP adapter - Communicating with HTTP back-end systems</i> .

Table 2. Getting Started modules and samples (continued)

Module	Sample (if any)	Description
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.
Using Java in adapters	Exercise and code sample	This module explains how to use Java in adapters. <b>Note:</b> This module has the same code sample as the module <i>HTTP adapter - Communicating with HTTP back-end systems</i> .
<b>5. Advanced client side development</b>		
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		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.
<i>Highly revised:</i> JSONStore - Common JSONStore Usage	Exercise and code sample	This module explains the common tasks that you can perform on a local JSON collection.
<i>Highly revised:</i> 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.
<b>6. Adding native functionality to hybrid applications with Apache Cordova</b>		
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.

Table 2. Getting Started modules and samples (continued)

Module	Sample (if any)	Description
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.
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. <b>Note:</b> This module has the same code sample as the module <i>iOS - Using native pages in hybrid applications</i> .
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.
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.
<b>7. Developing native applications with Worklight</b>		
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.
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.
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.
<b>8. Authentication and security</b>		

Table 2. Getting Started modules and samples (continued)

Module	Sample (if any)	Description
Authentication concepts		This module explains how to protect 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.
Adapter-based authentication	Exercise and code sample	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.
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.
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.
NEW: Application Authenticity Protection		This module explains how to work with the application authenticity protection.
<b>9. Advanced topics</b>		
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.
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.

Table 2. Getting Started modules and samples (continued)

Module	Sample (if any)	Description
NEW: Location services	Exercise and code sample	This module explains how to use geo-location services with IBM Worklight.
NEW: Testing Worklight Mobile Applications with the Mobile Test Workbench	iOS mobile client sample application	This module explains how to test your applications with IBM Worklight.
<b>10. Moving to production</b>		
<i>Highly revised:</i> 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.
Reports and analytics		This module explains the BIRT reports that you can use to collect the analytics data that pertains to applications and to devices that are accessing the Worklight Server.
NEW: Operational Analytics		This module explains how you can use the operational analytics features of IBM Worklight that replace much of the functionality of BIRT reports in earlier versions of IBM Worklight.
NEW: Distributing mobile applications with Application Center		This module explains how you can deploy and distribute your apps with IBM Worklight Application Center.
<b>11. Integrating with other products</b>		
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.
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.
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	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.
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 3. 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.

## 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 4. 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.
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 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 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.

---

## Chapter 4. Installing and configuring

This topic is intended for IT developers and administrators who want to install and configure IBM Worklight.

This topic describes the tasks required to install and configure the different components of IBM Worklight. It also contains information about installing and configuring database and application server software that you need 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)

---

### IBM Worklight installation overview

IBM Worklight provides the following installable parts: Worklight Studio, Worklight Server, and IBM Mobile Test Workbench for Worklight. This section gives an overview of the installation process.

Install IBM Installation Manager 1.6.3 or later separately before installing IBM Worklight.

To ensure the correct installation of Worklight Server, see “Installation prerequisites” on page 42.

#### Installing Worklight Studio on Eclipse with a P2 update site

You install Worklight Studio into an existing Eclipse. You also must install extra software development kits and Eclipse plug-ins for each mobile environment (Android Development Toolkit, for instance).

#### Installing Worklight Server with Installation Manager

Worklight server-side components are installed on your application server, and databases are created on your database system. Some application server and database configurations are required

---

### Installing Worklight Studio

You install Worklight Studio from your existing Eclipse IDE workbench.

#### Before you begin

- All Worklight Studio editions are installed with a P2 Eclipse update. Since IBM Worklight V6.0.0, IBM Installation Manager for Worklight Studio is no longer provided.
- Worklight Studio Developer Edition provides every Worklight Studio function available in Worklight Studio Consumer Edition and Worklight Studio Enterprise Edition, except for some security-related features.

To know more about the specificities of IBM Worklight Developer Edition, IBM Worklight Consumer Edition, and IBM Worklight Enterprise Edition, see “IBM Worklight editions” on page 8.

- Ensure that your computer meets the system requirements for the software that you install. For example, only Eclipse 4.2.2 (Juno) is supported (Java Enterprise Edition, or Classic). For more information, see “System requirements for using IBM Worklight” on page 8.

**Note:** Errors happen immediately if you use another version of Eclipse.

- Ensure that an Internet connection is available in case dependencies that are required by the installation are not already included in the Eclipse IDE.

### About this task

- 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, complete the following procedure.

### Procedure

1. Start your Eclipse IDE workbench.
2. Click **Help** > **Install new software**.
3. Beside the **Work with** field, click **Add**.
4. In the Add Repository window, click **Archive**.
5. Browse to the location of the P2 update .zip file on the DVD, or of the archive file on your machine, and click **Open**.
6. Click **OK** to exit the Add Repository window.
7. Select the features of Worklight Studio that you want to install, and then click **Next**.
8. On the Install Details page, review the features that you install, and then click **Next**.
9. 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.
10. Follow the prompts to complete the installation.

### What to do next

Before you run Worklight Studio, determine whether you must run extra post-installation tasks.

## Running post-installation tasks

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

### 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 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.

## Starting Worklight Studio

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.

#### 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 Chapter 3, "Tutorials and samples," on page 25.

## 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 Chapter 3, "Tutorials and samples," on page 25.

**Note:** On Ubuntu (Linux), you must check that the Android SDK works properly. You might need to add some `.lib` files. For more information, see <http://developer.android.com/sdk/installing/index.html>.

## 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.

**Note:** BlackBerry OS 10 is not supported by the current version of the Application Center.

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 Chapter 3, "Tutorials and samples," on page 25.

### Installing tools for Windows Phone 7.5

To build and sign applications for Windows Phone 7.5 (deprecated in IBM Worklight V6.0.0), 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.

### 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 Chapter 3, "Tutorials and samples," on page 25.

### 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/br211384.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.

## 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 10080 by default.

If port 10080 is occupied by another application that is running on your development computer, you can configure the Worklight Studio internal server to use a different port. To do so, follow these instructions:

### Procedure

1. Open the Servers view in Eclipse.
2. Expand the **Worklight Development Server** list.
3. Double-click **Server Configuration [server.xml] worklight**.
4. In the Server Configuration window, click **HTTP endpoint**.
5. Change the **Port** value to any port number of your choice.

---

## Installing IBM Mobile Test Workbench for Worklight

You must install IBM Mobile Test Workbench for Worklight into an Eclipse IDE where Worklight Studio V6.0.0 or later is already installed.

### Before you begin

- Your computer must meet the system requirements for the software that you install. For more information, see “System requirements for using IBM Worklight” on page 8.
- Worklight Studio V6.0.0 or later must be already installed on your computer.
- Ensure that an Internet connection is available in case dependencies that are required by the installation are not already included in the Eclipse IDE.

### About this task

- To install IBM Mobile Test Workbench for Worklight on top of IBM Worklight Developer Edition, go to the IBM Worklight development website at <https://www.ibm.com/developerworks/mobile/worklight/>.
- To install IBM Mobile Test Workbench for Worklight on top of IBM Worklight Consumer Edition or IBM Worklight Enterprise Edition, complete the following procedure.

### Procedure

1. Start your Eclipse IDE workbench.
2. Click **Help > Install new software**.
3. Beside the **Work with** field, click **Add**.
4. In the Add Repository window, click **Archive**.
5. Browse to the location of the P2 update .zip file on the DVD, or of the archive file on your machine, and click **Open**.
6. Click **OK** to exit the Add Repository window.
7. Select the features of IBM Mobile Test Workbench for Worklight that you want to install, and then click **Next**.
8. On the Install Details page, review the features that you install, and then click **Next**.

9. 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.
10. Follow the prompts to complete the installation.

## What to do next

When IBM Mobile Test Workbench for Worklight is installed, you can install the mobile test client. For more information about the mobile test client installation, see [Installing the mobile test client](#).

To understand the various available features, see [Testing with IBM Worklight](#).

Make sure you add a JDK to your path in Eclipse. For more information, see ["Troubleshooting IBM Mobile Test Workbench for Worklight."](#)

## Troubleshooting IBM Mobile Test Workbench for Worklight

If you do not set Eclipse to use a JDK installed on your system, you cannot test Android applications with IBM Mobile Test Workbench for Worklight. The testing process fails and you get error messages.

### About this task

To test Android applications with IBM Mobile Test Workbench for Worklight, you must add the path to the JDK in Eclipse.

### Procedure

1. In Eclipse, go to **Window > Preferences > Java > Installed JREs**.
2. Select the JDK check box to set the JDK as the default JRE.

---

## Installing Worklight Server

IBM installations are based on an IBM product called IBM Installation Manager. Install IBM Installation Manager 1.6.3.1 or later separately before you install IBM Worklight.

**Important:** Ensure that you use IBM Installation Manager 1.6.3.1 or later. This version contains an important fix for an issue identified in IBM Installation Manager 1.6.3. See <http://www.ibm.com/support/docview.wss?uid=swg24035049>.

The Worklight Server installer copies onto your computer all the tools and libraries that are required for deploying an IBM Worklight Project or the Application Center in production, and the IBM WebSphere Analytics Platform.

Worklight Server can also automatically deploy the Application Center at installation time. In this case, a database management system and an application server are required as prerequisites and must be installed before you start the Worklight Server installer.

The installer can also help with upgrading an existing installation of Worklight Server to the current version. See Chapter 5, "Upgrading from one version of IBM Worklight to another," on page 181.

The following topics describe the installation of Worklight Server, installation prerequisites, and the procedures for a manual installation and configuration of

Application Center. After Worklight Server is installed, a Worklight project must be deployed to an application server. This deployment installs a Worklight Console that can be used to upload applications and adapters. The instructions in “Overview of the Worklight Server installation process” on page 43 are based on a simple installation scenario. For a complete description of the process of deploying a Worklight project, see Chapter 8, “Deploying IBM Worklight projects,” on page 663.

## Installation prerequisites

For smooth installation of Worklight Server, ensure that you fulfill all required environment setup and software prerequisites before you attempt installation.

You can find a complete list of supported hardware together with prerequisite software under: IBM Worklight and IBM Mobile Foundation detailed system requirements.

**Important:** If a version of Worklight Server is already installed, review Chapter 5, “Upgrading from one version of IBM Worklight to another,” on page 181 before you install Worklight Server and deploy a Worklight project on the same application server or databases. Failure to do so can result in an incomplete installation and a non-functional Worklight Server.

Download the IBM Mobile Platform foundation package or the IBM Worklight package from the IBM Passport Advantage® site.

Ensure that you have the latest fix packs for the IBM Worklight product. If you are connected to the Internet during the installation, IBM Installation manager can download the latest fix packs for you.

The package contains an Install Wizard that guides you through the Worklight Server installation.

Worklight Server requires an application server and relies on a database management system.

You can use any of the following application servers:

- WebSphere Application Server Liberty Core.
- WebSphere Application Server.
- Apache Tomcat.

You can use any of the following database management systems:

- IBM DB2®
- MySQL
- Oracle
- Apache Derby in embedded mode. Included in the installation image.

**Note:** Apache Derby is supplied for evaluation and testing purposes only and is not supported for production-grade IBM Worklight servers.

The IBM Worklight installer can install the Application Center and deploy it to your application server. In this case, the application server and the database management system (if different from Apache Derby) must be installed before you start the Worklight Server installer. If you do not need the Application Center or decide to install it manually, you do not need to install the application server and

database management system before starting the Worklight Server installer. However, you need them before deploying IBM Worklight projects.

The IBM Mobile Platform Foundation and IBM Worklight packages include the following installers:

- IBM DB2 Workgroup Server Edition
- IBM DB2 Enterprise Server Edition (on zLinux only)
- IBM WebSphere Application Server Liberty Core

## Overview of the Worklight Server installation process

Learn about the Worklight Server installation process based on a simple configuration that creates a functional Worklight Server that is usable for demonstration purposes or tests.

### Before you begin

- Install Worklight Studio on your computer. You use Worklight Studio to create a simple IBM Worklight project.
- Install Apache Ant on your computer. You use Apache Ant to deploy the IBM Worklight project. You can download Apache Ant from <http://ant.apache.org/bindownload.cgi>.

### About this task

This task shows you how to install the Worklight Server on a simple configuration and shows you where to find the following tools and information:

- Tools to install a Worklight Server and the Application Center, and tools to deploy an IBM Worklight project.
- Information about configuring the Worklight Server and the Application Center.
- Information about manual Worklight Server installation. Manual installation provides greater flexibility, but can make the diagnosis of issues more complex, and make the description of a configuration to IBM Support more difficult.

This task involves installing the following components:

- An IBM WebSphere Application Server Liberty Core application server.
- A Derby database (installed by the Worklight Server installer).
- The Application Center.
- A very simple IBM Worklight project and its console.

### Procedure

1. Install IBM WebSphere Application Server Liberty Core. The installer for IBM WebSphere Application Server Liberty Core is provided as part of the package for IBM Worklight or IBM Mobile Foundation.
  - a. Load the repository for IBM WebSphere Application Server Liberty Core in IBM Installation Manager and install the product.
  - b. During the installation process, take note of the installation directory of Liberty. You need this information later on in the procedure.
2. Create a server for Liberty. You use this server to install the Application Center and to deploy an IBM Worklight project and its console.
  - a. Go to the installation directory of Liberty. For example, on Windows, if the product is installed with Administrator rights, it is located by default in `C:\Program Files\IBM\WebSphere\Liberty`.

- b. Type the command that creates a server. In this scenario, the chosen name is `simpleServer`.

**On UNIX and Linux systems:**

```
bin/server create simpleServer
```

**On Windows systems:**

```
bin\server.bat create simpleServer
```

The server is created with all default settings. For more information about configuring a Liberty Server, read the file `README.txt` in the Liberty installation directory. Default settings are sufficient for this walkthrough.

3. Install Worklight Server.
  - a. Load the repository for Worklight Server in IBM Installation Manager and install the product.
    - 1) In the **Configuration Choice** panel, select the first choice. This option installs Application Center.
    - 2) In the **Database Choice** panel, select **Install Apache Derby**. This database is not recommended for production use, but is sufficient for this walkthrough.
    - 3) In the Application Server Selection panel, select **WebSphere Application Server**.
    - 4) In the **Application Server Configuration** panel, select the installation directory for IBM WebSphere Application Server Liberty Core that is installed in step 2.
    - 5) Select **simpleServer** as the server name.
    - 6) Install the product.

The files that are described in “Distribution structure of Worklight Server” on page 62 are installed on your computer.

4. Explore Application Center. Application Center is now functional. The artifacts of the Application Center are deployed into the Liberty Server, which now includes the features that Application Center requires, and a demonstration user account exists. The required Derby database also exists.
  - a. To test the Application Center, start the Liberty server.

**On UNIX and Linux systems:**

```
bin/server start simpleServer
```

**On Windows systems:**

```
bin\server.bat start simpleServer
```

- b. Open the Application Center by using the program shortcut that the installer creates: **IBM Worklight Server > Application Center**. Alternatively, you can enter the URL for the Application Center into a browser window. When a Liberty server is created with default settings, the default URL for Application Center is `http://localhost:9080/appcenterconsole/`.
- c. Log in to the Application Center with the demonstration account credentials (login: `demo`, password: `demo`)
- d. Explore further by using any of the following resources:
  - See “Configuring the Application Center after installation” on page 118.
  - See “Distribution structure of Worklight Server” on page 62 for a list of IBM Worklight applications that you can compile and upload to the application center. These applications provide access to the Application Center for mobile devices.



- If you are considering a manual installation of Application Center, see “Manually installing Application Center” on page 66. In some cases, manual installations can make the diagnosis of issues more complex, and can make the description of a configuration to IBM Support more difficult.
5. Create a simple IBM Worklight project. You create a project to deploy a console.
    - a. Complete the following steps:
      - 1) Install Worklight Studio on your computer. See “Installing Worklight Studio” on page 35.
      - 2) Start Worklight Studio.
      - 3) Create an IBM Worklight project (**File > New > Worklight Project**).
      - 4) Assign the name `simpleProject`, and accept the default project template **Hybrid Application**.
      - 5) In the next panel, name the application `simpleApp`, and then click **Finish**.
    - b. Build the application.
      - 1) In the Project Explorer view in Worklight Studio, open the project.
      - 2) Open the **apps** folder, right-click the subfolder **simpleApp**, and then click **Run As/Build and Deploy**.
      - 3) In the Project Explorer view, open the **bin** folder that was created by this task. Right-click **simpleProject.war** and click **Properties**. The properties show the path to the WAR file. This path is used in the next step. For example, if the path of the Eclipse workspace is `C:\workspaces\WorklightStudioWorkspace`, the path to the WAR file is `C:\workspaces\WorklightStudioWorkspace\simpleProject\bin\simpleProject.war`.
  6. Create an Ant file to deploy an IBM Worklight project. For this step, Apache Ant must be installed on your computer. IBM Worklight uses the Apache Ant Java library and command-line tool to automate build and deployment tasks.
    - a. Copy the `<WorklightInstallationDirectory>/WorklightServer/configuration-samples/configure-liberty-derby.xml` file to a working directory so that you can edit it and create a modified version to deploy an IBM Worklight project to the Liberty application server. This file is an example of the use of Ant tasks to deploy an IBM Worklight project. See “Deploying an IBM Worklight project” on page 665.
    - b. Open your local copy of the file and modify the seven properties that describe the deployment directories:

**Note:** The following values correspond to a default installation on Microsoft Windows with administrator rights. Check the actual values for your configuration.

- For property `worklight.project.war.file`, the absolute path of the project that was created in step 5.  
`<property name="worklight.project.war.file" value="C:\workspaces\WorklightStudioWorkspace\simpleProject\bin\simpleProject.war"/>`
- For property `worklight.server.install.dir`, the installation directory for the Worklight Server that was installed in step 3.  
`<property name="worklight.server.install.dir" value="C:\Program Files\IBM\Worklight"/>`
- For property `derby.databases.data.dir`, choose the directory where you want to create the databases for Application Center.  
`<property name="derby.databases.data.dir" value="c:\simpleServerTest\derby"/>`
- For property `was.liberty.install.dir`, the installation directory of Liberty, in step 1.

- ```
<property name="was.liberty.install.dir" value="C:\Program Files\IBM\WebSphere\Liberty"/>
```
- For property `was.liberty.server.name`, the name of the server that was created in step 2.
 

```
<property name="was.liberty.server.name" value="simpleServer"/>
```
  - For property `shortcuts.dir`, the name of a directory to create a UNIX shortcut and a Windows shortcut to the console.
 

```
<property name="shortcuts.dir" value="c:\simpleServerTest\shortcuts"/>
```
  - For property `reports.dir`, the name of a directory to create BIRT reports.
 

```
<property name="reports.dir" value="c:\simpleServerTest\reports"/>
```
7. Run the Ant script to create the databases for the deployed IBM Worklight project.
    - a. In a console, type the following command:
 

```
ant -f configure-liberty-derby.xml databases
```

**Note:** If you do not have the Ant command in your PATH variable, type the full path to the Ant command.

8. Run the Ant script to deploy the IBM Worklight project. In a console, type the following command:
 

```
ant -f configure-liberty-derby.xml install
```

The output of the Ant task displays the details of the steps that are completed during deployment. At the end of the traces, it displays the Worklight Console. Take a note of the URL: it is used in step 9.

```
[...]
[configureapplicationserver] [replace] Replacing in c:\simpleServerTest\shortcuts\worklight-con
[configureapplicationserver]
[configureapplicationserver] cleanup:
[configureapplicationserver]
[configureapplicationserver] install:
[configureapplicationserver]
[configureapplicationserver] BUILD SUCCESSFUL
[configureapplicationserver] Total time: 24 seconds
```

9. Restart the Liberty server and open the Worklight Console.
  - a. Go to the Liberty Installation Directory. Type the following command:
 

```
bin\server.bat stop simpleServer
```
  - b. Restart the server with the following command:
 

```
bin\server.bat start simpleServer
```
  - c. Wait a few seconds while the Liberty server completes initializing and loads all its applications.
  - d. Open a web browser and open the URL of the Worklight Console that you noted in step 8.

## What to do next

If you want to explore the Worklight Console further, you can complete the following tasks:

- Deploy an application as described in “Deploying applications and adapters to Worklight Server” on page 733.
- Review Chapter 9, “Administering IBM Worklight applications,” on page 755.
- Review “Deploying an IBM Worklight project” on page 665.

- Review “Configuration of IBM Worklight applications on the server” on page 714 and “Configuring an IBM Worklight project in production by using JNDI environment entries” on page 723
- Review the options to deploy an IBM Worklight project manually. In some cases, manual installations can make the diagnosis of issues more complex, and can make the description of a configuration to IBM Support more difficult. See “Deploying a project WAR file and configuring the application server manually” on page 710.

## Running IBM Installation Manager

IBM Installation Manager installs the IBM Worklight files and tools on your computer.

IBM Installation Manager helps you install, update, modify, and uninstall packages on your computer. The installer for Worklight Server does not support rollback operations and updates from versions 5.x to 6.0 cannot be undone.

The way you use IBM Installation Manager to upgrade from a previous release depends on your upgrade path.

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/v1r6/index.jsp>.

### Optional creation of databases

If you want to activate the option to install the Application Center when you run the IBM Worklight Server installer, you need to have certain database access rights that entitle you to create the tables that are required by the Application Center.

If you have sufficient database administration credentials, and if you enter the administrator user name and password in the installer when prompted, the installer can create the databases for you. Otherwise, you need to ask your database administrator to create the required database for you. The database needs to be created before you start the IBM Worklight Server installer.

The following topics describe the procedure for the supported database management systems.

#### Creating the DB2 database for Application Center:

During IBM Worklight installation, the installer can create the Application Center database for you.

#### About this task

The installer can create the Application Center database 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 Application Center database for you. For more information, see the DB2 Solution user documentation.

When you manually create the database, you can replace the database name (here APPCNTR) and the password with a database name and password of your choosing.

**Important:** You can name your database and user differently, or set a different password, but ensure that you enter the appropriate database name, user name, and password correctly across the DB2 database setup. DB2 has a database name limit of 8 characters on all platforms, and 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, for example, named `wluser` in a DB2 admin group such as `DB2USERS`, using the appropriate commands for your operating system. Give it a password, for example, `wluser`. If you want multiple IBM Worklight Servers to connect to the same database, use a different user name for each connection. Each database user has a separate default schema. For more information about database users, 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 `~/sql1lib/bin` and enter `./db2`.
  - Enter database manager and SQL statements similar to the following example to create the Application Center database, replacing the user name `wluser` with your chosen user names:

```
CREATE DATABASE APPCNTR COLLATE USING SYSTEM PAGESIZE 32768
CONNECT TO APPCNTR
GRANT CONNECT ON DATABASE TO USER wluser
DISCONNECT APPCNTR
QUIT
```
3. The installer can create the database tables and objects for Application Center in a specific schema. This allows you to use the same database for Application Center and for a Worklight project. If the `IMPLICIT_SCHEMA` authority is granted to the user created in step 1 (the default in the database creation script in step 2), no further action is required. If the user does not have the `IMPLICIT_SCHEMA` authority, you need to create a `SCHEMA` for the Application Center database tables and objects.

### Creating the MySQL database for Application Center:

During the IBM® Worklight installation, the installer can create the Application Center database for you.

### About this task

The installer can create the database 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, you can replace the database name (here `APPCNTR`) and password with a database name and password of your choosing. Note that MySQL database names are case-sensitive on Unix.

## Procedure

1. Start the MySQL command-line tool.
2. Enter the following commands:

```
CREATE DATABASE APPCNTR CHARACTER SET utf8 COLLATE utf8_general_ci;  
GRANT ALL privileges ON APPCNTR.* TO 'worklight'@'Worklight-host' IDENTIFIED BY 'password';  
GRANT ALL privileges ON APPCNTR.* TO 'worklight'@'localhost' IDENTIFIED BY 'password';  
Flush privileges;
```

Here, you need to replace *Worklight-host* with the name of the host on which IBM Worklight runs.

## Creating the Oracle database for Application Center:

During the IBM® Worklight installation, the installer can create the Application Center database or the user and schema inside an existing database for you.

### About this task

The installer can create the database or user and schema inside an existing database 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 database or user and schema for you. When you manually create the database or user, you can use database names, user names, and a password of your choosing. Note that lowercase characters in Oracle user names can lead to trouble.

## Procedure

1. If you do not already have a database named ORCL, use the Oracle Database Configuration Assistant (DBCA) and follow the steps in the wizard to create a new general-purpose database named ORCL:
  - a. Use global database name *ORCL\_your\_domain*, and system identifier (SID) ORCL.
  - 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 procedure, accepting the default values.
2. Create a database user either by using Oracle Database Control, or by using the Oracle SQLPlus command-line interpreter.
  - Using Oracle Database Control.
    - a. Connect as SYSDBA.
    - b. Go to the **Users** page: click **Server**, then **Users** in the **Security** section.
    - c. Create a user, for example, named APPCENTER. If you want multiple IBM Worklight Servers to connect to the same general-purpose database you created in step 1, use a different user name for each connection. Each database user has a separate default schema.
    - d. Assign 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**
- Using the Oracle SQLPlus command-line interpreter.  
The commands in the following example create a user named APPCENTER for the database:  
CONNECT system/<system\_password>@ORCL  
CREATE USER APPCENTER IDENTIFIED BY password;  
GRANT CONNECT, RESOURCE, CREATE VIEW TO APPCENTER;  
DISCONNECT;

## Single-user versus multi-user installations

You can install Worklight Server 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.

When you have chosen an administrator installation of Worklight Server, it is advisable to run the application server from a non-administrator user account. Running it from an administrator or root user account is dangerous in terms of security risks.

Because of this, during an administrator installation of Worklight Server, you have the option to choose an operating system user or an operating system user group. Each of the users in this group can:

- Run the specified application server (if WebSphere Application Server Liberty Profile, or Apache Tomcat).
- Modify the Application Center Derby database (if Apache Derby is chosen as your database management system).

In this case, the Worklight Server installer will set restrictive access permissions on the Liberty or Tomcat configuration files, so as to:

1. Allow the specified users to run the application server.
2. At the same time, protect the database or user passwords that these files contain.

### 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.

The following constraints regarding user accounts on UNIX apply:

- If the application server is owned by a non-root user, you can install Worklight Server 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 application server is owned by root, you can install Worklight Server 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.

### Installing a new version of Worklight Server

Create a fresh installation of Worklight Server by creating a new package group in IBM Installation Manager.

#### Procedure

1. Start IBM Installation Manager.
2. On the IBM Installation Manager main page, click the **Install** button.
3. In the panel that prompts for the package group name and the installation directory, select **Create a new package group**.
4. Complete the installation by following the instructions that are displayed.

### Upgrading Worklight Server from a previous release

The way that you use IBM Installation Manager to upgrade from a previous version of Worklight Server depends on your upgrade path.

#### Before you begin

Before you apply these instructions, see Chapter 5, “Upgrading from one version of IBM Worklight to another,” on page 181. It describes important steps to upgrade IBM Worklight applications, or to upgrade a production server in a production environment.

#### Procedure

1. Start the IBM Installation Manager.
2. Depending on your upgrade path, take one of the following actions:
  - To upgrade from V5.x to V6.0 or later:
    - a. Click the **Install** button.
    - b. In the panel that prompts for the package group name and the installation directory, select **Use the existing package group**. In this situation, installation of V6.0 or later automatically removes a V5.x installation that was installed in the same directory.
  - To upgrade from V6.x to a newer version, click the **Update** button.

### Installing Worklight Server into WebSphere Application Server Network Deployment

To install Worklight Server into a set of WebSphere Application Server Network Deployment servers, run IBM Installation Manager on the machine where the deployment manager is running.

#### Procedure

1. When IBM Installation Manager prompts you to specify the database type, select any option other than **Apache Derby**. IBM Worklight supports Apache Derby only in embedded mode, and this choice is incompatible with deployment through WebSphere Application Server Network Deployment.
2. In the installer panel in which you specify the WebSphere Application Server installation directory, select the deployment manager profile.

**Attention:** Do not select an application server profile and then a single managed server: doing so causes the deployment manager to overwrite the configuration of the server regardless of whether you install on the machine on which the deployment manager is running or on a different machine.

3. Select the required scope depending on where you want Worklight Server to be installed. The following table lists the available scopes:

Table 5. Selecting the required scope

Scope	Explanation
Cell	Installs Worklight Server into all application servers of the cell.
Cluster	Installs Worklight Server into all application servers of the specified cluster.
Node (excluding clusters)	Installs Worklight Server into all application servers of the specified node that are not in a cluster.
Server	Installs Worklight Server into the specified server, which is not in a cluster.

4. Restart the target servers by following the procedure in “Completing the installation” on page 62.

## Results

The installation has no effect outside the set of servers in the specified scope. The JDBC providers and JDBC data sources are defined with the specified scope. The entities that have a cell-wide scope (the applications and, for DB2, the authentication alias) have a suffix in their name that makes them unique. So, you can install Worklight Server in different configurations or even different versions of Worklight Server, in different clusters of the same cell.

**Note:** Because the JDBC driver is installed only in the specified set of application servers, the **Test connection** button for the JDBC data sources in the WebSphere Application Server administration console of the deployment manager might not work.

## What to do next

You need to complete the following additional configuration:

- If you use a front-end HTTP server, you need to configure the public URL

## Silent installation

You can use IBM Installation Manager to perform silent installation of Worklight Server on multiple machines or on machines where a GUI interface is not available.

## 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 Worklight Server on a set of machines that are configured in the same way.

- You want to install Worklight Server 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.

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.

For a list of the parameters created in the response file by the IBM Installation Manager wizard, see “Silent installation parameters” on page 55.

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='w15test'/>
    <data key='user.database.db2.password' value='{xyzy}72840FD1KRHW8AC13S'/>
    <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
  </install>
  <preference name='com.ibm.cic.common.core.preferences.eclipseCache' value='/n/java/rational/
  <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' value='true' />
<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' />

    <!-- 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='w15test' />
    <data key='user.database.db2.password' value='{xyzz}72840FD1KRHW8AC13S' />
    <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 Worklight Server by using the response file on the target machine, as described in Install a package silently by using a response file.

**Silent installation parameters:**

The response file that you create for silent installations by running the IBM Installation Manager wizard supports a number of parameters.

*Table 6. Parameters available for silent installation*

Key	When necessary	Description	Allowed values
<b>user.appserver.selection2</b>	Always	Type of application server. was means preinstalled WAS 7.0, 8.0, or 8.5. tomcat means Tomcat 7.0 or newer.	was, tomcat, none  The value none means that the installer will not install the Application Center. If this value is used, both <b>user.appserver.selection2</b> and <b>user.database.selection2</b> must take the value none.
<b>user.appserver.was.installdir</b>	\${user.appserver.selection2} == was	WAS installation directory.	An absolute directory name.

Table 6. Parameters available for silent installation (continued)

Key	When necessary	Description	Allowed values
<b>user.appserver.was.profile</b>	\${user.appserver.selection2} == was	Profile into which to install the applications. For WAS ND, specify the Deployment Manager profile. Liberty means the Liberty profile (subdirectory wlp).	The name of one of the WAS profiles.
<b>user.appserver.was.cell</b>	\${user.appserver.selection2} == was && \${user.appserver.was.profile} != Liberty	WAS cell into which to install the applications.	The name of the WAS cell.
<b>user.appserver.was.node</b>	\${user.appserver.selection2} == was && \${user.appserver.was.profile} != Liberty	WAS node into which to install the applications. This corresponds to the current machine.	The name of the WAS node of the current machine.
<b>user.appserver.was.scope</b>	\${user.appserver.selection2} == was && \${user.appserver.was.profile} != Liberty	Type of set of servers into which to install the applications. server means a standalone server. nd-cell means a WAS ND cell. nd-cluster means a WAS ND cluster. nd-node means a WAS ND node (excluding clusters). nd-server means a managed WAS ND server.	server, nd-cell, nd-cluster, nd-node, nd-server



Table 6. Parameters available for silent installation (continued)

Key	When necessary	Description	Allowed values
<b>user.appserver.was.serverInstance</b>	<pre>{user.appserver.selection2} == was &amp;&amp; \${user.appserver.was.profile} != Liberty &amp;&amp; \${user.appserver.was.scope} == server</pre>	Name of WAS server into which to install the applications.	The name of a WAS server on the current machine.
<b>user.appserver.was.nd.cluster</b>	<pre>{user.appserver.selection2} == was &amp;&amp; \${user.appserver.was.profile} != Liberty &amp;&amp; \${user.appserver.was.scope} == nd-cluster</pre>	Name of WAS ND cluster into which to install the applications.	The name of a WAS ND cluster in the WAS cell.
<b>user.appserver.was.nd.node</b>	<pre>{user.appserver.selection2} == was &amp;&amp; \${user.appserver.was.profile} != Liberty &amp;&amp; (\${user.appserver.was.scope} == nd-node     \${user.appserver.was.scope} == nd-server)</pre>	Name of WAS ND node into which to install the applications.	The name of a WAS ND node in the WAS cell.
<b>user.appserver.was.nd.server</b>	<pre>{user.appserver.selection2} == was &amp;&amp; \${user.appserver.was.profile} != Liberty &amp;&amp; \${user.appserver.was.scope} == nd-server</pre>	Name of WAS ND server into which to install the applications.	The name of a WAS ND server in the given WAS ND node.
<b>user.appserver.was.admin.name</b>	<pre>{user.appserver.selection2} == was &amp;&amp; \${user.appserver.was.profile} != Liberty</pre>	Name of WAS administrator.	
<b>user.appserver.was.admin.password</b>	<pre>{user.appserver.selection2} == was &amp;&amp; \${user.appserver.was.profile} != Liberty</pre>	Password of WAS administrator, optionally encrypted in a specific way.	
<b>user.appserver.was.appcenteradmin.password</b>	<pre>{user.appserver.selection2} == was &amp;&amp; \${user.appserver.was.profile} != Liberty</pre>	Password of appcenteradmin user to add to the WAS users list, optionally encrypted in a specific way.	
<b>user.appserver.was.serial</b>	<pre>{user.appserver.selection2} == was &amp;&amp; \${user.appserver.was.profile} != Liberty</pre>	Suffix that distinguishes the applications to be installed from other installations of Worklight Server.	String of 10 decimal digits.

Table 6. Parameters available for silent installation (continued)

Key	When necessary	Description	Allowed values
<b>user.appserver.wasliberty.server.instance</b>	$\$(user.appserver.selection2)$ == was && $\$(user.appserver.was.profile)$ == Liberty	Name of WAS Liberty Server into which to install the applications.	
<b>user.appserver.tomcat.installdir</b>	$\$(user.appserver.selection2)$ == tomcat	Apache Tomcat installation directory.	An absolute directory name.
<b>user.database.selection2</b>	Always	Type of database management system used to store the databases.	derby, db2, mysql, oracle, none  The value none means that the installer will not install the Application Center. If this value is used, both <b>user.appserver.selection2</b> and <b>user.database.selection2</b> must take the value none.
<b>user.database.preinstalled</b>	Always	true means a preinstalled database management system, false means Apache Derby to install.	true, false
<b>user.database.derby.datadir</b>	$\$(user.database.selection2)$ == derby	The directory in which to create or assume the Derby databases.	An absolute directory name.
<b>user.database.db2.host</b>	$\$(user.database.selection2)$ == db2	The host name or IP address of the DB2 database server.	

Table 6. Parameters available for silent installation (continued)

Key	When necessary	Description	Allowed values
<b>user.database.db2.port</b>	\${user.database.selection2} == db2	The port where the DB2 database server listens for JDBC connections. Usually 50000.	A number between 1 and 65535.
<b>user.database.db2.driver</b>	\${user.database.selection2} == db2	The absolute file name of db2jcc.jar or db2jcc4.jar.	An absolute file name.
<b>user.database.db2.appcenter.userName</b>	\${user.database.selection2} == db2	The user name used to access the DB2 database for Application Center.	Non-empty.
<b>user.database.db2.appcenter.password</b>	\${user.database.selection2} == db2	The password used to access the DB2 database for Application Center, optionally encrypted in a specific way.	Non-empty password.
<b>user.database.db2.appcenter.dbName</b>	\${user.database.selection2} == db2	The name of the DB2 database for Application Center.	Non-empty; a valid DB2 database name.
<b>user.database.db2.appcenter.schema</b>	\${user.database.selection2} == db2	The name of the schema for Application Center in the DB2 database.	
<b>user.database.mysql.host</b>	\${user.database.selection2} == mysql	The host name or IP address of the MySQL database server.	
<b>user.database.mysql.port</b>	\${user.database.selection2} == mysql	The port where the MySQL database server listens for JDBC connections. Usually 3306.	A number between 1 and 65535.

Table 6. Parameters available for silent installation (continued)

Key	When necessary	Description	Allowed values
<b>user.database.mysql.driver</b>	\${user.database.selection2} == mysql	The absolute file name of mysql-connector-java-5.*-bin.jar.	An absolute file name.
<b>user.database.mysql.appcenter.username</b>	\${user.database.selection2} == mysql	The user name used to access the MySQL database for Application Center.	Non-empty.
<b>user.database.mysql.appcenter.password</b>	\${password} == mysql	The password used to access the MySQL database for Application Center, optionally encrypted in a specific way.	
<b>user.database.mysql.appcenter.dbname</b>	\${name} == mysql	The name of the MySQL database for Application Center.	Non-empty, a valid MySQL database name.
<b>user.database.oracle.host</b>	\${user.database.selection2} == oracle, unless \${user.database.oracle.appcenter.jdbc.url} is specified	The host name or IP address of the Oracle database server.	
<b>user.database.oracle.port</b>	\${user.database.selection2} == oracle, unless \${user.database.oracle.appcenter.jdbc.url} is specified	The port where the Oracle database server listens for JDBC connections. Usually 1521.	A number between 1 and 65535.
<b>user.database.oracle.driver</b>	\${user.database.selection2} == oracle	The absolute file name of ojdbc6.jar.	An absolute file name.

Table 6. Parameters available for silent installation (continued)

Key	When necessary	Description	Allowed values
<code>user.database.oracle.appcenter</code>	<code>\$(user.database.selection2)</code> == oracle	The user name used to access the Oracle database for Application Center.	A string consisting of 1 to 30 characters: ASCII digits, ASCII uppercase and lowercase letters, '_', '#', '\$' are allowed.
<code>user.database.oracle.appcenter</code>	<code>\$(user.database.selection2)</code> == oracle	The user name used to access the Oracle database for Application Center, in a syntax suitable for JDBC.	Same as <code>\$(user.database.oracle.appcenter)</code> if it starts with an alphabetic character and does not contain lowercase characters, otherwise it must be <code>\$(user.database.oracle.appcenter)</code> surrounded by double quotes.
<code>user.database.oracle.appcenter</code>	<code>\$(password)</code> == oracle	The password used to access the Oracle database for Application Center, optionally encrypted in a specific way.	The password must be a string consisting of 1 to 30 characters: ASCII digits, ASCII uppercase and lowercase letters, '_', '#', '\$' are allowed.
<code>user.database.oracle.appcenter</code>	<code>\$(name)</code> == oracle, unless <code>\$(user.database.oracle.appcenter)</code> is specified	The name of the Oracle Application Center.	Non-empty, a valid Oracle database name.
<code>user.database.oracle.appcenter</code>	<code>\$(url)</code> == oracle, unless <code>\$(user.database.oracle.host)</code> , <code>\$(user.database.oracle.port)</code> , <code>\$(user.database.oracle.appcenter)</code> are all specified	The JDBC URL of the Oracle database for Application Center.	A valid Oracle JDBC URL. Starts with "jdbc:oracle:".

Table 6. Parameters available for silent installation (continued)

Key	When necessary	Description	Allowed values
<code>user.writable.data.user</code>	Always	The operating system user that is allowed to run the installed server.	An operating system user name, or empty.
<code>user.writable.data.group2</code>	Always	The operating system users group that is allowed to run the installed server.	An operating system users group name, or empty.

## 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 and have it launched without the **Application Security** active before you installed Application Center or Worklight Server.

The IBM Worklight installer must activate the **Application Security** of WebSphere Application Server (if not active yet) to install Application Center. Then, for this activation to take place, you must restart the application server after the installation of Worklight Server completed.

- If you are using WebSphere Application Server Liberty Profile or Apache Tomcat.
- After you upgraded from a previous version of 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 have been installed.

To restart these servers with the deployment manager console, select **Applications > Application Types > WebSphere enterprise applications > IBM\_Application\_Center\_Services > Target specific application status**.

- You do not have to restart the deployment manager or the node agents.

**Note:** The installation has installed only the IBM Application Center into the application server. A Worklight Console is not installed by default. To install a Worklight Console, you need to follow the steps in Chapter 8, “Deploying IBM Worklight projects,” on page 663.

## Distribution structure of Worklight Server

The Worklight Server files and tools are installed in and below the Worklight Server installation directory.



Table 7. Files in the Worklight Server installation directory

Item	Description
analytics.zip	Archive for the IBM WebSphere Analytics Platform. See “Installing and configuring the IBM WebSphere Analytics Platform” on page 174 for instructions to install and configure this platform.

Table 8. Files and subdirectories in the WorklightServer subdirectory

Item	Description
worklight-jee-library.jar	The Worklight server library for production. See Chapter 8, “Deploying IBM Worklight projects,” on page 663 for instructions on deploying an IBM Worklight Project and this library to an Application Server. The deployment is typically performed by using Ant tasks, but instructions for manual deployment are also provided.
worklight-ant.jar	A set of Ant tasks that help you build and deploy projects, applications, and adapters to your Worklight Server. See Chapter 8, “Deploying IBM Worklight projects,” on page 663 for detailed documentation of the Ant tasks that are provided in this library.
configuration-samples	Sample Ant files for configuring a database for the Worklight Server and deploying an IBM Worklight Project to an Application Server. See “Sample configuration files” on page 706 for instructions on how to use these Ant projects.
databases	SQL scripts to be used for the manual creation of tables for Worklight Server, instead of using Ant tasks for the automatic configuration of the tables for Worklight Server. These scripts are described in “Creating and configuring the databases manually” on page 678.
encrypt.bat and encrypt.sh	Tools to encrypt confidential properties that are used to configure a Worklight Server, such as a database password or a certificate. This tool is documented in “Storing properties in encrypted format” on page 721.
report-templates	Report templates to configure BIRT Reports for your Application Server. They are documented in “Manually configuring BIRT Reports for your application server” on page 875.

Table 9. Files and subdirectories in the ApplicationCenter subdirectory

Item	Description
ApplicationCenter/installer	<b>IbmApplicationCenter.apk</b> The Android version of the Application Center Mobile client.

Table 9. Files and subdirectories in the ApplicationCenter subdirectory (continued)

Item	Description
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/	<p><b>appcenterconsole.war</b> The WAR file for the Application Center console user interface web application.</p> <p><b>applicationcenter.war</b> The WAR file for the Application Center REST services web application.</p> <p><b>applicationcenter.ear</b> The enterprise application archive (EAR) file to be deployed under IBM PureApplication System.</p>
ApplicationCenter/databases	<p><b>create-appcenter-derby.sql</b> The SQL script to recreate the application center database on derby.</p> <p><b>create-appcenter-db2.sql</b> The SQL script to recreate the application center database on DB2.</p> <p><b>create-appcenter-mysql.sql</b> The SQL script to recreate the application center database on MySQL.</p> <p><b>create-appcenter-oracle.sql</b> The SQL script to recreate the application center database on Oracle.</p>

Table 9. Files and subdirectories in the ApplicationCenter subdirectory (continued)

Item	Description
ApplicationCenter/tools	<p><b>android-sdk</b> The directory that contains the part of the Android SDK required by the Application Center console.</p> <p><b>applicationcenterdeploytool.jar</b> The JAR file that contains the Ant task to deploy an application to the Application Center.</p> <p><b>acdeploytool.bat</b> The startup script of the deployment tool for use on Microsoft Windows systems.</p> <p><b>acdeploytool.sh</b> The startup script of the deployment tool for use on UNIX systems.</p> <p><b>build.xml</b> Example of an Ant script to deploy applications to the Application Center.</p> <p><b>dbconvertertool.sh</b> The startup script of the database converter tool for use on UNIX systems.</p> <p><b>dbconvertertool.bat</b> The startup script of the database converter tool for use on Microsoft Windows systems.</p> <p><b>dbconvertertool.jar</b> The main library of the database converter tool.</p> <p><b>lib</b> The directory that contains all Java™ Archive (JAR) files that are required by the database converter tool.</p> <p><b>json4j.jar</b> The required JSon4J Java archive file.</p> <p><b>README.TXT</b> Readme file that explains how to use the deployment tool.</p>

Table 10. Files and subdirectories in the License subdirectory

Item	Description
License-mfce	License for IBM Mobile Foundation Consumer Edition
License-mfee	License for IBM Mobile Foundation Enterprise Edition
License-wce	License for IBM Worklight Consumer Edition

Table 10. Files and subdirectories in the License subdirectory (continued)

Item	Description
License-wee	License for IBM Worklight Enterprise Edition

## Manually installing Application Center

In some cases, you might want to reconfigure Worklight Server so that it uses a different database or schema from the one that was specified during installation of Worklight Server. The way that you do this reconfiguration depends on the type of database and on the kind of application server, as explained in the following topics.

**Note:** Whether you install Application Center with IBM Installation Manager as part of the Worklight Server installation or manually, one point to bear in mind is that "rolling updates" (see "Deciding between in-place upgrade and rolling upgrade" on page 194) of Application Center are not supported. That is, you cannot install two versions of Application Center (for example, V5.0.6 and V6.0.0) that operate on the same database.

### Configuring the DB2 database manually for Application Center

You configure the DB2 database manually by creating the database, creating the database tables, and then configuring the relevant application server to use this database setup.

#### Procedure

1. Create the database. This step is described in "Creating the DB2 database for Application Center" on page 47.
2. Create the tables in the database. This step is described in "Setting up your DB2 database manually for Application Center."
3. Perform the application server-specific setup as listed below.

#### Setting up your DB2 database manually for Application Center:

You can set up your DB2 database for Application Center manually using the procedures in this section.

#### About this task

Set up your DB2 database for Application Center by creating the database schema.

#### Procedure

1. Create a system user, **worklight**, in a DB2 admin group such as **DB2USERS**, by 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.

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 `~/sql1lib/bin` and enter `./db2`.
3. 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
```

4. Invoke DB2 with the following commands to create the **APPCNTR** tables, in a schema named **APPSCHM** (the name of the schema can be changed). This command can be run on an existing database that has a page size compatible with the one defined in step 3.

```
db2 CONNECT TO APPCNTR
db2 SET CURRENT SCHEMA = 'APPSCHM'
db2 -vf <worklight_install_dir>/ApplicationCenter/databases/create-appcenter-db2.sql -t
```

### Configuring Liberty Profile for DB2 manually for Application Center:

If you want to manually set up and configure your DB2 database for Application Center with WebSphere Application Server Liberty Profile, use the following procedure.

#### 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 (`worklightServer` may be replaced in this path by the name of your server) as follows:

```
<library id="DB2Lib">
  <fileset dir="{shared.resource.dir}/db2" includes="*.jar"/>
</library>

<!-- Declare the IBM Application Center database. -->
<dataSource jndiName="jdbc/AppCenterDS" transactional="false">
  <jdbcDriver libraryRef="DB2Lib"/>
  <properties.db2.jcc databaseName="APPCNTR" currentSchema="APPSCHM"
    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 **APPCNTR** 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 for Application Center:

If you want to manually set up and configure your DB2 database for Application Center with WebSphere Application Server, use the following procedure.

### 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 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. Create JAAS-J2C authentication data, specifying the DB2 user name and password for **Container Connection**.
  - g. Select the component-managed authentication alias that you created.
  - h. Click **Next** and **Finish**.
  - i. Click **Save**.
  - j. In **Resources > JDBC > Data sources**, select the new data source.
  - k. Click **WebSphere Application Server data source properties**.
  - l. Select the **Non-transactional data source** check box.
  - m. Click **OK**.
  - n. Click **Save**.



- o. Click **Custom properties** for the datasource, select property **currentSchema**, and set the value to the schema used to create the Application Center tables (APPSCHM in this example).
4. Test the data source connection by selecting **Data Source** and clicking **Test Connection**.

### Configuring Apache Tomcat for DB2 manually for Application Center:

If you want to manually set up and configure your DB2 database for Application Center with Apache Tomcat server, use the following procedure.

#### 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"
    name="jdbc/AppCenterDS"
    username="worklight"
    password="password"
    type="javax.sql.DataSource"
    url="jdbc:db2://server:50000/APPCNTR:currentSchema=APPSCHM;" />
...
</Context>
```

Where **worklight** after **user=** is the name of the system user with "CONNECT" access to the **APPCNTR** database that you have previously created, and **password** after **password=** is this user's password. If you have defined either a different user name, or a different password, or both, replace these entries accordingly.

DB2 has a user name and password length limit of 8 characters for UNIX and Linux systems, and 30 characters for Windows.

### Configuring the Apache Derby database manually for Application Center

You configure the Apache Derby database manually by creating the database and database tables, and then configuring the relevant application server to use this database setup.

#### Procedure

1. Create the database and the tables within them. This step is described in "Setting up your Apache Derby database manually for Application Center" on page 70
2. Configure the application server to use this database setup. Go to one of the following topics:
  - "Configuring Liberty Profile for Derby manually for Application Center" on page 70
  - "Configuring WebSphere Application Server for Derby manually for Application Center" on page 71

- “Configuring Apache Tomcat for Derby manually for Application Center” on page 72

### Setting up your Apache Derby database manually for Application Center:

You can set up your Apache Derby database for Application Center manually using the procedures in this section.

#### About this task

Set up your Apache Derby database for Application Center by creating the database schema.

#### Procedure

1. 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](#).

2. At the command prompt, enter the following commands:

```
connect 'jdbc:derby:APPCNTR;user=APPCENTER;create=true';
run '<worklight_install_dir>/ApplicationCenter/databases/create-appcenter-derby.sql';
quit;
```

### Configuring Liberty Profile for Derby manually for Application Center:

If you want to manually set up and configure your Apache Derby database for Application Center with WebSphere Application Server Liberty Profile, use the following procedure.

#### About this task

Complete the Apache Derby database setup procedure before continuing.

#### Procedure

Configure the data source in the `$LIBERTY_HOME/usr/servers/worklightServer/server.xml` file (`worklightServer` may be replaced in this path by the name of your server) as follows:

```
<!-- Declare the jar files for Derby access through JDBC. -->
<library id="derbyLib">
  <fileset dir="C:/Drivers/derby" includes="derby.jar" />
</library>

<!-- Declare the IBM Application Center database. -->
<dataSource jndiName="jdbc/AppCenterDS" transactional="false" 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 for Application Center:

If you want to manually set up and configure your Apache Derby database for Application Center with WebSphere Application Server, use the following procedure.

### 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/6.0/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**.
  - b. Set **Scope** to **Node level**.
  - c. Click **New**.
  - d. Set **Database Type** to **User-defined**.
  - e. Set **class Implementation name** to `org.apache.derby.jdbc.EmbeddedConnectionPoolDataSource40`.
  - f. Set **Name** to **Worklight - Derby JDBC Provider**.
  - g. Set **Description** to **Derby JDBC provider for Worklight**.
  - h. Click **Next**.
  - i. Set the **Class path** to `WAS_INSTALL_DIR/optionalLibraries/IBM/Worklight/6.0/derby/derby.jar`.
  - j. Click **Finish**.
3. Create the data source for the IBM Worklight database.
  - a. In the WebSphere Application Server console, click **Resources > JDBC > Data sources**.
  - b. Set **Scope** to **Node Level**.
  - c. Click **New**.
  - d. Set **Data source Name** to **Application Center Database**.
  - e. Set **JNDI name** to `jdbc/AppCenterDS`.
  - f. Click **Next**.
  - g. Select the existing JDBC Provider that is named **Worklight - Derby JDBC Provider**.
  - h. Click **Next**.
  - i. Click **Next**.
  - j. Click **Finish**.
  - k. Click **Save**.
  - l. In the table, click the **Application Center Database** datasource that you created.
  - m. Under **Additional Properties**, click **Custom properties**.
  - n. Click **databaseName**.
  - o. Set **Value** to the path to the APPCNTR database that is created in "Setting up your Apache Derby database manually for Application Center" on page 70.

- p. Click **OK**.
- q. Click **Save**.
- r. At the top of the page, click **Application Center Database**.
- s. Under **Additional Properties**, click **WebSphere Application Server data source properties**.
- t. Select **Non-transactional datasource**.
- u. Click **OK**.
- v. Click **Save**.
- w. In the table, select the **Application Center Database** datasource that you created.
- x. Click **test connection** (only if you are not on the console of a WAS Deployment Manager).

### Configuring Apache Tomcat for Derby manually for Application Center:

If you want to manually set up and configure your Apache Derby database for Application Center with the Apache Tomcat server, use the following procedure.

#### 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/AppCenterDS"
    username="APPCENTER"
    password=""
    type="javax.sql.DataSource"
    url="jdbc:derby:DERBY_DATABASES_DIR/APPCNTR"/>
  ...
</Context>
```

### Configuring the MySQL database manually for Application Center

You configure the MySQL database manually by creating the database, creating the database tables, and then configuring the relevant application server to use this database setup.

#### Procedure

1. Create the database. This step is described in "Creating the MySQL database for Application Center" on page 48.
2. Create the tables in the database. This step is described in "Setting up your MySQL database manually for Application Center."
3. Perform the application server-specific setup as listed below.

### Setting up your MySQL database manually for Application Center:

You can set up your MySQL database for Application Center manually using the procedures in this section.

### About this task

Complete the following procedure to set up your MySQL database.

#### Procedure

1. Create the database schema.

- a. Run a MySQL command-line client with options `-u root`.
- b. Enter the following commands:

```
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 APPCNTR;
SOURCE <worklight_install_dir>/ApplicationCenter/databases/create-appcenter-mysql.sql;
```

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. Add the following property to your MySQL option file: `max_allowed_packet=256M`

For more information about option files, see the MySQL documentation at MySQL.

#### Configuring Liberty Profile for MySQL manually for Application Center:

If you want to manually set up and configure your MySQL database for Application Center with WebSphere Application Server Liberty Profile, use the following procedure.

### 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/usr/servers/worklightServer/server.xml` file (worklightServer may be replaced in this path by the name of your server) 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 Application Center database. -->
<dataSource jndiName="jdbc/AppCenterDS" transactional="false">
  <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 for Application Center:

If you want to manually set up and configure your MySQL database for Application Center with WebSphere Application Server, use the following procedure.

### 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. 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.
3. Set the custom properties of the new data source.
  - a. Select the new data source.
  - b. Click **Custom properties**.
  - c. Set the following properties:

```
portNumber = 3306
relaxAutoCommit=true
databaseName = APPCNTR
serverName = the host name of the MySQL server
user = the user name of the MySQL server
password = the password associated with the user name
```
4. Set the WebSphere Application Server custom properties of the new data source.
  - a. In **Resources > JDBC > Data sources**, select the new data source.
  - b. Click **WebSphere Application Server data source properties**.
  - c. Select the **Non-transactional data source** check box.
  - d. Click **OK**.
  - e. Click **Save**.



## Configuring Apache Tomcat for MySQL manually for Application Center:

If you want to manually set up and configure your MySQL database for Application Center with the Apache Tomcat server, use the following procedure.

### About this task

Complete the MySQL database setup procedure before continuing.

### 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/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>
```

## Configuring the Oracle database manually for Application Center

You configure the Oracle database manually by creating the database, creating the database tables, and then configuring the relevant application server to use this database setup.

### Procedure

1. Create the database. This step is described in “Creating the Oracle database for Application Center” on page 49.
2. Create the tables in the database. This step is described in “Setting up your Oracle database manually for Application Center.”
3. Perform the application server-specific setup as listed below.

### Setting up your Oracle database manually for Application Center:

You can set up your Oracle database for Application Center manually using the procedures in this section.

### About this task

Complete the following procedure to set up your Oracle database.

### Procedure

1. Ensure that you have at least one Oracle database. In many Oracle installations, the default database has the SID (name) ORCL. For best results, the character set of the database should be set to **Unicode (AL32UTF8)**.
2. Create the user APPCENTER, either by using Oracle Database Control, or by using the Oracle SQLPlus command-line interpreter.

Create the user for the IBM Worklight database/schema, 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 APPCENTER 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
```

To create the user by using Oracle SQLPlus, enter the following commands:

```
CONNECT system/<system_password>@ORCL
CREATE USER APPCENTER IDENTIFIED BY password;
GRANT CONNECT, RESOURCE, CREATE VIEW TO APPCENTER;
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 Application Center database by running the create-appcenter-oracle.sql file:
 

```
CONNECT APPCENTER/<APPCENTER_password>@ORCL
@<worklight_install_dir>/ApplicationCenter/databases/create-appcenter-oracle.sql
DISCONNECT;
```
4. 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 for Application Center:

If you want to manually set up and configure your Oracle database for Application Center with WebSphere Application Server Liberty Profile, use the following procedure.

#### 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 (worklightServer may be replaced in this path by the name of your server) 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 Application Center database. -->
<dataSource jndiName="jdbc/AppCenterDS" transactional="false">
```

```

<jdbcDriver libraryRef="OracleLib"/>
<properties.oracle driverType="thin"
  serverName="oserver" portNumber="1521"
  databaseName="ORCL"
  user="APPCENTER" password="APPCENTER_password"/>
</dataSource>

```

where **APPCENTER** after **user=** is the user name, **APPCENTER\_password** 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 for Application Center:

If you want to manually set up and configure your Oracle database for Application Center with WebSphere Application Server, use the following procedure.

#### 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 11. 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**.
  - c. Set **JNDI name** to jdbc/AppCenterDS.
  - 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/ORCL**, 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 = APPCENTER**.
- l. Set **password = APPCENTER\_password**.
- m. Click **OK** and save the changes.
- n. In **Resources > JDBC > Data sources**, select the new data source.
- o. Click **WebSphere Application Server data source properties**.
- p. Select the **Non-transactional data source** check box.
- q. Click **OK**.
- r. Click **Save**.

### Configuring Apache Tomcat for Oracle manually for Application Center:

If you want to manually set up and configure your Oracle database for Application Center with the Apache Tomcat server, use the following procedure.

#### 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/AppCenterDS"
    auth="Container"
    type="javax.sql.DataSource"
    driverClassName="oracle.jdbc.driver.OracleDriver"
    url="jdbc:oracle:thin:@oserver:1521/ORCL"
    username="APPCENTER"
    password="APPCENTER_password"/>
...
</Context>
```

Where **APPCENTER** after **username=** is the name of the system user with "CONNECT" access to the **APPCNTR** database that you have previously created, and **APPCENTER\_password** after **password=** is this user's password. If you have defined either a different user name, or a different password, or both, replace these values accordingly.

### Deploying the Application Center WAR files and configuring the application server manually

The procedure to manually deploy the Application Center WAR files manually to an application server depends on the type of application server being configured, as detailed here.

These manual instructions assume that you are familiar with your application server.

**Note:** Using the Worklight Server installer to install Application Center is more reliable than installing manually, and should be used whenever possible.

If you prefer to use the manual process, follow these steps to configure your application server for Application Center. You must deploy the

appcenterconsole.war and applicationcenter.war files to your Application Center. The files are located in `<worklightInstallDir>/ApplicationCenter/console`.

### Configuring WebSphere Application Server Liberty Profile for Application Center manually:

To configure WebSphere Application Server Liberty Profile for Application Center manually, you must modify the `server.xml` file.

#### About this task

In addition to modifications for the databases that are described in “Manually installing Application Center” on page 66, you must make the following modifications to the `server.xml` file.

**Note:** In the following procedure, when the example uses `worklight.war`, it should be the name of your Worklight project, for example, `myProject.war`.

#### 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 for the Application Center:

```
<!-- The directory with binaries of the 'aapt' program, from the Android SDK's platform-to
<jndiEntry jndiName="android.aapt.dir" value="WL_INSTALL_DIR/ApplicationCenter/tools/android-s

<!-- Declare the IBM Application Center Console application. -->
<application id="appcenterconsole" name="appcenterconsole" location="appcenterconsole.war" type="war"
  <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" type="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 120.

## Configuring WebSphere Application Server for Application Center manually:

To configure WebSphere Application Server for Application Center manually, you must configure variables, custom properties, and class loader policies.

## 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. Enable application security.
  - a. Click **Security > Global Security**.
  - b. Ensure that the **Enable administrative security** check box is selected. Application security can only be enabled if administrative security is enabled.
  - c. Ensure that the **Enable application security** check box is selected.
  - d. Click **OK**.
  - e. Save the changes.

For more details, see [http://pic.dhe.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.base.doc/info/aes/ae/tsec\\_csec2.html](http://pic.dhe.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.base.doc/info/aes/ae/tsec_csec2.html).

3. Create the Application Center JDBC data source and provider. See the instructions in the appropriate subsection in “Manually installing Application Center” on page 66.
4. Install the Application Center console 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/ApplicationCenter/console`.
  - c. Select `appcenterconsole.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 **Next** until you reach the Map context roots for web modules page.
  - g. In the **Context Root** field, type `/appcenterconsole`.
  - h. Click **Next**.
  - i. Click **Finish**.
5. 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 **appcenterconsole\_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 **ApplicationCenterConsole** 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 **appcenterconsole\_war** and click **Start**.
6. Repeat step 4, selecting **applicationcenter.war** in sub-step c, and using a **Context Root** of /applicationcenter in sub-step g.
7. Repeat step 5, selecting **applicationcenter.war** from the list of applications in sub-step b.
8. 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)**.
9. Configure a JNDI environment entry to indicate the directory with binaries of the aapt program, from the Android SDK's platform-tools package. For a standalone server:
  - a. Click **Applications > WebSphere enterprise applications**.
  - b. From the list of applications, select **applicationcenter\_war**.
  - c. In the "Web Module Properties" section, select **Environment entries for Web modules**.
  - d. Assign to the variable `android.aapt.dir` the value `WL_INSTALL_DIR/ApplicationCenter/tools/android-sdk` where `WL_INSTALL_DIR` is the Worklight Server installation directory.

For WebSphere Application Server Network Deployment, you must:

- a. Copy the `WL_INSTALL_DIR/ApplicationCenter/tools/android-sdk` directory to a location in the config directory of the deployment manager's profile. This will be propagated to the servers through the file synchronization service; for example, `WAS_INSTALL_DIR/profiles/Dmgr01/config/cells/cell-name/clusters/cluster-name/android-sdk`.
- b. Configure the environment entry `android.aapt.dir` with value `${USER_INSTALL_ROOT}/config/cells/cell-name/clusters/cluster-name/android-sdk`.
- c. Click **System administration > Nodes**, select the nodes, and click **Full Synchronize**.

## Results

You can now access the Application Center at `http://<server>:<port>/appcenterconsole`, 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 Application Center, see “Configuring WebSphere Application Server full profile” on page 119.

## Configuring Apache Tomcat for Application Center manually:

To configure Apache Tomcat for Application Center manually, you must copy JAR and WAR files to Tomcat, add database drivers, edit the `server.xml` file, and then start Tomcat.

### Procedure

1. Add the database drivers to the Tomcat `lib` directory. See the instructions for the appropriate DBMS in “Manually installing Application Center” on page 66.
2. 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 Application Center console and services applications and a user registry:

```
<!-- Declare the IBM Application Center Console application. -->
<Context path="/appcenterconsole" docBase="appcenterconsole">

  <!-- Define the AppCenter services endpoint in order for the AppCenter
  console to be able to invoke the REST service.
  You need to enable this property if the server is behind a reverse
  proxy or if the context root of the Application Center Services
  application is different from '/applicationcenter'. -->
  <!-- <Environment name="ibm.appcenter.services.endpoint"
  value="http://proxy-host:proxy-port/applicationcenter"
  type="java.lang.String" override="false"/>
  -->
</Context>

<!-- Declare the IBM Application Center Services application. -->
<Context path="/applicationcenter" docBase="applicationcenter">

  <!-- The directory with binaries of the 'aapt' program, from
  the Android SDK's platform-tools package. -->
  <Environment name="android.aapt.dir"
  value="WL_INSTALL_DIR/ApplicationCenter/tools/android-sdk"
  type="java.lang.String" override="false"/>

  <!-- The protocol of the application resources URI.
  This property is optional. It is only needed if the protocol
  of the external and internal URI are different. -->
  <!-- <Environment name="ibm.appcenter.proxy.protocol"
  value="http" type="java.lang.String" override="false"/>
  -->

  <!-- The hostname of the application resources URI. -->
  <!-- <Environment name="ibm.appcenter.proxy.host"
  value="proxy-host"
  type="java.lang.String" override="false"/>
  -->
```

```

<!-- The port of the application resources URI.
      This property is optional. -->
<!-- <Environment name="ibm.appcenter.proxy.port"
      value="proxy-port"
      type="java.lang.Integer" override="false"/> -->

<!-- Declare the IBM Application Center Services database. -->
<!-- <Resource name="jdbc/AppCenterDS" type="javax.sql.DataSource" ... -->

</Context>

<!-- 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/realms-howto.html . -->
<Realm className="org.apache.catalina.realm.MemoryRealm"/>

```

where you fill in the <Resource> element as described in one of the sections:

- “Configuring Apache Tomcat for DB2 manually for Application Center” on page 69
- “Configuring Apache Tomcat for Derby manually for Application Center” on page 72
- “Configuring Apache Tomcat for MySQL manually for Application Center” on page 75
- “Configuring Apache Tomcat for Oracle manually for Application Center” on page 78

### 3. Copy the Application Center WAR files 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\ap

```

### 4. Start Tomcat.

#### What to do next

For additional steps to configure the Application Center, see “Configuring Apache Tomcat” on page 121.

---

## Configuring Worklight Server

Consider your backup and recovery policy, optimize your Worklight Server configuration, and apply access restrictions and security options.

### 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.

It is advisable to back up the IBM Worklight database as-is. When Reports are enabled, the database can become quite large. Consider the benefits of backing them up separately. Report tables can be configured to be stored on a different database instance.

## Optimization and tuning of Worklight Server

Optimize the Worklight Server configuration by tuning the allocation of JVM memory, HTTP connections, back-end connections, and internal settings.

For best results, install a Worklight Server on a 64-bit server.

### JVM memory allocation

The Java instance of the application server allocates memory. Consider the following guidelines:

- Set the JVM to have at least 2 GB memory.
- In a production environment, set the minimum heap size and maximum heap size to the same value to avoid heap expansion and contraction.
- 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)

### Tuning HTTP connections

You tune HTTP connections by configuring threading and execution settings for the application server.

Each incoming request requires a thread for the duration of the request. If more simultaneous requests are received than can be handled by the currently available request-processing threads, more threads are created up to the configured maximum.

Apply the following settings:

- Liberty: See the executor section: [http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.wlp.nd.multiplatform.doc%2Fautodita%2Frwlp\\_metatype\\_4ic.html](http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.wlp.nd.multiplatform.doc%2Fautodita%2Frwlp_metatype_4ic.html).
- WebSphere Application Server: Log in to the admin console. Go to **Servers > Server types > WebSphere application servers > server\_name > Web container**. By default, the maximum number of threads is 50.
- Apache Tomcat: See <http://tomcat.apache.org/tomcat-7.0-doc/config/http.html>. By default, the maximum number of threads is 200.

Bear in mind the following points when you configure HTTP threads:

- If, for example, the longest call takes 500 milliseconds and you configure a maximum of 50 threads, you can handle approximately 100 requests per second.
- If your environment includes a back-end system that runs slowly, increase the number of default threads. In addition, increase the number of back-end connection threads (See “Tuning back-end connections” on page 85).

## Tuning back-end connections

Consider the following factors when you set the `maxConcurrentConnectionsPerNode` parameter in the connectivity element of the `adapter.xml` file:

- If the back-end system imposes no limitation on the number of incoming connections, set the number of connection threads per adapter to the number of HTTP threads in the application server. A more refined guideline would be to take account of the percentage of requests to each back-end system.
- If the back-end system imposes a limitation on the number of incoming connection threads, do not exceed  $BACKEND\_MAX\_CONNECTIONS / NUM\_OF\_CLUSTER\_NODES$  connection threads, where `BACKEND_MAX_CONNECTIONS` is the maximum number of incoming connections that are defined in the back-end server, and `NUM_OF_CLUSTER_NODES` is the number IBM Worklight server nodes in the cluster.

## Worklight Server internal configuration

Consider the following factors:

- A session is an object that is stored in server memory for each connecting device. Among other things, it stores authentication information. The number of active sessions is the difference between the number of opened sessions and the number of sessions that are timing out because of inactivity. The `serverSessionTimeout` property configures the session timeout and affects the server memory consumption. The default session timeout is 10 minutes.
- The mobile client "heartbeat" property causes the mobile client to ping the server while the app is in the foreground. This feature prevents the server session from timing out.
- When a mobile app runs in the background, it no longer interacts with the server or sends a "heartbeat". The server session drops after the specified server session timeout period.
- For example, suppose every minute 1,000 users start a session against the server. Even if they exit the application after 3 minutes, their sessions remain active on the server for 10 minutes, leaving  $10 \times 1,000 = 10,000$  active sessions.

The following `worklight.properties` parameters affect the intervals of background tasks that perform various actions on the database and file system:

### **`cluster.data.synchronization.taskFrequencyInSeconds`**

Application and adapter files are read from the file system and are stored in the database that enables the synchronization of the deployment data between all cluster nodes. The parameter controls the synchronization interval of the file system with the database content. Every 2 seconds (the default interval), each Worklight Server node checks the database to see whether new adapters or applications are deployed in another Worklight server node. If new adapters or applications are found, they are deployed to the local file system. If you increase the interval, the database is queried less frequently but the Worklight Server nodes synchronize less frequently with the latest adapters and applications.

### **`deployables.cleanup.taskFrequencyInSeconds`**

Deletes unused deployables from the file system. The default frequency is 24 hours.

### **sso.cleanup.taskFrequencyInSeconds**

The SSO (single sign-on) mechanism stores session data in a database table. This parameter configures the interval for the SSO cleanup task that checks whether there are inactive accounts in the SSO table. If inactive accounts are found, they are deleted. The default frequency is 5 seconds. Accounts are considered inactive if they remain idle for longer than `serverSessionTimeout`.

## **Optimization and tuning of Worklight Server project databases**

General information about ways that you can improve the performance of the project databases or schemas that support Worklight Server is provided in this topic.

The following sections provide general information about database tuning, and techniques you can use to optimize your database performance for IBM Worklight. In the following sections, the examples that are provided are for the IBM DB2 database. If you use MySQL or Oracle, consult that vendor's documentation for the corresponding procedures.

### **Database disks**

You can find some overview information about the Worklight Server project databases in the **Database usage and size** section of the Scalability and Hardware Sizing PDF document. Its accompanying Excel spreadsheet Hardware Calculator can aid you in computing the hardware configuration that is best suited to your planned server environment.

When you are computing your hardware needs, it is a good idea to consider servers that offer multiple disks because the correct use of them when you set up your Worklight Server project databases can greatly improve their performance.

For example, whether you use DB2, MySQL, or Oracle, you can almost always speed up database performance by configuring the database to use separate disks to store its database logs, index, and data. This allows faster access to your data with every transaction because there is no contention resulting from the same disk attempting to write to its log files or access its index at the same time it processes the data transaction.

### **Database compression**

Using your database vendor's compression feature can decrease the size of the database and decrease I/O time.

For example, in tests that were performed on IBM DB2, adding `COMPRESS YES` to the SQL that creates the `APP_ACTIVITY_REPORT` table decreased the size of that table on the disk by a factor of 3 and decreased its I/O time by a factor of 2.

CPU time might increase as a result of this compression, but it was not observed in the tests on the `APP_ACTIVITY_REPORT` table, possibly because most of the activity was INSERTs and the aggregation task was not monitored deeply.



## On DB2, use the `INLINE_LENGTH`

If your database is DB2, consider using the `INLINE_LENGTH` option when creating the tables used for SSO information, and for other tables that contain data that is stored as large objects (LOBs), but which in actuality are not very large at all, usually a few kilobytes in size.

By constraining the size of these LOBs, the performance of LOB data access can be improved by placing the LOB data within the formatted rows on data pages instead of in the LOB storage object. For more information about this technique, see [Inline LOBs improve performance](#).

## Database table partitions

A partition is a division of a logical database table into distinct independent parts. Using table partitions to map each of the table partitions to a different tablespace can enable performance improvements and facilitate purging accumulated data. This suggestion is primarily relevant only to the `APP_ACTIVITY_REPORT` table that holds the majority of the row data.

**Note:** Partitioned tables are not the same thing as a partitioned database (DPF) environment, which is not suggested for use with IBM Worklight.

To show how database partitions can be used, consider this example from DB2:

- A partition is defined on the `ACTIVITY_TIMESTAMP` column in the `APP_ACTIVITY_REPORT` table.
- Each partition contains one day's data.
- The number of partitions is the number of days of data you want to save.
- Each partition is created in a different table space.
- Thus in the SQL example that follows, you create seven partitions in DB2:

```
CREATE TABLESPACE app_act_rep_1;
CREATE TABLESPACE app_act_rep_2;
CREATE TABLESPACE app_act_rep_3;
CREATE TABLESPACE app_act_rep_4;
CREATE TABLESPACE app_act_rep_5;
CREATE TABLESPACE app_act_rep_6;
CREATE TABLESPACE app_act_rep_7;

CREATE TABLE "APP_ACTIVITY_REPORT" (
  "ID" BIGINT NOT NULL ,
  "ACTIVITY" CLOB(1048576) LOGGED NOT COMPACT ,
  "ACTIVITY_TIMESTAMP" TIMESTAMP ,
  "ADAPTER" VARCHAR(254) ,
  "DEVICE_ID" VARCHAR(254) ,
  "DEVICE_MODEL" VARCHAR(254) ,
  "DEVICE_OS" VARCHAR(254) ,
  "ENVIRONMENT" VARCHAR(254) ,
  "GADGET_NAME" VARCHAR(254) ,
  "GADGET_VERSION" VARCHAR(254) ,
  "IP_ADDRESS" VARCHAR(254) ,
  "PROC" VARCHAR(254) ,
  "SESSION_ID" VARCHAR(254) ,
  "SOURCE" VARCHAR(254) ,
  "USER_AGENT" VARCHAR(254) )
  IN app_act_rep_1, app_act_rep_2, app_act_rep_3, app_act_rep_4,
  app_act_rep_5, app_act_rep_6, app_act_rep_7
  PARTITION BY RANGE (ACTIVITY_TIMESTAMP)
```

```
(STARTING FROM ('2013-02-25-00.00.00.000000')
  ENDING AT ('2013-03-04-00.00.00.000000') EXCLUSIVE
  EVERY (1 DAY)
);
```

## Database purge

Now that this high-volume data is allocated to separate tablespaces, the task of periodically purging the data is simplified. This suggestion is also primarily relevant only to the APP\_ACTIVITY\_REPORT table that holds the majority of the row data. The process that is used in this DB2 example is as follows:

- Aggregate data either with an IBM Worklight process or with a client external process.
- When the data is no longer needed (the aggregation task should successfully process the data), it can be deleted.
- The most effective way to delete the data is to delete the partition. In DB2, this data purge can be done by detaching the partition to a temp table, then truncating that temp table and attaching a new day to the partition. The process can be implemented as a scheduled stored procedure process in the database, as in the following example:

```
ALTER TABLE "APP_ACTIVITY_REPORT"
  DETACH PARTITION part0
  INTO temptable;

TRUNCATE TABLE temptable;

ALTER TABLE "APP_ACTIVITY_REPORT"
  ATTACH PARTITION part0
  STARTING FROM ('2013-02-25-00.00.00.000000')
  ENDING AT ('2013-03-26-00.00.00.000000') EXCLUSIVE
  FROM temptable;
```

## Security configuration

Configure the security of the Worklight Server as detailed here.

### Database and certificate security passwords

When you configure a Worklight Server, you must typically configure database and certificate passwords for security.

Configuration of a 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 in JNDI properties of the application server. Defaults can be stored in the `worklight.properties` file. See “Configuration of IBM Worklight applications on the server” on page 714 for information about individual properties.

You can encrypt any or all of these passwords. For more information, see “Storing properties in encrypted format” on page 721.

### 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. 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 authentication  
This document is provided "as is" documentation.  
In case of issues, refer to the online user documentation.

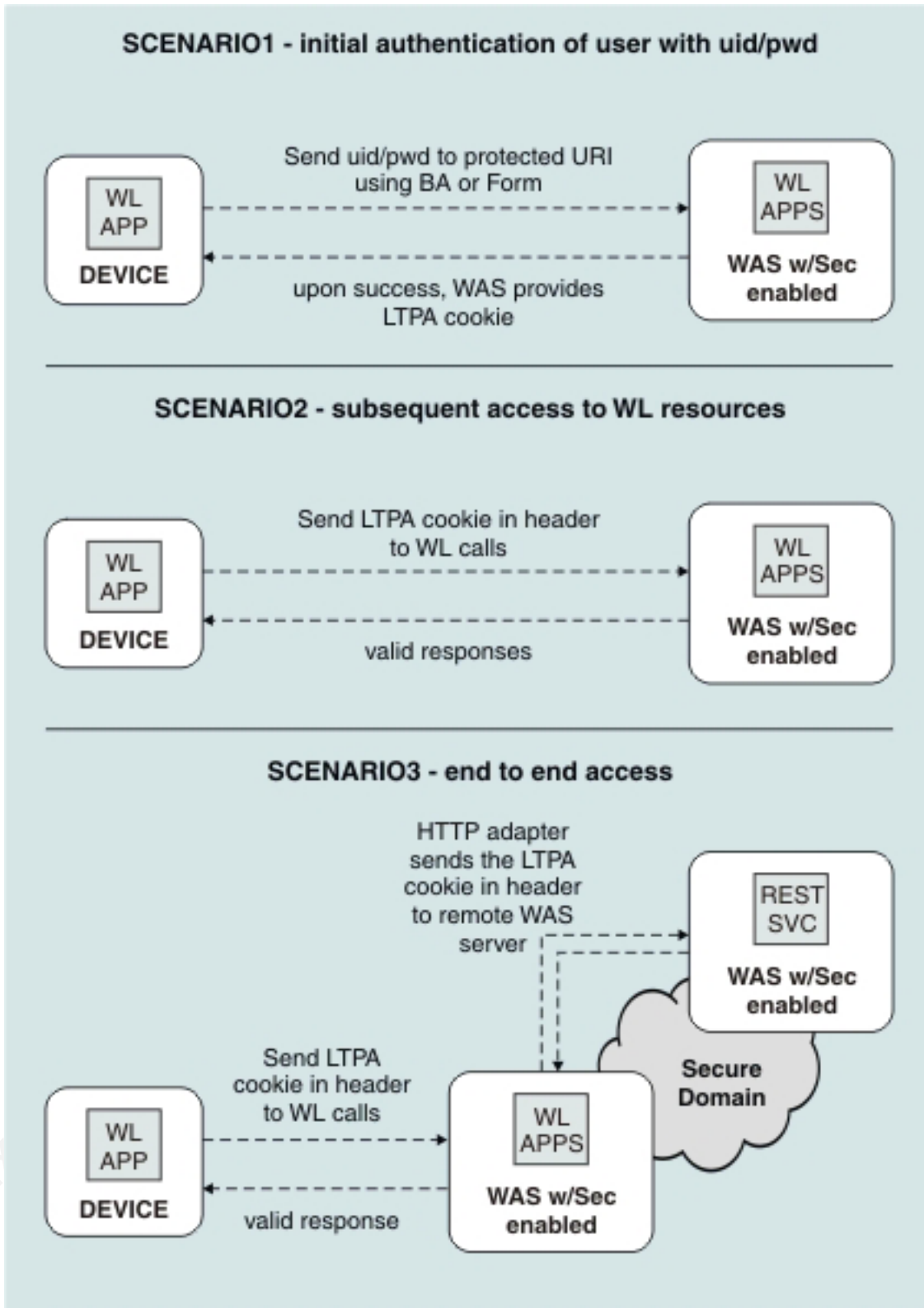


Figure 1. Phases in securing an IBM Worklight app

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 12. WebSphere Application Security Options

	Option 1	Option 2
<b>BENEFITS</b>	<p>Uses the traditional WebSphere Application Server authentication and trust model.</p> <p>The container enforces all security, so it can use existing third-party SSO products to secure the Java Platform, Enterprise Edition container.</p>	<p>Uses the traditional WebSphere Application Server authentication and trust model without the impact of modifying the IBM Worklight Project WAR.</p> <p>The container enforces all security, so it can use existing third-party SSO products to secure the Java Platform, Enterprise Edition container.</p> <p>The layered authentication of device, application, application instance, and user functions as intended.</p> <p>Flexibility in configuring specific security settings that are specific to the IBM Worklight runtime environment without being hindered by the underlying container security.</p>
<b>USAGE</b>	<p>Suitable for scenarios where the devices can be trusted and access for rogue applications is restricted.</p>	<p>Suitable for scenarios where the devices or the apps on the devices cannot be trusted. The multi-step authenticity checking built into IBM Worklight ensures denial of service to devices subjected to unauthorized modifications, rogue applications, and unauthorized users.</p>

## 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.

### 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 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 Worklight Server installation on WebSphere Application Server. The 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 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 what you 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 Worklight Server WAR file level and authenticates users within the 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 Worklight Server



WAR file level and authenticates users within the Worklight Server runtime environment.

### Procedure

1. Complete the same steps as for the Security Option 1, but do not secure the WAR file.

To secure WebSphere Application Server and secure the 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 n
    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";
};

```

- Restart WebSphere Application Server for the modification of the app.policy file to take effect.

## Transmitting IBM Worklight data on the BlackBerry Enterprise Server MDS channel

If you install IBM Worklight in an environment that includes a BlackBerry Enterprise Server, you can use the BlackBerry MDS channel to transmit IBM Worklight data.

### About this task

Figure 2 shows an environment in which apps that are installed on BlackBerry devices transmit data by using the BlackBerry MDS channel. When you install IBM Worklight in environments such as these, you can configure IBM Worklight data to use the same channel.

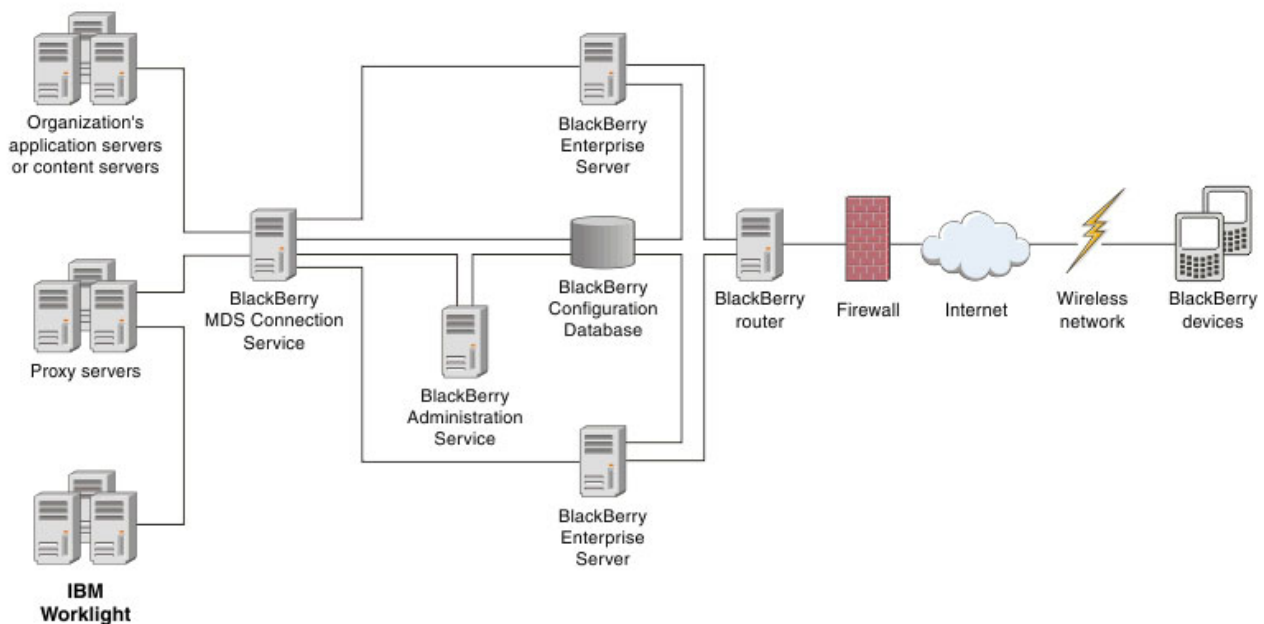


Figure 2. IBM Worklight with BlackBerry Enterprise Server

### Procedure

On the BlackBerry Enterprise Server, configure an MDS connection service to the IBM Worklight Server or to its intermediary proxy server. For information about how to configure an MDS connection service, see the BlackBerry Enterprise Server documentation.

# Protecting your mobile application traffic by using IBM WebSphere DataPower as a security gateway

You can use IBM WebSphere DataPower® in the DMZ of your enterprise to protect Worklight mobile application traffic.

## Before you begin

Ensure that Worklight Studio is installed, and establish your stand-alone server environment on Liberty or WebSphere Application Server before you start this procedure.

## About this task

Protecting mobile application traffic that comes into your network from customer and employee devices involves taking precautions to prevent the data from being altered, authenticating users, and allowing only authorized users to access applications. You can use the security gateway features of IBM WebSphere DataPower to protect mobile application traffic that is initiated by a client IBM Worklight application.

Enterprise topologies are designed to include different zones of protection so that specific processes can be secured and optimized. You can use IBM WebSphere DataPower in different ways in the DMZ and in other zones within your network to protect enterprise resources. As you start to build out Worklight applications to be delivered to the devices of your customers and employees, these methods can be applied to mobile traffic. The following procedure demonstrates the use of IBM WebSphere DataPower as a front-end reverse proxy and security gateway. It uses a multi-protocol gateway (MPGW) service to proxy and secure access to Worklight mobile applications. Two alternative authentication options are demonstrated: HTTP basic authentication and HTML forms-based login between the mobile client and DataPower.

Consider adopting the following phased approach to establishing IBM WebSphere DataPower as a security gateway:

1. Install and configure an IBM Worklight environment and test the installation with a simple application without DataPower acting as the reverse proxy. Test that your application logic works.
2. Configure an MPGW on DataPower to proxy the mobile application or the Worklight console. Part of the configuration involves selecting one of the following authentication options:
  - Use basic authentication for end user authentication with AAA, and generate a single sign-on (SSO) LTPA token for Worklight Server running on WebSphere Application Server if the user successfully authenticates.
  - Use HTML form-based login with AAA, and generate a single sign-on (SSO) LTPA token for Worklight Server, running on WebSphere Application Server if the user successfully authenticates.
3. Test the reverse proxy:
  - Update the Worklight configuration on the server with the reverse proxy configuration (described later in this procedure).
  - Update the mobile security test configuration of each mobile application to use forms-based authentication so that the application requests the user to authenticate immediately upon application startup. Either HTTP basic authentication or HTML forms-based login is supported before the

application starts. For web widgets, widget resources are only accessible to the browser after a user has successfully authenticated.

## Procedure

1. Set up Worklight-specific configuration.
  - a. For each app that you are configuring, modify the authenticationConfig.xml file on the server to include the following security test, realm, and login module declarations:

```
<securityTests>
  <mobileSecurityTest name="WASSTest-securityTest">
    <testDeviceId provisioningType="none"/>
    <testUser realm="WASLTPARealm"/>
  </mobileSecurityTest>
  <webSecurityTest name="WASSTest-web-securityTest">
    <testUser realm="WASLTPARealm"/>
  </webSecurityTest>
</securityTests>

<realms>
  <!-- For websphere -->
  <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"/>
  </realm>
</realms>

<loginModules>
  <!-- For websphere -->
  <loginModule name="WASLTPAModule">
    <className>com.worklight.core.auth.ext.WebSphereLoginModule</className>
  </loginModule>
</loginModules>
```

The authenticationConfig.xml file is usually available in this directory:  
<WAS\_INSTALL\_DIR>/profiles/<WAS\_PROFILE>/installedApps/<WAS\_CELL>/  
IBM\_Worklight\_Console.ear/worklight.war/WEB-INF/classes/conf.

- b. Restart the Worklight Console enterprise application.
2. Update your client mobile app.
    - a. In your client mobile app, add the following JavaScript to your HTML Worklight application:

```
function showLoginScreen() {
  $("#index").hide();
  $("#authPage").show();
}

function showMainScreen() {
  $("#authPage").hide();
  $("#index").show();
}

var myChallengeHandler = WL.Client.createChallengeHandler("WASLTPARealm");
var lastRequestURL;

myChallengeHandler.isCustomResponse = function(response) {

  //A normal login form has been returned
  var findError = response.responseText.search("DataPower/Worklight Error");
  if(findError >= 0)
  {
    return true;
  }
}
```

```

//A normal login form has been returned
var findLoginForm = response.responseText.search("DataPower/Worklight Form Login");
if(findLoginForm >= 0)
{
    lastRequestURL = response.request.url;
    return true;
}

//This response is a worklight server response, handle it normally
return false;
};

myChallengeHandler.handleChallenge = function(response) {
    showLoginScreen();
};

challengeHandler1.handleFailure = function(response) {
    console.log("Error during WL authentication.");
};

myChallengeHandler.submitLoginFormCallback = function(response) {
    var isCustom = myChallengeHandler.isCustomResponse(response);
    if(isCustom) {
        myChallengeHandler.handleChallenge(response);
    }
    else {
        //hide the login screen, you are logged in
        showMainScreen();

        myChallengeHandler.submitSuccess();
    }
};

//When the login button is pressed, submit a login form
$("#loginButton").click(function(){
    var reqURL = "/j_security_check";
    alert(lastRequestURL);
    var options = {method: "POST"};
    options.parameters = {
        j_username: $("#username").val(),
        j_password: $("#password").val(),
        originalUrl : lastRequestURL,
        login: "Login"
    };

    options.headers = {};
    myChallengeHandler.submitLoginForm(reqURL, options, myChallengeHandler);

});

```

- b. If you want to retrieve the LTPA key file used for authentication from the Worklight Server, you can also use the Worklight API function “WL.Client.login” on page 517: `WL.Client.login("WASLTPARealm");`

This call triggers the **myChallengeHandler.isCustomResponse** method with a JSON response, where you can retrieve the LTPA key file.

```

if (response.responseJSON.WASLTPARealm && response.responseJSON.WASLTPARealm.isUserAuthenticated)
var sessionKey = response.responseJSON.WASLTPARealm.attributes.LtpaToken;

```

For any subsequent adapter calls that need to be proxied through the reverse proxy, you can include this `sessionKey` as a header within the request.

Ensure that the HTML body for your Worklight app reflects the login information that is to be handled by DataPower.

- c. To add the authentication test to an application or device, add a securityTest attribute to the environment's tag in the application-descriptor.xml file in your project to use the security test you declared in the authenticationConfig.xml file on the server side in step 1a. Here is an iPad example:

```
<ipad bundleId="com.Datapower" securityTest="WASTest-securityTest" version="1.0">
  <worklightSettings include="true"/>
  <security>
    <encryptWebResources enabled="false"/>
    <testWebResourcesChecksum enabled="false" ignoreFileExtensions="png, jpg, jpeg, gif, mp4,
  </security>
</ipad>
```

3. Define a multi-protocol gateway.
- In the IBM DataPower WebGUI, in the search box under Control Panel, enter Multi-Protocol, and then click **New Multi-Protocol Gateway**.
  - On the **General Configuration** page, provide the following details:

Table 13. General Configuration

Field	Description
Multi-Protocol Gateway Name	Provide a name for your gateway.
Response Type	Select <b>Non-XML</b> . This allows HTTP Web application traffic (including JSON, JavaScript, and CSS) to pass through the appliance.
Request Type	Select <b>Non-XML</b> . This allows HTTP Web application requests to be handled by the appliance.
Front Side Protocol	<p>Select <b>HTTPS (SSL)</b>. For this type of interaction in which user credentials are passed between the client and server, HTTPS is appropriate. Provide the following additional front-side handler details:</p> <p><b>Name</b> Enter a name for the configuration.</p> <p><b>Port Number</b> Enter a number for the listening port. This port number must match the port number that you specify if you define an AAA policy that uses HTML forms-based authentication. (See Table 15 on page 99.)</p> <p><b>Allowed Methods and Versions</b> Select the <b>GET method</b> check box to enable support for HTTP Get.</p> <p><b>SSL Proxy</b> Select an SSL Reverse Proxy profile to identify the SSL server.</p>



Table 13. General Configuration (continued)

Field	Description
Multi-Protocol Gateway Policy	<p>Click +, and then create rules to define the policies listed in the following topics depending on the type of authentication you decide to use:</p> <ul style="list-style-type: none"> <li>• Policy worklight-basicauth for HTTP basic authentication: see “Rules for HTTP basic authentication” on page 100.</li> <li>• Policy mpgw-form for HTML form-based login authentication: see “Rules for HTML forms-based authentication” on page 102.</li> </ul>
Backend URL	Specify the address and port of the Worklight Server that is hosted on the WebSphere Application Server.

4. Create an AAA policy that supports the HTTP basic authentication or HTML forms-based login policy you defined in the previous step.
  - a. In the IBM DataPower WebGUI, in the search box under Control Panel, enter AAA, and then click the **Add** button.
  - b. Provide information depending on the type of authentication you want to use:
    - For HTTP basic authentication, provide the information listed in the following table:

Table 14. AAA policy for HTTP basic authentication

Phase	Description
Extract Identity	In the <b>Methods</b> field, select <b>HTTP Authentication Header</b> .
Authenticate	Choose the authentication method. If WebSphere Application Server is using LDAP, configure LDAP here.
Extract Resource	Select <b>URL Sent by Client</b> .
Post processing	Generate an LTPA token. Specify <b>LTPA Token Expiry</b> , <b>LTPA Key File</b> , and <b>LTPA Key File Password</b> .

- For HTML forms-based login, provide the information listed in the following table:

Table 15. AAA policy for HTML forms-based authentication

Phase	Description
Extract Identity	In the <b>Methods</b> field, select <b>HTML Forms-based Authentication</b> . Select or create an HTML forms-based policy that has the <b>Use SSL for Login</b> option enabled, assigns <b>SSL Port</b> to the port number on which the MPGWS is listening (that was specified in step 3), and has the <b>Enable Session Migration</b> option disabled.
Authenticate	Choose the authentication method. If WebSphere Application Server is using LDAP, configure LDAP here.

Table 15. AAA policy for HTML forms-based authentication (continued)

Phase	Description
Extract Resource	Select <b>URL Sent by Client</b> .
Post processing	Generate an LTPA token. Specify <b>LTPA Token Expiry, LTPA Key File, and LTPA Key File Password</b> .

5. On the Advanced page, specify the advanced settings listed in the following table.

Table 16. Advanced settings

Field	Value
Persistent Connections	<b>On</b> .
Allow Cache-Control Header	<b>Off</b>
Loop Detection	<b>Off</b>
Follow Redirects	<b>Off</b> . This prevents the DataPower back-end user agent from resolving redirects from the back-end. Web applications typically require a client browser to resolve redirects so that they can maintain the context for "directory" along with setting an LTPA cookie on the client.
Allow Chunked Uploads	<b>Off</b>
MIME Back Header Processing	<b>Off</b>
MIME Front Header Processing	<b>Off</b>

## Results

Your Worklight mobile application traffic is now protected by an IBM WebSphere DataPower secure gateway. Authentication is enforced on the DataPower device and the credentials (header or LTPA token) are forwarded downstream to Worklight Server to establish the user identity as part of the mobile traffic.

### Rules for HTTP basic authentication

Add rules to define an HTTP basic authentication policy named `worklight-basicauth`.

You create the `worklight-basicauth` policy as part of the process of defining a multi-protocol gateway. See "Protecting your mobile application traffic by using IBM WebSphere DataPower as a security gateway" on page 95, Table 13 on page 98.

Table 17. HTTP Basic Authentication properties

Property	Value
Policy Name	<code>worklight-basicauth</code>

Table 17. HTTP Basic Authentication properties (continued)

Property	Value
Order of configured rules	<ol style="list-style-type: none"> <li>1. worklight-basicauth_rule_0: see Table 18</li> <li>2. worklight-basicauth_rule_3: see Table 21</li> <li>3. worklight-basicauth_rule_1: see Table 19</li> <li>4. worklight-basicauth_rule_2: see Table 20</li> </ol>

Table 18. Properties of worklight-basicauth\_rule\_0

Property	Value
Direction	<b>Client to Server or Both Directions.</b>
Match	<ul style="list-style-type: none"> <li>• Type = URL</li> <li>• Pattern = /favicon.ico</li> </ul>
Advanced	"Set Variable" -> var://service/mpgw/skip-backside = 1
Result	Not applicable.

Table 19. Properties of worklight-basicauth\_rule\_1

Property	Value
Direction	<b>Client to Server.</b>
Match	<ul style="list-style-type: none"> <li>• Type = URL</li> <li>• Pattern = *</li> </ul>
AAA	BasicAuth2LTPA <ul style="list-style-type: none"> <li>• Output: NULL</li> </ul>
Result	Not applicable.

Table 20. Properties of worklight-basicauth\_rule\_2

Property	Value
Direction	<b>Server to Client.</b>
Match	<ul style="list-style-type: none"> <li>• Type = URL</li> <li>• Pattern = *</li> </ul>
Filter	Provide a custom stylesheet that handles redirect and content-type rewrite. For a sample redirect stylesheet, see "Sample redirect stylesheet" on page 105. <ul style="list-style-type: none"> <li>• Output: NULL</li> </ul>
Result	Not applicable.

Table 21. Properties of worklight-basicauth\_rule\_3

Property	Value
Direction	<b>Client to Server.</b>

Table 21. Properties of *worklight-basicauth\_rule\_3* (continued)

Property	Value
Match	<ul style="list-style-type: none"> <li>Type = HTTP</li> <li>HTTP header tag = Cookie</li> <li>HTTP value match = *LtpaToken*</li> </ul>
AAA	VerifyLTPA <ul style="list-style-type: none"> <li>Output: NULL</li> </ul>
Result	Not applicable.

## Rules for HTML forms-based authentication

Add rules to define an HTML forms-based authentication policy named `mpgw-form`.

You create the `mpgw-form` policy as part of the process of defining a multi-protocol gateway. See “Protecting your mobile application traffic by using IBM WebSphere DataPower as a security gateway” on page 95, Table 13 on page 98.

Table 22. HTTP Form-Based Login properties

Property	Value
Policy Name	<code>mpgw-form</code>
Order of configured rules	<ol style="list-style-type: none"> <li><code>mpgw-form_rule_0</code>: see Table 23</li> <li><code>mpgw-form_rule_1</code>: see Table 24</li> <li><code>mpgw-form_rule_2</code>: see Table 25 on page 103</li> <li><code>mpgw-form_rule_3</code>: see Table 26 on page 103</li> <li><code>mpgw-form_rule_6</code>: see Table 27 on page 103</li> </ol>

Table 23. Properties of *mpgw-form\_rule\_0*

Property	Value
Direction	<b>Client to Server or Both Directions.</b>
Match	<ul style="list-style-type: none"> <li>Type = URL</li> <li>Pattern = <code>/favicon.ico</code></li> </ul>
Advanced	"Set Variable" -> <code>var://service/mpgw/skip-backside = 1</code>
Result	Not applicable.

Table 24. Properties of *mpgw-form\_rule\_1*

Property	Value
Direction	<b>Client to Server.</b>
Match	<ul style="list-style-type: none"> <li>Type = HTTP</li> <li>HTTP header tag = Cookie</li> <li>HTTP value match = *LtpaToken*</li> </ul>
AAA	VerifyLTPA <ul style="list-style-type: none"> <li>Output: NULL</li> </ul>
Result	Not applicable.

Table 25. Properties of *mpgw-form\_rule\_2*

Property	Value
Direction	<b>Client to Server.</b>
Match	<ul style="list-style-type: none"> <li>• Match with PCRE = on</li> <li>• Type = URL</li> <li>• Pattern = <code>/(Login Error)Page\.htm(1)?(\?originalUrl=.*)?</code></li> </ul>
Transform	<p>Provide a custom stylesheet that builds either a Login or Error HTML page. For a sample stylesheet, see “Sample form login stylesheet.”</p> <p><b>Note:</b> The HTML Login Form policy allows you to specify whether you retrieve the login and error pages from DataPower or from the back-end application server.</p>
Advanced	Select the <b>set-var</b> action and specify the service variable: <code>var://service/routing-url</code> and value with the endpoint of your login page.
Result	Not applicable.

Table 26. Properties of *mpgw-form\_rule\_3*

Property	Value
Direction	<b>Client to Server.</b>
Match	<ul style="list-style-type: none"> <li>• Type = URL</li> <li>• Pattern = *</li> </ul>
Advanced	"Convert Query Parameter to XML". Accept default values for other selections.
AAA	Form2LTPA

Table 27. Properties of *mpgw-form\_rule\_6*

Property	Value
Direction	<b>Server to Client.</b>
Match	<ul style="list-style-type: none"> <li>• Type = URL</li> <li>• Pattern = *</li> </ul>
Filter	<p>Provide a custom stylesheet that handles redirect and content-type rewrite. For a sample redirect stylesheet, see “Sample redirect stylesheet” on page 105.</p> <ul style="list-style-type: none"> <li>• Output: NULL</li> </ul>
Result	Not applicable.

### Sample form login stylesheet

You can use this sample stylesheet to generate the HTML form login page or error page when creating rules to define an HTML forms-based authentication policy.

You provide a custom stylesheet when defining rule *mpgw-form\_rule\_2*. See “Rules for HTML forms-based authentication” on page 102, Table 25.

```

<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0" xmlns:dp="http://www.
  <xsl:output method="html" omit-xml-declaration="yes" />
  <xsl:template match="/">
    <xsl:choose>
      <xsl:when test="contains(dp:variable('var://service/URI' ), 'LoginPage.htm')">
        <xsl:variable name="uri_temp" select="dp:decode( dp:variable('var://service/URI' ), 'url')">
          <xsl:variable name="uri">
            <xsl:choose>
              <xsl:when test="contains($uri_temp, 'originalUrl')">
                <xsl:value-of select="$uri_temp" />
              </xsl:when>
              <xsl:otherwise>
                <xsl:value-of select="dp:decode(dp:http-request-header('Cookie'), 'url')" />
              </xsl:otherwise>
            </xsl:choose>
          </xsl:variable>
          <xsl:variable name="redirect_uri_preprocess">
            <xsl:for-each select="re:match($uri, '(.*)originalUrl=(.*)')">
              <xsl:if test="position()=3">
                <xsl:value-of select="." />
              </xsl:if>
            </xsl:for-each>
          </xsl:variable>
          <xsl:variable name="redirect_uri">
            <xsl:choose>
              <xsl:when test="contains($redirect_uri_preprocess, ';')">
                <xsl:value-of select="substring-before($redirect_uri_preprocess, ';')" />
              <xsl:otherwise>
                <xsl:value-of select="$redirect_uri_preprocess" />
              </xsl:otherwise>
            </xsl:choose>
          </xsl:variable>
          <html>
            <head>
              <meta http-equiv="Pragma" content="no-cache" />
              <title>Login Page</title>
            </head>
            <body>
              <h2>DataPower/Worklight Form Login</h2>
              <form name="LoginForm" method="post" action="j_security_check">
                <p>
                  <strong>Please enter your user ID and password.</strong>
                  <br />
                  <font size="2" color="grey">If you have forgotten your user ID or password, please con
                </p>
                <p>
                  <table>
                    <tr>
                      <td>User ID:</td>
                      <td>
                        <input type="text" size="20" name="j_username" />
                      </td>
                    </tr>
                    <tr>
                      <td>Password:</td>
                      <td>
                        <input type="password" size="20" name="j_password" />
                      </td>
                    </tr>
                  </table>
                </p>
                <p>
                  <input type="hidden" name="originalUrl">
                    <xsl:attribute name="value">
                      <xsl:value-of select="$redirect_uri" />
                </p>

```



```

        </xsl:attribute>
    </input>
    <input type="submit" name="login" value="Login" />
</p>
</form>
</body>
</html>
</xsl:when>
<xsl:otherwise>
    <!-- error -->
    <html>
    <head>
        <meta http-equiv="Pragma" content="no-cache" />
        <title>Error Page</title>
    </head>
    <body>
        <h2>DataPower/Worklight Error</h2>
        <strong>doom!!</strong>
        unless you provide a valid user identity!
    </body>
    </html>
    </xsl:otherwise>
</xsl:choose>
<dp:set-response-header name="Content-Type" value="text/html" />
<dp:set-variable name="var://service/mpgw/skip-backside" value="true()" />
</xsl:template>
</xsl:stylesheet>

```

## Sample redirect stylesheet

You can use this sample stylesheet to handle redirect and content-type rewrite when creating rules to define an HTTP basic authentication policy or an HTML forms-based authentication policy.

You provide a custom stylesheet when defining rule `mpgw-form_rule_6` (see “Rules for HTML forms-based authentication” on page 102, Table 27 on page 103), and when defining rule `worklight-basicauth_rule_2` (see “Rules for HTTP basic authentication” on page 100, Table 20 on page 101).

```

<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:dp="http://www.datapower.com/extensions"
  xmlns:re="http://exslt.org/regular-expressions"
  extension-element-prefixes="dp re"
  exclude-result-prefixes="dp">
  <xsl:template match="/">
    <xsl:choose>
      <xsl:when test="dp:responding()">
        <xsl:variable name="code">
          <xsl:choose>
            <xsl:when test="dp:http-response-header('x-dp-response-code') != ''">
              <xsl:value-of select="substring(dp:http-response-header('x-dp-response-code'), 1, 3)" />
            </xsl:when>
            <xsl:otherwise>
              <xsl:value-of select="substring(dp:variable('var://service/error-headers'), 10, 3)" />
            </xsl:otherwise>
          </xsl:choose>
        </xsl:variable>
      </xsl:when>
      <xsl:when test="$code = '302'">
        <xsl:variable name="dphost" select="dp:http-request-header('Host')"/>
        <xsl:variable name="host" select="$dphost"/>
        <xsl:variable name="location" select="dp:http-response-header('Location')"/>
        <xsl:variable name="location_host">
          <xsl:for-each select="re:match($location, '(\w+):\\/(\\/([^\s/]+)\/)'">

```

```

        <xsl:if test="position()=3">
            <xsl:value-of select="." />
        </xsl:if>
    </xsl:for-each>
</xsl:variable>
<xsl:variable name="location_final">
    <xsl:value-of select="re:replace($location, $location_host, 'g', $host)" />
</xsl:variable>
<dp:set-http-response-header name="'Location'" value="$location_final" />
</xsl:when>
<xsl:otherwise>
    <xsl:variable name="orig-content" select="dp:variable('var://service/original-response-headers/Content-Type')"/>
    <xsl:if test="$orig-content != ''">
        <dp:set-http-response-header name="'Content-Type'" value='$orig-content' />
    </xsl:if>
</xsl:otherwise>
</xsl:choose>

<!-- the following prevent DataPower from overriding the
     response code coming back from WorkLight Server
-->
<dp:set-response-header name="'x-dp-response-code'" value="'-1'"/>

</xsl:when>
<xsl:otherwise/>
</xsl:choose>
</xsl:template>
</xsl:stylesheet>

```

## Configuring SSL between IBM Worklight Servers and clients by using certificates that are not signed by a trusted certificate authority

Getting SSL to work with certificates that are not signed by a known public certificate authority (CA) can be challenging. Each mobile platform has its own peculiarities, and adheres to and enforces different portions of the transport layer security (TLS) standard at different times.

**Note:** The following recommendations focus mostly on the iOS and Android environments. Support for X.509 certificates comes from the individual platforms, and not from IBM Worklight. For more information about specific requirements for X.509 certificates, see each mobile platform's documentation.

If you have difficulties with getting your application to access a IBM Worklight Server because of SSL-related issues, the likely cause is a bad server certificate. Another likely cause is a client that is not properly configured to trust your server. Many other reasons can cause an SSL handshake to fail, so not all possibilities are covered. Some hints and tips are provided to troubleshoot the most basic issues that are sometimes forgotten or overlooked. These issues are important when you deal with the mobile world and X.509 certificates.

### Basic concepts

A CA is an entity that issues certificates. A CA can issue (sign) other certificates or other CA certificates (intermediate CA certificates).

In a public key infrastructure (PKI), certificates are verified by using a hierarchical chain of trust. The topmost certificate in this tree is the root CA certificate.

You can purchase your certificates from a public Internet CA or operate your own private (local) CA to issue private certificates for your users and applications. A CA is meant to be an authority that is well-trusted by your clients. Most commercial CAs issue certificates that are automatically trusted by most web browsers and mobile platforms. Using private CAs means that you must take certain actions to ensure that the client trusts certificates that are signed by your root CA.

A certificate can be signed (issued) by one of the many public CAs that are known by your mobile platforms, a private CA, or by itself.

A self-signed certificate is a certificate that is signed by itself and has no CA that attests to its validity.

A self-signed CA is signed by itself. It is both a certificate and a CA. Because it is the topmost certificate in a tree, it is also the root CA.

Using certificates that are signed by private CAs is not recommended for production use on external Internet-facing servers because of security concerns. However, they might be the preferred option for development and testing environments due to their low cost. They are also often appropriate for internal (intranet) servers as they can be deployed quickly and easily.

Using self-signed certificates is not recommended because most mobile platforms do not support their use.

## Certificate types that are supported by different mobile platforms

Table 28. Certificate types that are supported by different mobile platforms

Platform	self-signed certificates	self-signed CA certificates	certificates that are signed by a private CA	certificates that are signed by a public CA
iOS	-	⊆	⊆	⊆
Android	-	⊆	⊆	⊆

## Self-signed certificates versus self-signed CAs

When you are dealing with mobile clients, the use of self-signed certificates is not recommended because mobile platforms, like Android and iOS, for example, do not allow the installation of these types of certificates onto the device's truststore. This restriction makes it impossible for the client to ever trust the server's certificate. Although self-signed certificates are often recommended for development and testing purposes, they will not work when the client is a mobile device.

The alternative is to use self-signed CA certificates instead of self-signed certificates. Self-signed CA certificates are as easy to acquire and are also as cost-effective of a solution.

You can create a self-signed CA with most tools. For example, the following command uses the `openssl` tool to create a self-signed CA:

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout privateKey.key -out certificate.crt -re
```

**Note:** X.509 version 1 certificates are not allowed by some mobile platforms. You must use X.509 version 3 certificates instead. If you are generating self-signed CA certificates, ensure that they are of the type X.509 version 3, and have the following extension defined: `basicConstraints = CA:TRUE`. See the appropriate tool's documentation for how to specify the required version and certificate extensions. For `openssl` commands, you can specify the `-reqexts v3_req` flag to indicate version 3 X.509 certificates, and the `-extensions v3_ca` flag to indicate that the certificate is also a CA.

You can check the certificate version and extensions by running the following `openssl` command:

```
openssl x509 -in certificate.crt -text -noout
```

## Establishing trust on the client

When you open a web page on your mobile browser or connect directly to your IBM Worklight Server on an HTTPS port, a client receives a server certificate in the SSL handshake. The client then evaluates the server certificate against its list of known and trusted CAs to establish trust. Each mobile platform includes a set of trusted CAs that are deemed trustworthy for issuing SSL certificates. Trust is established if your server certificate is signed by a CA that is already trusted by the device. After trust is established, the SSL handshake is successful and you are allowed to open the web page on a browser or connect directly to your server.

However, if your server uses a certificate that is signed by a CA that is unknown to the client, the trust cannot be established, and your SSL handshake fails. To ensure your client device trusts your server's certificate, you must install the trust anchor certificate (root CA) on the client device.

**Note:** Only the root CA certificate (trust anchor) needs to be installed. You do not need to install any other certificates, such as intermediaries, on the device.

For iOS, see "Installing the root CA on iOS" on page 111.

For Android, see "Installing the root CA on Android" on page 114.

## Configuring Android

It is important to note that the following flag is set to `true` by default in all IBM Worklight hybrid applications:

```
android:debuggable="true"
```

Setting the flag to `true` tells Android to ignore SSL errors under certain conditions. The use of this flag is highly discouraged for production environments. It is not necessary if you properly configured your server with a certificate that is signed by a CA that is trusted by your client device.

## Handling the certificate chain

If you are using a server certificate that is not signed by itself, you must ensure that the server sends the full certificate chain to the client.

For the client to validate the certificate path, it must have access to the full certificate chain. To ensure that the client has access to the full certificate chain, including intermediate certificates, ensure that all the certificates in the chain are in the server-side keystore file.

For the WebSphere Application Server Liberty profile, see “Updating your keystore and Liberty profile configuration to use a certificate chain” on page 115.

## Handling certificate extensions

RFC 5280 (and its predecessors) defines a number of certificate extensions that provide extra information about the certificate. Certificate extensions provide a means of expanding the original X.509 certificate information standards.

When an extension is specified in an X.509 certificate, the extension must specify whether it is a critical or non-critical extension. A client that is processing a certificate with a critical extension that the client does not recognize, or which the client cannot process, must reject the certificate. A non-critical extension can be ignored if it is not recognized.

Not all mobile platforms recognize or process certain certificate extensions in the same manner. For this reason, you must follow the RFC as closely as possible. Avoid certificate extensions unless you know that all of your targeted mobile platforms can handle them as you expect.

## CRL support

If your certificate supports certificate revocation lists (CRLs), ensure that the CRL URL is valid and accessible. Otherwise, certificate chain validation fails.

## Tools to use to verify the server certificate

To debug certificate path validation problems, try the `openssl s_client` command line tool. This tool generates good diagnostic information that is helpful in debugging SSL issues.

The following example shows how to use the `openssl s_client` command line tool:

```
openssl s_client -CApath $HOME/CAdir -connect hostname:port
```

The following example shows how to inspect a certificate:

```
openssl x509 -in certificate.crt -text -noout
```

## Troubleshooting problems with server certificates that are not signed by a trusted certificate authority

Table 29. Troubleshoot problems with server certificates

Problem	Actions to take
Unable to install the root CA on iOS. Certificate installs, but after installation, iOS shows the certificate as not trusted.	The certificate is not identified as a certificate authority. Ensure that the certificate specifies a certificate extension: <code>basicConstraints = CA:TRUE</code>  For more information, see “Self-signed certificates versus self-signed CAs” on page 107.  Ensure that the certificate is in PEM format.  Ensure that the certificate has a <code>.crt</code> file extension.

Table 29. Troubleshoot problems with server certificates (continued)

Problem	Actions to take
<p>Unable to install the root CA on Android.</p> <p>After installation, the certificate does not show up in the system's trusted credentials.</p>	<p>The certificate is an X.509 version 1 certificate or does not have the following certificate extension:</p> <p>basicConstraints = CA:TRUE</p> <p>For more information, see "Self-signed certificates versus self-signed CAs" on page 107.</p> <p>Ensure that the certificate is in PEM or DER format.</p> <p>Ensure that the certificate has a .crt file extension.</p>
<p>"errorCode": "UNRESPONSIVE_HOST", "errorMsg": "The service is currently not available."</p>	<p>This error usually indicates an SSL handshake failure.</p> <p>The client cannot establish trust for the server certificate.</p> <ol style="list-style-type: none"> <li>1. Ensure that you installed the server's root CA on the client device. For more information, see "Establishing trust on the client" on page 108.</li> <li>2. Ensure that the server sends the complete certificate chain and in the right order. For more information, see "Handling the certificate chain" on page 108.</li> </ol> <p>The server certificate is invalid.</p> <ol style="list-style-type: none"> <li>1. Check the validity of the server certificate. For more information, see "Tools to use to verify the server certificate" on page 109.</li> <li>2. Ensure that the CRL URL is valid and reachable. For more information, see "CRL support" on page 109.</li> <li>3. The server certificate contains a critical certificate extension that is not recognized by the client platform. For more information, see "Handling certificate extensions" on page 109.</li> </ol>



Table 29. Troubleshoot problems with server certificates (continued)





Problem	Actions to take
<p>SSL works on Android, but does not work on iOS.</p>	<p>When Android is in debuggable mode, Cordova ignores most SSL errors. This behavior gives the impression that things are working. Android is in debuggable mode when the APK is unsigned, or when you explicitly set the mode in the manifest. IBM Worklight sets the mode to debuggable in the manifest by default. To make this behavior consistent, set the Android application to <code>debuggable:false</code> in the manifest, or sign the APK. Make sure that there is no explicit declaration in the manifest that sets it to debuggable mode. For more information about how trust server certificates from a private CA, see “Configuring SSL between IBM Worklight Servers and clients by using certificates that are not signed by a trusted certificate authority” on page 106.</p>
<p>After installation, the certificate does not show up in the system’s trusted credentials or truststore.</p>	<p>Ensure that you did not install the server certificate by accessing the protected resource directly from your browser. This action imports the certificate only into the browser space and not into the device system truststore. The only requirement is that you install the root CA.</p> <p>For more information about how to properly install the root CA on the device, see the following topics.</p> <p>For iOS, see “Installing the root CA on iOS.”</p> <p>For Android, see “Installing the root CA on Android” on page 114.</p>

**Related tasks:**

“Configuring SSL for Liberty profile” on page 144

Create a key store, import the Secure Socket Layer (SSL) certificate, and edit the `server.xml` file to configure SSL on Liberty profile.

**Related information:**

-  Security with HTTPS and SSL
-  HTTPS Server Trust Evaluation
-  The Transport Layer Security (TLS) Protocol Version 1.2
-  RFC 5280 - Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile

**Installing the root CA on iOS**

The root CA must be installed on the client device to ensure that the client trusts server certificates that are signed by your private CAs.

## About this task

To establish trust for your server certificate, you must install the trust anchor certificate (root CA) on the client device.

**Note:** Only the root CA certificate (trust anchor) must be installed. You do not need to install any other certificates, such as intermediaries, on the device.

## Procedure

1. Ensure that the root CA is in PEM file format and has a .crt file extension. Convert as needed.
2. Run the following command to view the certificate details.  

```
openssl x509 -in certificate.crt -text -noout
```
3. Ensure that the certificate is of version X.509 v3. The certificate details must show Version 3.

**Note:** The following openssl flag generates X.509 v3 certificates:

```
-reqexts v3_req
```

4. Ensure that the certificate is a certificate authority. The certificate details must show X509v3 Basic Constraints: CA:TRUE

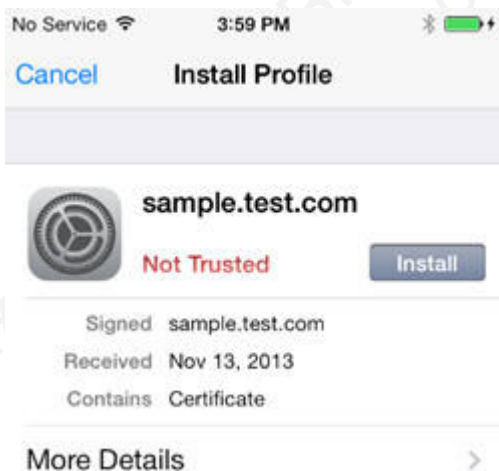
**Note:** The following openssl flag generates the CA extension:

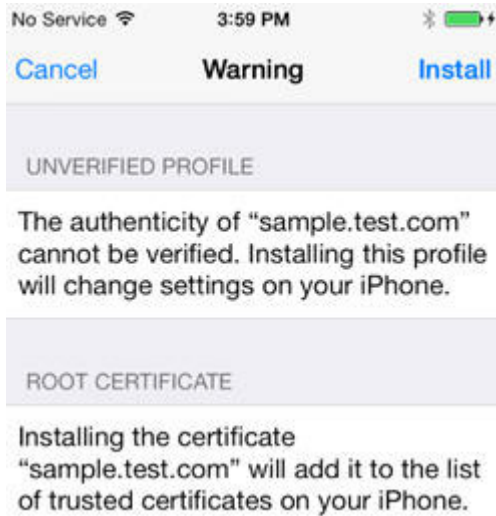
```
-extensions v3_ca
```

5. To download the certificate file on the device, send it as an email attachment or host it on a secure website.

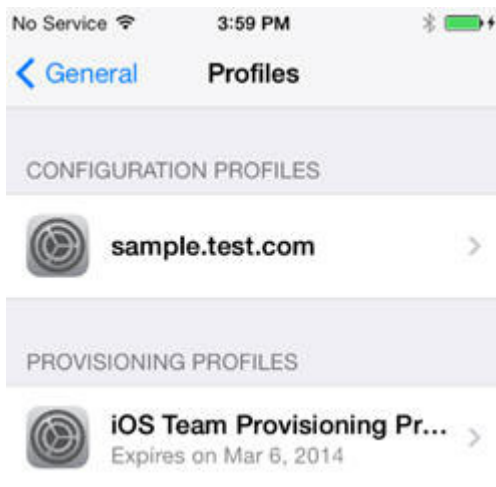
**Note:** Do not install the server certificate by accessing the protected resource directly from your browser. This action imports the certificate only into the browser space and not into the device system truststore.

6. After you have the certificate file on the device, click the file to allow the iOS system to install the certificate.

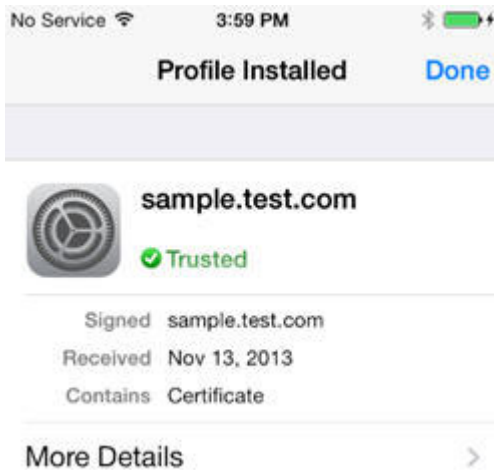




7. Check that the certificate was properly installed under **Settings > General > Profiles > Configuration Profiles**.



8. Ensure that the iOS device lists the CA as a trusted certificate authority.



## Installing the root CA on Android

The root CA must be installed on the client device to ensure that the client trusts server certificates that are signed by your private CAs.

### About this task

To establish trust for your server certificate, you must install the trust anchor certificate (root CA) on the client device.

**Note:** Only the root CA certificate (trust anchor) must be installed. You do not need to install any other certificates, such as intermediaries, on the device.

### Procedure

1. Ensure that the root CA is in PEM or DER file format and has a .crt file extension. Convert as needed.
2. Run the following command to view the certificate details.  

```
openssl x509 -in certificate.crt -text -noout
```
3. Ensure that the certificate is of version X.509 v3. The certificate details must show Version 3.

**Note:** The following openssl flag generates X.509 v3 certificates:

```
-reqexts v3_req
```

4. Ensure that the certificate is a certificate authority. The certificate details must show X509v3 Basic Constraints: CA:TRUE

**Note:** The following openssl flag generates the CA extension:

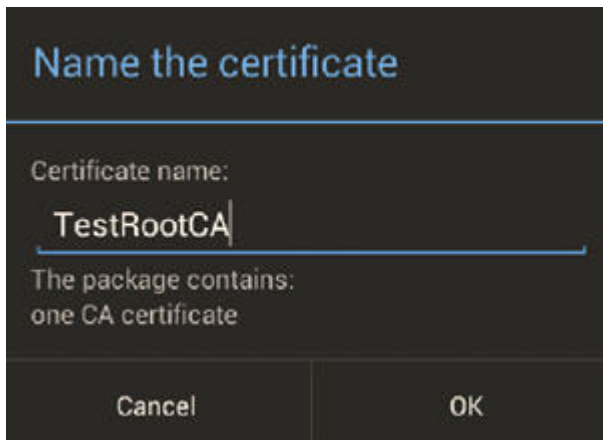
```
-extensions v3_ca
```

5. To download the certificate file on the device, send it as an email attachment or host it on a secure website.

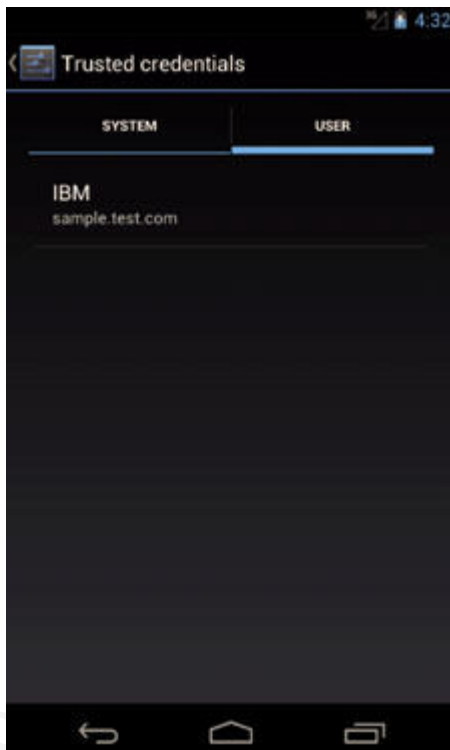
**Note:** Do not install the server certificate by accessing the protected resource directly from your browser. This action imports the certificate only into the browser space and not into the device system truststore.

6. After you have the file on the device, click the file to allow the Android system to install the certificate.

7. Provide an alias name for the certificate when you are prompted.



8. Check that the certificate was properly installed under **Settings > Security > Trusted Credentials > User**.



### Updating your keystore and Liberty profile configuration to use a certificate chain

You must ensure that your server sends the whole certificate chain to client devices on an SSL handshake.

## About this task

For the client to validate the certificate path, it must have access to the full certificate chain. To ensure that the client has access to the full certificate chain (including intermediate certificates), ensure that all the certificates in the chain are in the server-side keystore file.

Assuming that you have a root CA certificate, intermediate certificates, and a server certificate, the whole chain must be sent on the HTTPS connection. These certificates must be concatenated in one file, by concatenating in the following order: server certificate, intermediate CA certificates (if any exist, and if so, in the order in which they were signed), and finally the root CA.

The following example assumes that you have a server certificate (SERVER\_IDENTITY\_CERT\_NAME), one intermediate CA certificate (INTERMEDIATE\_CA\_CERT\_NAME), and a root CA (ROOT\_CA\_CERT\_NAME).

## Procedure

1. Open a terminal and navigate to a temporary working directory.

2. Concatenate your certificates to form the certificate chain.

- a. Concatenate the intermediate and the root CA certificates.

```
cat INTERMEDIATE_CA_CERT_NAME ROOT_CA_CERT_NAME > INTERMEDIATE_CA_CHAIN_CERT_NAME
```

- b. Add the server certificate to the chain.

```
cat .SERVER_IDENTITY_CERT_NAME SIGNING_CA_CHAIN_CERT_NAME > server_chain.crt
```

3. Export the private key and certificate chain into a .p12 keystore.

```
openssl pkcs12 -export -in server_chain.crt -inkey server/server_key.pem -out server/server.p12 -passout pass:passServerP12 -passin pass:passServer
```

4. Update your Liberty profile server.xml file.

- a. Enable the SSL feature.

```
<featureManager>
...
  <feature>ssl-1.0</feature>
...
</featureManager>
```

- b. Create an SSL configuration.

```
<ssl id="mySSLSettings" keyStoreRef="myKeyStore" />
<keyStore id="myKeyStore"
  location="server/server.p12"
  type="PKCS12"
  password="passServer12" />
```

- c. Configure your HTTP endpoint to use this SSL configuration or set the configuration as the default.

```
<sslDefault sslRef="mySSLSettings" />
```

## What to do next

For more information, see [Enabling SSL communication for the Liberty profile](#).

## Handling MySQL stale connections

Instructions for how to configure your application server to avoid MySQL timeout issues.

The MySQL database closes its connections after a period of non-activity on a connection. This timeout is defined by the system variable called `wait_timeout`. The default is 28000 seconds (8 hours).



When an application tries to connect to the database after MySQL closes the connection, the following exception is generated:

```
com.mysql.jdbc.exceptions.jdbc4.MySQLNonTransientConnectionException: No operations allowed after
```

The following sections provide the configuration elements specific to each application server you can use to avoid this exception if you use the MySQL database.

## Apache Tomcat configuration

Edit the `server.xml` and `context.xml` files, and for every `<Resource>` element add the following properties:

- `testOnBorrow="true"`
- `validationQuery="select 1"`

For example:

```
<Resource name="jdbc/AppCenterDS"
  type="javax.sql.DataSource"
  driverClassName="com.mysql.jdbc.Driver"
  ...
  testOnBorrow="true"
  validationQuery="select 1"
/>
```

## WebSphere Application Server Liberty Profile configuration

**Note:** MySQL in combination with WebSphere Application Server Liberty Profile or WebSphere Application Server Full Profile is not classified as a supported configuration. For more information, see WebSphere Application Server Support Statement. We suggest that you use IBM DB2 or another database supported by WebSphere Application Server to benefit from a configuration that is fully supported by IBM Support.

Edit the `server.xml` file and for every `<dataSource>` element (Worklight and Application Center databases) add a `<connectionManager>` element with the `agedTimeout` property:

```
<connectionManager agedTimeout="timeout"/>
```

For example:

```
<dataSource jndiName="jdbc/AppCenterDS" transactional="false">
  <connectionManager agedTimeout="7h30m"/>
  <jdbcDriver libraryRef="MySQLLib"/>
  ...
</dataSource>
```

## WebSphere Application Server Full Profile configuration

**Note:** MySQL in combination with WebSphere Application Server Liberty Profile or WebSphere Application Server Full Profile is not classified as a supported configuration. For more information, see WebSphere Application Server Support Statement. We suggest that you use IBM DB2 or another database supported by WebSphere Application Server to benefit from a configuration that is fully supported by IBM Support.

1. Log in to the WebSphere Application Server console.
2. Select **Resources > JDBC > Data sources**.
3. For each MySQL data source:

- a. Click the data source.
- b. Select **Connection pool properties** under **Additional Properties**.
- c. Modify the value of the **Aged timeout** property. The value must be lower than the MySQL `wait_timeout` system variable to have the connections purged prior to the time that MySQL closes these connections.
- d. Click **OK**.

---

## Installing the Application Center

You install IBM Worklight Application Center as part of the Worklight Server installation.

The Application Center is part of Worklight Server. To install the Application Center, see “Installing Worklight Server” on page 41.

When you install an IBM Worklight edition through IBM Installation Manager, the Application Center and Worklight Console are installed in the web application server that you designate. You have minimal additional configuration to do. See “Configuring the Application Center after installation.”

If you chose a manual setup in the installer, see the documentation of the server of your choice.

For a list of installed files and tools, see “Distribution structure of Worklight Server” on page 62.

---

## Configuring 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.

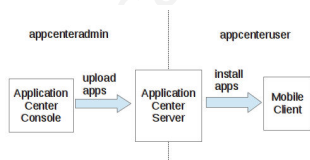


Figure 3. JEE security roles of the Application Center and the components that they influence

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 121 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 772 for how to build the Application Center mobile client.

**Related concepts:**

“Managing users with LDAP” on page 121  
 Use the Lightweight Directory Access Protocol (LDAP) registry to manage users.

**Related reference:**

“Preparations for using the mobile client” on page 772  
 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.

## 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**.

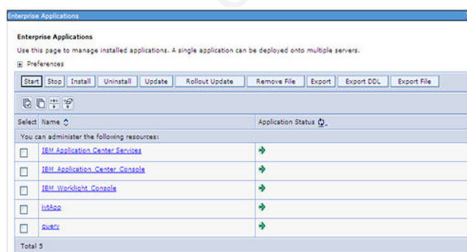


Figure 4. Mapping the Application Center roles

- d. Select **IBM\_Application\_Center\_Services**.

- e. In the **Configuration** tab, select **Details > Security role to user/group mapping**.

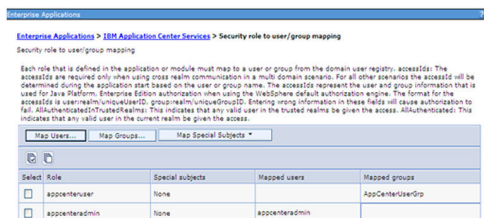


Figure 5. Mapping the **appcenteruser** and **appcenteradmin** roles: user groups

- 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 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.

### 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 **appcenteradminingroup**. 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.

### Procedure

1. Edit the `server.xml` file.

For example:

```
<security-role name="appcenteradmin">
  <group name="appcenteradminingroup"/>
</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 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.

## Managing users with LDAP

Use the Lightweight Directory Access Protocol (LDAP) registry to manage users.

LDAP is a way to centralize the user management for multiple web applications in an LDAP Server that maintains a user registry. It can be used instead of specifying one by one the users for the security roles **appcenteradmin** and **appcenteruser**.

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).

Since IBM Worklight V6.0, use the JNDI environment entries for defining LDAP configuration properties.

Expert users could configure the application servers to use LDAP authentication by using the methods that were documented in releases before IBM Worklight V6.0.

## LDAP with WebSphere Application Server V7

Use LDAP to authenticate users and define the users and groups who can install mobile applications with the Application Center; you can use the JNDI environment or the VMM API to define the LDAP mapping

You use LDAP to define the roles **appcenteradmin** and **appcenteruser**. Then, you have two ways of defining LDAP mapping for WebSphere Application Server V7:

- By using the JNDI environment with a stand-alone LDAP configuration
- By using federated repositories with the Virtual Member Manager (VMM) API

### Configuring LDAP authentication (WebSphere Application Server V7):

Define the users who can access the Application Center console and the users who can log in to the client by configuring LDAP as a stand-alone LDAP server or as a federated repository.

#### About this task

This procedure shows you how to use LDAP to define the roles **appcenteradmin** and **appcenteruser** in WebSphere Application Server V7.

#### Procedure

1. Log in to the WebSphere Application Server console.
2. In **Security > Global Security**, verify that administrative security and application security are enabled.
3. Select **Federated repositories** or **Standalone LDAP registry**.
4. Click **Configure**. For federated repositories, follow step 5. For stand-alone LDAP registry, follow step 6
5. **Option for federated repositories:** add the new repository and configure the required additional properties.
  - a. To add a new repository, click **Add Base entry to Realm**.
  - b. Specify the value of "Distinguished name of a base entry that uniquely identifies entries in the realm" and click **Add Repository**.
  - c. Select **LDAP Repository**.
  - d. Give this repository a name and enter the values required to connect to your LDAP server.
  - e. Under **Additional Properties**, click **LDAP entity types**.
  - f. Configure the **Group**, **OrgContainer**, and **PersonAccount** properties. These configuration details depend on your LDAP server.
6. **Option for stand-alone LDAP registry:** Configure access control (ACL) management. You can use JNDI properties for this configuration, but you cannot use VMM.
  - a. Enter the values of **General Properties**. These values depend on your LDAP server.
  - b. Under **Additional Properties**, click **Advanced Lightweight Directory Access Protocol (LDAP)** and configure the user and group filters and maps. These configuration details depend on your LDAP server.
7. Save the configuration, log out, and restart the server.
8. In the WebSphere Application Server console, map the security roles to users and groups.



- a. In the **Configuration** tab, select **Applications > WebSphere Enterprise applications**.
  - b. Select "IBM\_Application\_Center\_Services".
  - c. In the **Configuration** tab, select **Details > Security role to user/group mapping**.
  - d. For **appcenteradmin** and **appcenteruser** roles, select **Map groups**. This selection enables you to select users and groups inside the WebSphere user repository, including LDAP users and groups. The selected users can access the Application Center as **appcenteradmin** or **appcenteruser**. You can also map the roles to **Special Subjects** "All authenticated in application realm" to give everyone in the WebSphere user repository, including everyone registered in the LDAP registry, access to the Application Center.
9. Repeat the procedure described in step 8 on page 122 for **IBM\_Application\_Center\_Console**. (Make sure that you select "IBM\_Application\_Center\_Console" in step 8.b instead of "IBM\_Application\_Center\_Services".)
  10. Click **Save** to save your changes.

#### **Configuring LDAP ACL management with JNDI (WebSphere Application Server V7):**

Use LDAP to define the users and groups who can install mobile applications with the Application Center by using the JNDI environment.

#### **About this task**

Since IBM Worklight V6.0, two configuration approaches are available: the JNDI API or the Virtual Member Manager (VMM) API. This procedure shows you how to use the JNDI API to configure LDAP based on the federated repository configuration or with the stand-alone LDAP registry. Only the simple type of LDAP authentication is supported.

#### **Procedure**

1. Log in to the WebSphere Application Server console.
2. Select **Applications > Application Types > WebSphere enterprise applications**.
3. Click **IBM\_Application\_Center\_Services**.
4. In the **Web Module Properties** section, select "Environment entries for Web modules".
  - a. For the **ibm.appcenter.ldap.vmm.active** entry, assign the value "false".
  - b. For the **ibm.appcenter.ldap.active** entry, assign the value "true".
5. Continue to configure the remaining entries:
  - **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.
  - `ibm.appcenter.ldap.group.nesting`: management of nested groups. If nested groups are not managed, set the value to `false`.
    - a. Enter the value of each property.
    - b. Click **OK** and save the configuration.
6. **Option:** *If security binding is required, follow this step.* Configure the following entries:
- `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. Enter the value of each optional property and click **OK**. Set the value of the `ibm.appcenter.ldap.security.bindpwd` property to the encoded password generated by the "WebSphere PropFilePasswordEncoder" utility.
    - b. Save the configuration.
7. **Option:** *If users and groups are defined in the same subtree (the properties `ibm.appcenter.ldap.user.base` and `ibm.appcenter.ldap.group.base` have the same value), follow this step.* Configure the following entries:
- `ibm.appcenter.ldap.user.filter`: LDAP user search filter for the attribute of user login name. Use `%v` as the placeholder for the login name attribute.
  - `ibm.appcenter.ldap.group.filter`: LDAP group search filter. Use `%v` as the placeholder for the group attribute.
  - `ibm.appcenter.ldap.user.displayName.filter`: LDAP user search filter for the attribute of user display name. Use `%v` as the placeholder for the user display name attribute.
    - a. Enter the value of each optional property and click **OK**.
    - b. Save the configuration.

## Results

The following figure shows the values to assign to each property.

The value of Environment entry defined in the deployment descriptor can be edited.

Web module	URI	Name	Type	Description	Value
ApplicationCenterServices	ApplicationCenterServices-6.0.0.jar-SLSAPPNOT.na.WEB-INF/web.xml	ibm.appcenter.proxy.host	String		
ApplicationCenterServices	ApplicationCenterServices-6.0.0.jar-SLSAPPNOT.na.WEB-INF/web.xml	ibm.appcenter.proxy.port	Integer		
ApplicationCenterServices	ApplicationCenterServices-6.0.0.jar-SLSAPPNOT.na.WEB-INF/web.xml	ibm.appcenter.ldap.active	String		true
ApplicationCenterServices	ApplicationCenterServices-6.0.0.jar-SLSAPPNOT.na.WEB-INF/web.xml	ibm.appcenter.ldap.vmm.active	String		false
ApplicationCenterServices	ApplicationCenterServices-6.0.0.jar-SLSAPPNOT.na.WEB-INF/web.xml	ibm.appcenter.ldap.connectionURL	String		jdbc:dspace.ibm.com:389
ApplicationCenterServices	ApplicationCenterServices-6.0.0.jar-SLSAPPNOT.na.WEB-INF/web.xml	ibm.appcenter.ldap.user.base	String		ou=bluepages.o=ibm.co
ApplicationCenterServices	ApplicationCenterServices-6.0.0.jar-SLSAPPNOT.na.WEB-INF/web.xml	ibm.appcenter.ldap.user.loginName	String		emailAddress
ApplicationCenterServices	ApplicationCenterServices-6.0.0.jar-SLSAPPNOT.na.WEB-INF/web.xml	ibm.appcenter.ldap.user.displayName	String		cn
ApplicationCenterServices	ApplicationCenterServices-6.0.0.jar-SLSAPPNOT.na.WEB-INF/web.xml	ibm.appcenter.ldap.user.groupmembership	String		ibm-allGroups
ApplicationCenterServices	ApplicationCenterServices-6.0.0.jar-SLSAPPNOT.na.WEB-INF/web.xml	ibm.appcenter.ldap.group.base	String		ou=memberlat.o=ibm
ApplicationCenterServices	ApplicationCenterServices-6.0.0.jar-SLSAPPNOT.na.WEB-INF/web.xml	ibm.appcenter.ldap.group.name	String		cn
ApplicationCenterServices	ApplicationCenterServices-6.0.0.jar-SLSAPPNOT.na.WEB-INF/web.xml	ibm.appcenter.ldap.group.uniqueMember	String		uniqueMember
ApplicationCenterServices	ApplicationCenterServices-6.0.0.jar-SLSAPPNOT.na.WEB-INF/web.xml	ibm.appcenter.ldap.group.nesting	String		
ApplicationCenterServices	ApplicationCenterServices-6.0.0.jar-SLSAPPNOT.na.WEB-INF/web.xml	ibm.appcenter.ldap.security.binddn	String		
ApplicationCenterServices	ApplicationCenterServices-6.0.0.jar-SLSAPPNOT.na.WEB-INF/web.xml	ibm.appcenter.ldap.security.bindpwd	String		
ApplicationCenterServices	ApplicationCenterServices-6.0.0.jar-SLSAPPNOT.na.WEB-INF/web.xml	android.aapt	String		
ApplicationCenterServices	ApplicationCenterServices-6.0.0.jar-SLSAPPNOT.na.WEB-INF/web.xml	android.aapt.dir	String		

Figure 6. Environment entries and their values (LDAP and WebSphere Application Server V7)

## Configuring LDAP ACL management with VMM (WebSphere Application Server V7):

Use LDAP to define the users and groups who can install mobile applications with the Application Center with the Virtual Member Manager (VMM) API.

### About this task

Since IBM Worklight V6.0, two configuration approaches are available: the JNDI API or the VMM API. This procedure shows you how to use the VMM API to configure LDAP based on the federated repository configuration.

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.

### Procedure

1. Configure the attribute mapping. For users, the Application Center refers to these VMM attributes:

- **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 to LDAP attributes, you must map the VMM attributes to the corresponding LDAP attributes.

In WebSphere Application Server V7, you cannot configure this mapping with the WebSphere Application Server console.

- a. Find in the file `{WAS_HOME}/profiles/{profileName}/config/cells/{cellName}/wim/config/wimconfig.xml` the section that contains the LDAP repository configuration with `id="your LDAP id"`:

```
<config:repositories xsi:type="config:LdapRepositoryType" adapterClassName="com.ibm.ws.wim.  
id="your LDAP id"....
```

Where your LDAP id is the user ID configured for you in the LDAP repository.

- b. In this section, after the element `<config:attributeConfiguration>`, add these entries:

```
<config:attributes name="your LDAP attribute for the user full name" propertyName="sn">  
  <config:entityTypes>PersonAccount</config:entityTypes>  
</config:attributes>  
<config:attributes name="your LDAP attribute for the user login name " property  
  <config:entityTypes>PersonAccount</config:entityTypes>  
</config:attributes>
```

- c. Save the file and restart the server.

2. Configure the Application Center for ACL management with LDAP. In WebSphere Application Server V7, only a WebSphere administrator user can run VMM access. (VMM roles are only supported by WebSphere Application Server V8.)

You must define these properties:

- `ibm.appcenter.ldap.active = true.`
- `ibm.appcenter.ldap.vmm.active = true.`
- `ibm.appcenter.ldap.vmm.adminuser = WebSphere administrator user.`

- `ibm.appcenter.ldap.vmm.adminpwd` = *WebSphere administrator password*. The password can be encoded or not.
  - a. Log in to the WebSphere Application Server console.
  - b. Select **Applications > Application Types > WebSphere enterprise applications**.
  - c. In the "Web Module Properties" section, select **IBM\_Application\_Center\_Services** and then select **Environment entries for Web modules**.
  - d. Set the values for the properties.
  - e. Click **OK** and save the configuration. The application is automatically restarted.
3. **Optional:** Encode the password with the **PropFilePasswordEncoder** utility.
- a. Create a file `pwd.txt` that contains the entry `adminpwd=your clear password`, where your clear password is the unencoded administrator password.
  - b. Run this command:
 

```
{WAS_HOME}/profiles/profile name/bin/PropFilePasswordEncoder "file path/ pwd.txt" adminpwd
```
  - c. Open the `pwd.txt` file and copy the encoded password into the value of the **`ibm.appcenter.ldap.vmm.adminpwd`** property.

### LDAP with 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.

### 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 (WebSphere Application Server V8.x):

Use LDAP to define users who can access the Application Center console and users who can log in to the client.

## About this task

You can configure LDAP based on the federated repository configuration only. This procedure shows you how to use LDAP to define the roles **appcenteradmin** and **appcenteruser** in WebSphere Application Server V8.x.

## Procedure

1. Log in to the WebSphere Application Server console.
2. Select **Security > Global security** and verify that administrative security and application security are enabled.
3. In the "User account repository" section, select **Federated repositories**.
4. Click **Configure**.
5. Add a new repository and configure the required repository.
  - a. Click **Add Base entry to Realm**.
  - b. Specify the value of "Distinguished name of a base entry that uniquely identifies entries in the realm" and click **Add Repository**.
  - c. Select **LDAP Repository**.
  - d. Give this repository a name and enter the values required to connect to your LDAP server.
  - e. Under **Additional Properties**, click **LDAP entity types**.
  - f. Configure the **Group**, **OrgContainer**, and **PersonAccount** properties. These configuration details depend on your LDAP server.
6. Save the configuration, log out, and restart the server.
7. In the WebSphere Application Server console, map the security roles to users and groups.
  - a. In the **Configuration** tab, select **Applications > WebSphere Enterprise applications**.
  - b. Select "IBM\_Application\_Center\_Services".
  - c. In the **Configuration** tab, select **Details > Security role to user/group mapping**.
  - d. For **appcenteradmin** and **appcenteruser** roles, select **Map groups**. This selection enables you to select users and groups inside the WebSphere user repository, including LDAP users and groups. The selected users can access the Application Center as **appcenteradmin** or **appcenteruser**. You can also map the roles to **Special Subjects** "All authenticated in application realm" to give everyone in the WebSphere user repository, including everyone registered in the LDAP registry, access to the Application Center.
8. Repeat the procedure described in step 7 for **IBM\_Application\_Center\_Console**. (Make sure that you select "IBM\_Application\_Center\_Console" in step 7.b instead of "IBM\_Application\_Center\_Services".)
9. Click **Save** to save your changes.

## What to do next

You must enable ACL management with LDAP. See "Configuring LDAP ACL management (WebSphere Application Server V8.x)."

## Configuring LDAP ACL management (WebSphere Application Server V8.x):

Use LDAP to define the users and groups who can install mobile applications with the Application Center with the Virtual Member Manager (VMM) API.

## About this task

To configure ACL with LDAP, you should define three properties: **uid**, **sn**, and **cn**. These properties enable the login name and the full name of users and the name of user groups to be identified in the Application Center.

Then you should enable ACL management with VMM. 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. In the "User account repository" section, select **Configure**.
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**.

The screenshot shows the 'General Properties' panel for the 'mail' property. The 'Name' field contains 'mail'. The 'Property name' field contains 'uid'. The 'Syntax' field is empty. The 'Entity types' field contains 'PersonAccount'. The 'Default value' field is empty. The 'Default attribute' field is empty. At the bottom, there are buttons for 'Apply', 'OK', 'Reset', and 'Cancel'.

Figure 7. 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.
  - b. For **Property name** enter **sn**.
  - c. For **Entity types** enter the LDAP entity type.
  - d. Click **OK**.

The screenshot shows the 'General Properties' panel for the 'cn' property. The 'Name' field contains 'cn'. The 'Property name' field contains 'sn'. The 'Syntax' field is empty. The 'Entity types' field contains 'PersonAccount'. The 'Default value' field is empty. The 'Default attribute' field is empty. At the bottom, there are buttons for 'Apply', 'OK', 'Reset', and 'Cancel'.

Figure 8. 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**.
10. Enable ACL management with LDAP:
- a. Select **Servers > Server Types > WebSphere application servers**.
  - b. Select the appropriate application server.  
In a clustered environment you must configure all the servers in the cluster in the same way.
  - c. In the **Configuration** tab, under "Server Infrastructure", click the **Java and Process Management** tab and select **Process definition**.
  - d. In the **Configuration** tab, under "Additional Properties", select **Java Virtual Machine**,
  - e. In the **Configuration** tab, under "Additional Properties", select **Custom properties**.
  - f. Click **New**.
  - g. In the form, enter `ibm.appcenter.ldap.vmm.active` and assign the value "true".
  - h. Click **Apply**.
  - i. In the form, enter `ibm.appcenter.ldap.active` and assign the value "true".
  - j. Click **OK**.

## Results

The following figure shows an example of custom properties with the correct settings.

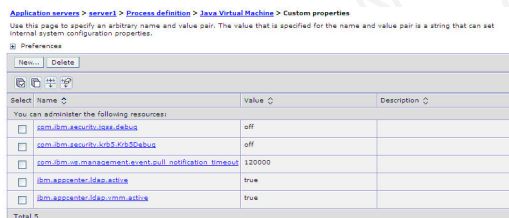


Figure 9. ACL management for Application Center with LDAP on WebSphere Application Server V8

## 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**.

## LDAP with Liberty Profile

Use LDAP to authenticate users and to define the users and groups who can install mobile applications with the Application Center by using the JNDI environment.

Using LDAP with Liberty Profile requires you to configure LDAP authentication and LDAP ACL management.

### Configuring LDAP authentication (Liberty Profile):

You configure LDAP authentication by defining an LDAP registry in the `server.xml` file and map LDAP users and groups to Application Center roles.

#### About this task

You can configure LDAP authentication of users and groups in the `server.xml` file by defining an LDAP registry. Then you map users and groups to Application Center roles. 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 an LDAP registry definition after the `<httpEndpoint>` element.

Example for the LDAP registry:

```
<ldapRegistry baseDN="o=ibm.com" host="employees.com" id="Employees"
ldapType="IBM Tivoli Directory Server" port="389" realm="AppCenterLdap"
recursiveSearch="true">
  <idsFilters
    groupFilter="(& (cn=%v) (|(objectclass=groupOfNames)(objectclass=groupOfUniqueNames))) " id="
    userFilter="(& (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.

3. Insert a security role definition after each Application Center application definition (**applicationcenter** and **appcenterconsole**).

Example for security role definition: 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.

#### 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.

### Configuring LDAP ACL management (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. Only the simple type of LDAP authentication is supported.

#### Properties

To be able to define JNDI entries, the following feature must be defined in the `server.xml` file:

```
<feature>jndi-1.0</feature>
```

Add an entry for each property in the `<server>` section of the `server.xml` file. This entry should have the following syntax:

```
<jndiEntry jndiName="JNDI_property_name" value="property_value"/>
```

Where:

`JNDI_property_name` is the name of the property you are adding.

`property_value` is the value of the property you are adding.

Table 30. JNDI properties for configuring ACL management with LDAP in the server.xml file

Property	Description
<b>ibm.appcenter.ldap.active</b>	Set to true to enable LDAP; set to false to disable LDAP.
<b>ibm.appcenter.ldap.connectionURL</b>	LDAP connection URL.
<b>ibm.appcenter.ldap.user.base</b>	Search base of users.
<b>ibm.appcenter.ldap.user.loginName</b>	LDAP login attribute.
<b>ibm.appcenter.ldap.user.displayName</b>	LDAP attribute for the user name to be displayed, for example, a person's full name.
<b>ibm.appcenter.ldap.group.base</b>	Search base of groups.
<b>ibm.appcenter.ldap.group.name</b>	LDAP attribute for the group name.
<b>ibm.appcenter.ldap.group.uniquemember</b>	LDAP attribute that identifies the members of a group.
<b>ibm.appcenter.ldap.user.groupmembership</b>	LDAP attribute that identifies the groups to which a user belongs.
<b>ibm.appcenter.ldap.group.nesting</b>	Management of nested groups: if nested groups are not managed, set the value to false.
<b>ibm.appcenter.ldap.user.filter</b>	<p>LDAP user search filter for the attribute of user login name. Use %v as the placeholder for the login name attribute.</p> <p>This property is only required when LDAP users and groups are defined in the same subtree; that is, when the properties <b>ibm.appcenter.ldap.user.base</b> and <b>ibm.appcenter.ldap.group.base</b> have the same value.</p>
<b>ibm.appcenter.ldap.displayName.filter</b>	<p>LDAP user search filter for the attribute of user display name. Use %v as the placeholder for the display name attribute.</p> <p>This property is only required when LDAP users and groups are defined in the same subtree; that is, when the properties <b>ibm.appcenter.ldap.user.base</b> and <b>ibm.appcenter.ldap.group.base</b> have the same value.</p>
<b>ibm.appcenter.ldap.group.filter</b>	<p>LDAP group search filter. Use %v as the placeholder for the group attribute.</p> <p>This property is only required when LDAP users and groups are defined in the same subtree; that is, when the properties <b>ibm.appcenter.ldap.user.base</b> and <b>ibm.appcenter.ldap.group.base</b> have the same value.</p>
<b>ibm.appcenter.ldap.security.binddn</b>	Property that identifies the distinguished name of the user permitted to search the LDAP directory. Use this property only if security binding is required.

Table 30. JNDI properties for configuring ACL management with LDAP in the server.xml file (continued)

Property	Description
<b>ibm.appcenter.ldap.security.bindpwd</b>	<p>Property that identifies the password of the user permitted to search the LDAP directory. Use this property only if security binding is required.</p> <p>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.</p> <p>Edit the Liberty Profile server.xml file to check whether the <i>classloader</i> is enabled to load the JAR file that decodes the password.</p>

### Example of setting properties for ACL management with LDAP

This example shows the settings of the properties in the server.xml file required for ACL management with LDAP.

```
<jndiEntry jndiName="ibm.appcenter.ldap.active" value="true"/>
<jndiEntry jndiName="ibm.appcenter.ldap.connectionURL" value="ldap://employees.com:389"/>
<jndiEntry jndiName="ibm.appcenter.ldap.user.base" value="ou=employees,o=ibm.com"/>
<jndiEntry jndiName="ibm.appcenter.ldap.user.loginName" value="emailAdress"/>
<jndiEntry jndiName="ibm.appcenter.ldap.user.displayName" value="cn"/>
<jndiEntry jndiName="ibm.appcenter.ldap.group.base" value="ou=groups,o=ibm.com"/>
<jndiEntry jndiName="ibm.appcenter.ldap.group.name" value="cn"/>
<jndiEntry jndiName="ibm.appcenter.ldap.group.uniquemember" value="uniqueMember"/>
<jndiEntry jndiName="ibm.appcenter.ldap.user.groupmembership" value="ibm-allGroups"/>
```

### LDAP with 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.

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. Only the simple type of LDAP authentication is supported.

#### Configuring LDAP authentication (Apache Tomcat):

Define the users who can access the Application Center Console and the users who can log in with the mobile client by mapping Java Platform, Enterprise Edition roles to LDAP roles.

#### 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. Only the simple type of LDAP authentication is supported.

You configure the Apache Tomcat server for LDAP authentication and configure security (Java™ Platform, Enterprise Edition) in the `web.xml` file of the Application Center Services web application (`applicationcenter.war`) and of the Application Center Console web application (`appcenterconsole.war`).

### 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.

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.

#### 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>
        <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>
    </web-resource-collection>
    <role-refinement>
        <role-name>appcenteradmin</role-name>
        <role-name>appcenteruser</role-name>
    </role-refinement>
</security-constraint>

```

```

<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>

```

### Configuring LDAP ACL management (Apache Tomcat):

Use LDAP to define the users and groups who can install mobile applications with the Application Center by defining the Application Center LDAP properties through JNDI.

#### Purpose

To configure LDAP ACL management of the Application Center; add an entry for each property in the <context> section of the IBM Application Center Services application in the server.xml file. This entry should have the following syntax:

```
<Environment name="JNDI_property_name" value="property_value" type="java.lang.String" override="fa
```

Where:

JNDI\_property\_name is the name of the property you are adding.

property\_value is the value of the property you are adding.

Table 31. Properties for configuring ACL management for LDAP in the server.xml file on Apache Tomcat

Property	Description
<b>ibm.appcenter.ldap.active</b>	Set to true to enable LDAP; set to false to disable LDAP.
<b>ibm.appcenter.ldap.connectionURL</b>	LDAP connection URL.
<b>ibm.appcenter.ldap.user.base</b>	Search base of users.



Table 31. Properties for configuring ACL management for LDAP in the server.xml file on Apache Tomcat (continued)

Property	Description
<b>ibm.appcenter.ldap.user.loginName</b>	LDAP login attribute.
<b>ibm.appcenter.ldap.user.displayName</b>	LDAP attribute for the user name to be displayed, for example, a person's full name.
<b>ibm.appcenter.ldap.group.base</b>	Search base of groups.
<b>ibm.appcenter.ldap.group.name</b>	LDAP attribute for the group name.
<b>ibm.appcenter.ldap.group.uniquemember</b>	LDAP attribute that identifies the members of a group.
<b>ibm.appcenter.ldap.user.groupmembership</b>	LDAP attribute that identifies the groups to which a user belongs.
<b>ibm.appcenter.ldap.group.nesting</b>	Management of nested groups: if nested groups are not managed, set the value to false.
<b>ibm.appcenter.ldap.user.filter</b>	<p>LDAP user search filter for the attribute of user login name. Use %v as the placeholder for the login name attribute.</p> <p>This property is only required when LDAP users and groups are defined in the same subtree; that is, when the properties <b>ibm.appcenter.ldap.user.base</b> and <b>ibm.appcenter.ldap.group.base</b> have the same value.</p>
<b>ibm.appcenter.ldap.displayName.filter</b>	<p>LDAP user search filter for the attribute of user display name. Use %v as the placeholder for the display name attribute.</p> <p>This property is only required when LDAP users and groups are defined in the same subtree; that is, when the properties <b>ibm.appcenter.ldap.user.base</b> and <b>ibm.appcenter.ldap.group.base</b> have the same value.</p>
<b>ibm.appcenter.ldap.group.filter</b>	<p>LDAP group search filter. Use %v as the placeholder for the group attribute.</p> <p>This property is only required when LDAP users and groups are defined in the same subtree; that is, when the properties <b>ibm.appcenter.ldap.user.base</b> and <b>ibm.appcenter.ldap.group.base</b> have the same value.</p>
<b>ibm.appcenter.ldap.security.binddn</b>	Property that identifies the distinguished name of the user permitted to search the LDAP directory. Use this property only if security binding is required.
<b>ibm.appcenter.ldap.security.bindpwd</b>	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 server.xml file.

```

<Environment name="ibm.appcenter.ldap.active" value="true" type="java.lang.String" override="false"
<Environment name="ibm.appcenter.ldap.connectionURL" value="ldap://bluepages.ibm.com:389" type="java.lang.String"
<Environment name="ibm.appcenter.ldap.user.base" value="ou=bluepages,o=ibm.com" type="java.lang.String"
<Environment name="ibm.appcenter.ldap.user.loginName" value="emailAddress" type="java.lang.String"
<Environment name="ibm.appcenter.ldap.user.displayName" value="cn" type="java.lang.String" override="false"
<Environment name="ibm.appcenter.ldap.group.base" value="ou=memberlist,ou=ibmgroups,o=ibm.com" type="java.lang.String"
<Environment name="ibm.appcenter.ldap.group.name" value="cn" type="java.lang.String" override="false"
<Environment name="ibm.appcenter.ldap.group.uniquemember" value="uniquemember" type="java.lang.String"

```

## Defining 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.

### 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, host name, 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, host name, 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`).



Figure 10. Configuration with secured reverse proxy

Table 32. The endpoint properties

Property name	Purpose	Example
<b>ibm.appcenter.services.endpoint</b>	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
<b>ibm.appcenter.proxy.protocol</b>	This property specifies the protocol required for external applications to connect to the Application Center.	https
<b>ibm.appcenter.proxy.host</b>	This property specifies the host name required for external applications to connect to the Application Center.	<b>appcntr.net</b>
<b>ibm.appcenter.proxy.port</b>	This property specifies the port required for external applications to connect to the Application Center.	443

## Configuring the endpoint of the application resources (Full Profile)

For the WebSphere Application Server full profile, configure the endpoint of the application resources in the environment entries of the **IBM\_Application\_Center\_Services** application.

### 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. Since IBM Worklight V6.0, you use the JNDI environment entries.

### Procedure

1. Log in to the WebSphere Application Server console.
2. Select **Applications > Application Types > WebSphere enterprise applications**.
3. Click **IBM Application Center Services**.
4. In the "Web Module Properties" section, select **Environment entries for Web modules**.
5. Assign the appropriate values for the following environment entries:
  - a. For **ibm.appcenter.proxy.host**, assign the hostname.
  - b. For **ibm.appcenter.ldap.port**, assign the port number.
  - c. For **ibm.appcenter.proxy.protocol**, assign the external protocol.
  - d. Click **OK** and save the configuration.
6. Select **Applications > Application Types > WebSphere enterprise applications**.
7. Click **IBM Application Center Console**.

8. In the "Web Module Properties" section, select **Environment entries for Web modules**.
9. For **ibm.appcenter.services.endpoint**, assign the full URI of the Application Center REST services (the URI of the applicationcenter.war file). 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.
10. Click **OK** and save the configuration.

## Configuring the endpoint of the application resources (Liberty profile)

For the Liberty profile, configure the endpoint of the application resources through the JNDI environment.

### Purpose

Since IBM Worklight V6.0, follow this procedure when you must change the URI protocol, hostname, and port used by the Application Center client to manage the applications on your device.

### Properties

Edit the server.xml file. To be able to define JNDI entries, the **<feature>** element must be defined correctly in the server.xml file:

```
<feature>jndi-1.0</feature>
```

Add an entry for each property in the <server> section of the server.xml file. This entry should have the following syntax:

```
<jndiEntry jndiName="JNDI_property_name" value="property_value"/>
```

Where:

JNDI\_property\_name is the name of the property you are adding.

property\_value is the value of the property you are adding.

Table 33. Properties in the server.xml file for configuring the endpoint of the application resources

Property	Description
<b>ibm.appcenter.services.endpoint</b>	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.
<b>ibm.appcenter.proxy.protocol</b>	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.
<b>ibm.appcenter.proxy.host</b>	The hostname of the application resources URI.
<b>ibm.appcenter.proxy.port</b>	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 `server.xml` file required for configuring the endpoint of the application resources.

```
<jndiEntry jndiName="ibm.appcenter.services.endpoint" value=" https://appcntr.net:443/applicationcenter" />
<jndiEntry jndiName="ibm.appcenter.proxy.protocol" value="https" />
<jndiEntry jndiName="ibm.appcenter.proxy.host" value="appcntr.net" />
<jndiEntry jndiName="ibm.appcenter.proxy.port" value=" 443"/>
```

## Configuring the endpoint of the application resources (Apache Tomcat)

For the Apache Tomcat server, configure the endpoint of the application resources in the `server.xml` file.

### Purpose

Since IBM Worklight V6.0, follow this procedure when you must change the URI protocol, hostname, and port used by the Application Center client to manage the applications on your device.

### Properties

Edit the `server.xml` file in the `conf` directory of your Apache Tomcat installation.

Add an entry for each property except **ibm.appcenter.services.endpoint** in the `<context>` section of the `IBM_Application_Center_Console` application. This entry should have the following syntax:

```
<Environment name="JNDI_property_name" value="property_value" type="property_type" override="false"/>
```

Where:

`JNDI_property_name` is the name of the property you are adding.

`property_value` is the value of the property you are adding.

`property_type` is the type of the property you are adding.

Table 34. Properties in the `server.xml` file for configuring the endpoint of the application resources

Property	Type	Description
<b>ibm.appcenter.services.endpoint</b>	<code>java.lang.String</code>	The URI of the Application Center REST services ( <code>applicationcenter.war</code> ). 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>ibm.appcenter.proxy.protocol</b>	<code>java.lang.String</code>	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.
<b>ibm.appcenter.proxy.host</b>	<code>java.lang.String</code>	The hostname of the application resources URI.

Table 34. Properties in the server.xml file for configuring the endpoint of the application resources (continued)

Property	Type	Description
<code>ibm.appcenter.proxy.port</code>	<code>java.lang.Integer</code>	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 server.xml properties for configuring the endpoint

This example shows the settings of the properties in the server.xml file required for configuring the endpoint of the application resources.

```
<Environment name="ibm.appcenter.services.endpoint" value="https://apcnet.net:443/applicationcenter" type="java.lang.String" override="true" />
<Environment name="ibm.appcenter.proxy.protocol" value="https" type="java.lang.String" override="true" />
<Environment name="ibm.appcenter.proxy.host" value="apcnet.net" type="java.lang.String" override="true" />
<Environment name="ibm.appcenter.proxy.port" value="443" type="java.lang.Integer" override="false" />
```

## Configuring Secure Sockets Layer (SSL)

Learn about configuring SSL for the Application Center on supported application servers and the limitations of certificate verification on mobile operating systems.

You can configure the Application Center with SSL or without SSL.

SSL transmits data over the network in a secured channel. You must purchase an official SSL certificate from an SSL certificate authority. The SSL certificate must be compatible with Android, iOS, and BlackBerry OS 6 and 7. Self-signed certificates do not work with the Application Center.

When the client accesses the server through SSL, the client verifies the server through the SSL certificate. If the server address matches the address filed in the SSL certificate, the client accepts the connection. For the verification to be successful, the client must know the root certificate of the certificate authority. Many root certificates are preinstalled on Android, iOS, and BlackBerry devices. The exact list of preinstalled root certificates varies between versions of mobile operating systems.

You should consult the SSL certificate authority for information about the mobile operating system versions that support its certificates.

If the SSL certificate verification fails, a normal web browser requests confirmation to contact an untrusted site. The same behavior occurs when you use a self-signed certificate that was not purchased from a certificate authority. When mobile applications are installed, this control is not performed by a normal web browser, but by operating system calls.

Some versions of Android and iOS operating systems do not support this confirmation dialog in system calls. This limitation is a reason to avoid self-signed certificates or SSL certificates that are not suited to mobile operating systems.

### Configuring SSL for WebSphere Application Server full profile

Request a Secure Sockets Layer (SSL) certificate and process the received documents to import them into the key store.



## About this task

This procedure indicates how to request an SSL certificate and import it and the chain certificate into your key store.

### Procedure

1. Create a request to a certificate authority; in the WebSphere administrative console, select **Security > SSL certificate and key management > Key stores and certificates > keystore > Personal certificate requests > New**.

Where *keystore* identifies your key store.

The request is sent to the certificate authority.

2. When you receive the SSL certificate, import it and the corresponding chain certificate into your key store by following the instructions provided by the certificate authority. In the WebSphere administrative console, you can find the corresponding option in **Security > SSL certificate and key management > Manage endpoint security configurations > node SSL settings > Key stores and certificates > keystore > Personal certificates > certificate > Receive a certificate from a certificate authority**.

Where:

- *node SSL settings* shows the SSL settings of the nodes in your configuration.
- *keystore* identifies your key store.
- *certificate* identifies the certificate that you received.

3. Create an SSL configuration. See the instructions in the user documentation that corresponds to the version of the WebSphere Application Server Full Profile that supports your applications.

You can find configuration details in the WebSphere administrative console at **Security > SSL certificate and key management > Manage endpoint security configurations > SSL Configurations**.

## Configuring SSL for Liberty profile

Create a key store, import the Secure Socket Layer (SSL) certificate, and edit the `server.xml` file to configure SSL on Liberty profile.

### About this task

Follow the steps in this procedure to configure SSL on Liberty profile.

### Procedure

1. Create a key store for your web server; use the `securityUtility` with the `createSSLCertificate` option. See Enabling SSL communication for the Liberty profile for more information.
2. Import the SSL certificate and the corresponding chain certificate into your key store by following the instructions provided by the certificate authority.
3. Enable the `ssl-1.0` Liberty feature in the `server.xml` file.

```
<featureManager>
  <feature>ssl-1.0</feature>
</featureManager>
```

4. Add the key store service object entry to the `server.xml` file. The `keyStore` element is called `defaultKeyStore` and contains the key store password. For example:

```
<keyStore id="defaultKeyStore" location="/path/to/myKeyStore.p12"
password="myPassword" type="PKCS12"/>
```



5. Make sure that the value of the **httpEndpoint** element in the `server.xml` file defines the **httpsPort** attribute. For example:  

```
<httpEndpoint id="defaultHttpEndpoint" host="*" httpPort="9080" httpsPort="9443" >
```
6. Restart the web server. Now you can access the web server by `https://myserver:9443/...`

## Configuring SSL for Apache Tomcat

Create a key store, import the Secure Socket Layer (SSL) certificate, and edit the `conf/server.xml` file to define a connector for SSL on Apache Tomcat.

### About this task

Follow the steps in this procedure to configure SSL on Apache Tomcat. See [SSL Configuration HOW-TO](#) for more details and examples of configuring SSL for Apache Tomcat.

### Procedure

1. Create a key store for your web server. You can use the Java **keytool** command to create a key store.  

```
keytool -genkey -alias tomcat -keyalg RSA -keystore /path/to/keystore.jks
```
2. Import the SSL certificate and the corresponding chain certificate into your key store by following the instructions provided by the certificate authority.
3. Edit the `conf/server.xml` file to define a connector to use SSL. This connector must point to your key store.  

```
<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
maxThreads="150" scheme="https" secure="true"
clientAuth="false" sslProtocol="TLS"
keystoreFile="/path/to/keystore.jks"
keystorePass="mypassword" />
```
4. Restart the web server. Now you can access the web server by `https://myserver:8443/...`

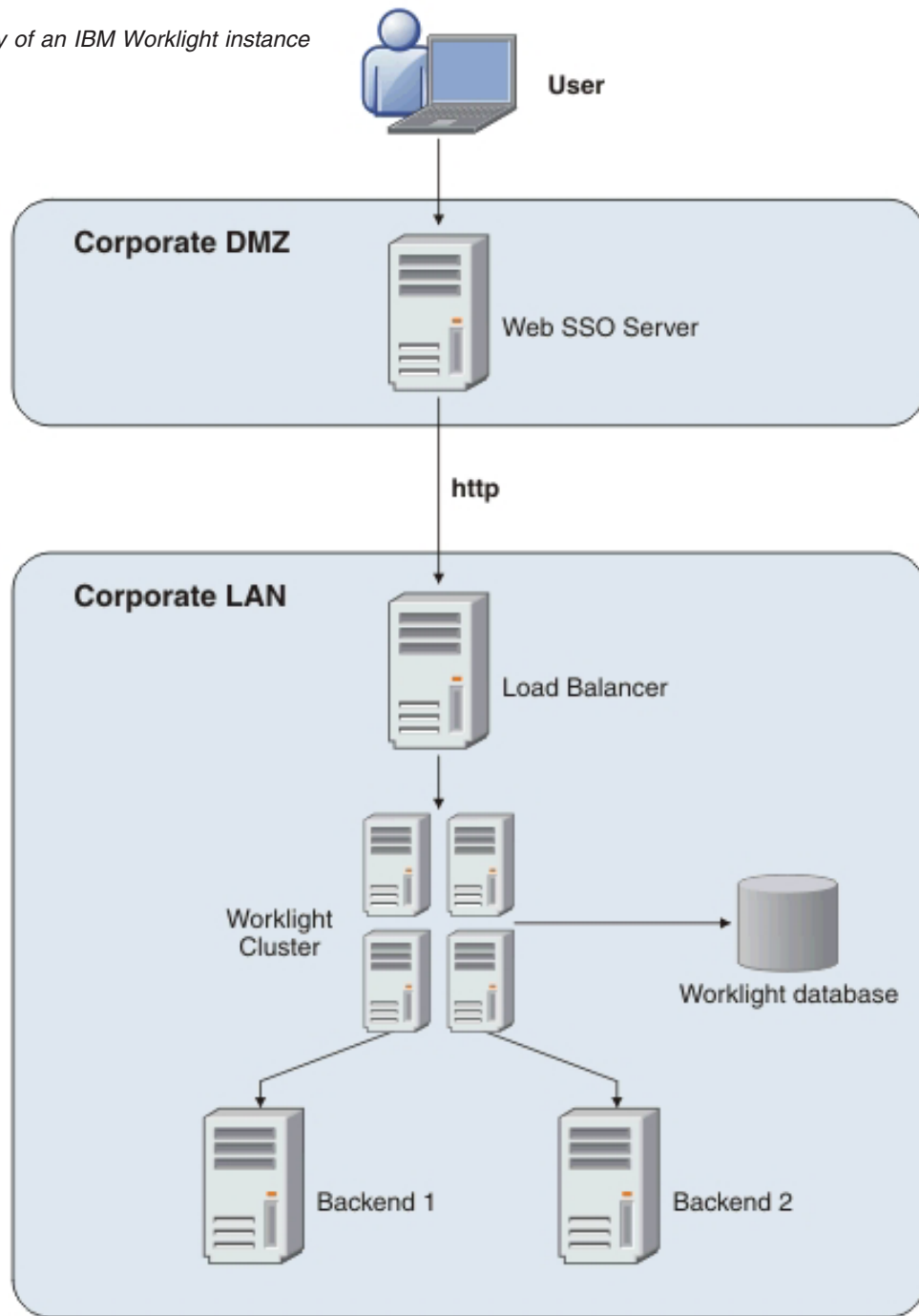
---

## Typical topologies 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 11. Typical topology of an IBM Worklight instance



Such a topology is based on the following principles:

- Worklight Server is installed in the organization LAN, connecting to various enterprise back-end systems.
- Worklight Server can be clustered for high availability and scalability.
- 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.

- Worklight Server is installed behind a web authentication infrastructure (Web SSO) acting as a reverse proxy and providing SSL.

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 Worklight Server is installed in the DMZ.

## Setting up IBM Worklight in an IBM WebSphere Application Server Network Deployment V8.5 cluster environment

You can set up an IBM Worklight cluster environment with WebSphere Application Server Network Deployment and IBM HTTP Server.

### About this task

This procedure explains how to set up IBM Worklight in the topology shown in Figure 12:

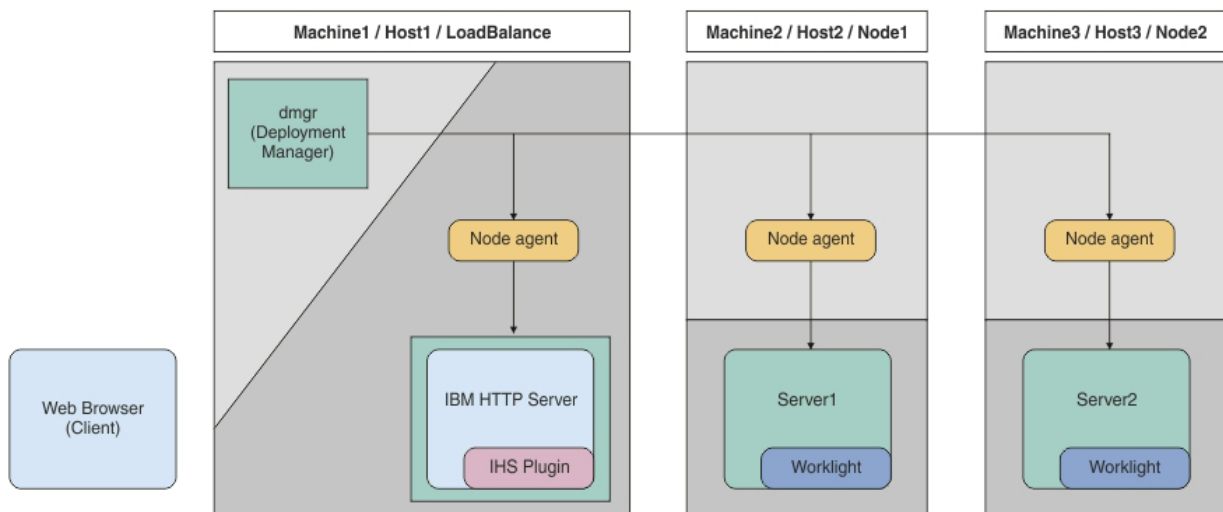


Figure 12. IBM Worklight cluster topology with IBM WebSphere Application Server Network Deployment

This procedure uses the hardware listed in Table 35 and the software listed in Table 36 on page 148.

Table 35. Hardware

Host name	Operating system	Description
Host1	RHEL 6.2	WebSphere Application Server Deployment Manager and IBM HTTP Server.
Host2	RHEL 6.2	WebSphere Application Server cluster node / server 1
Host3	RHEL 6.2	WebSphere Application Server cluster node / server 2

Table 35. Hardware (continued)

Host name	Operating system	Description
Host4	RHEL 6.2	DB2 server

Table 36. Software

Name	Description
IBM Installation Manager 1.6	Install IBM WebSphere Application Server Network Deployment, IBM HTTP Server , IBM Web Server Plug-ins for WebSphere Application Server, and IBM Worklight.
IBM WebSphere Application Server 8.5	WebSphere Application Server. You need to get the installation repository before you start this procedure.
IBM HTTP Server 8.5	IBM HTTP Server. You need to get the installation repository before you start this procedure. IBM HTTP Server is also included in the WebSphere Application Server installation repository.
Web Server Plug-ins 8.5	IBM HTTP Server Plugin. You need to get the installation repository before you start this procedure. IBM HTTP Server Plugin is also included in the WebSphere Application Server installation repository.
IBM Worklight V6.0.0	IBM Worklight. You need to get access to the installation repository before you start this procedure.
IBM DB2 V9.7 or later	DB2 Database. Your DB2 server must be available before you start the IBM Worklight installation.
Ant 1.8.3	Configure IBM Worklight with Liberty Profile Server.

## Procedure

1. Install WebSphere Application Server Network Deployment, IBM HTTP Server, and Web Server Plugins.
  - a. On the Host1 machine. log on with the "root" user ID and run IBM Installation Manager to install WebSphere Application Server Network Deployment, IBM HTTP server and Web Server Plugins. This documentation assumes that the applications are installed in the following places:
    - WebSphere Application Server Network Deployment home**  
/opt/WAS85
    - IBM HTTP Server home**  
/opt/IBM/HTTPServer
    - Web Server Plugins home**  
/opt/IBM/HTTPServer/Plugins
  - b. Repeat step 1a on Host2 and Host3, but install only WebSphere Application Server Network Deployment.
2. Create a deployment manager and nodes.

- a. To avoid network errors, add the host name and IP mapping to the `/etc/hosts` file.

**On Windows:**

Add the IP-to-host mapping to `C:\Windows\System32\drivers\etc\hosts`.

**On Linux:**

Add the IP-to-host mapping to `/etc/hosts`.

For example:

```
9.186.9.75 Host1
9.186.9.73 Host2
9.186.9.76 Host3
```

- b. Create a deployment manager and IBM HTTP Server node on Host1. You can change the profile name and profile path to suit your environment.

- 1) Create the deployment manager profile. The following command creates a profile named "dmgr":

**On Windows:**

```
./manageprofiles.bat -create -profileName dmgr
-profilePath ../profiles/dmgr -templatePath
../profileTemplates/management -severType
DEPLOYMENT_MANAGER
```

**On Linux:**

```
./manageprofiles.sh -create -profileName dmgr
-profilePath ../profiles/dmgr506 -templatePath
../profileTemplates/management -severType
DEPLOYMENT_MANAGER
```

- 2) Create an IBM HTTP Server node profile. The following command creates a profile named "ihs":

**On Windows:**

```
./manageprofiles.bat -create -profileName ihs
-profilePath ../profiles/ihs -templatePath
../profileTemplates/managed
```

**On Linux:**

```
./manageprofiles.sh -create -profileName ihs -profilePath
../profiles/ihs506 -templatePath ../profileTemplates/
managed
```

- 3) Start the deployment manager:

**On Windows:**

```
./startManager.bat
```

**On Linux:**

```
./startManager.sh
```

- 4) Add an IBM HTTP Server node to the deployment manager. The following command adds the node defined by the "ihs" profile to the deployment manager running on Host1, and assigns port 8879:

**On Windows:**

```
./addNode.bat Host1 8879 -profileName ihs
```

**On Linux:**

```
./addNode.sh Host1 8879 -profileName ihs
```

- 5) From the WebSphere Application Server administrative console, click **System administration > Nodes** and check that the node is added to

the deployment manager.

Select	Name	Host Name	Version	Discovery Protocol	Status
You can administer the following resources:					
	<a href="#">topoihsCellManager01</a>	topoihs	ND 8.5.5.0	TCP	↔
<input type="checkbox"/>	<a href="#">topoihsNode01</a>	topoihs	ND 8.5.5.0	TCP	↔

**Note:** Node names might be different from the profile names you specify because WebSphere Application Server automatically generates a display name for a new node.

c. Create Worklight node1 on Host2.

1) Create a profile for the node. The following command creates a profile named node1:

**On Windows:**

```
./manageprofiles.bat -create -profileName node1
-profilePath ../profiles/node1 -templatePath
../profileTemplates/managed
```

**On Linux:**

```
./manageprofiles.sh -create -profileName node1
-profilePath ../profiles/node1 -templatePath
../profileTemplates/managed
```

2) Add the node to the deployment manager. The following command adds the node defined by the node1 profile to the deployment manager running on Host1, and assigns port 8879:

**On Windows:**

```
./addNode.bat Host1 8879 -profileName node1
```

**On Linux:**

```
./addNode.sh Host1 8879 -profileName node1
```

d. Repeat step 2c to create Worklight node2 on Host3.

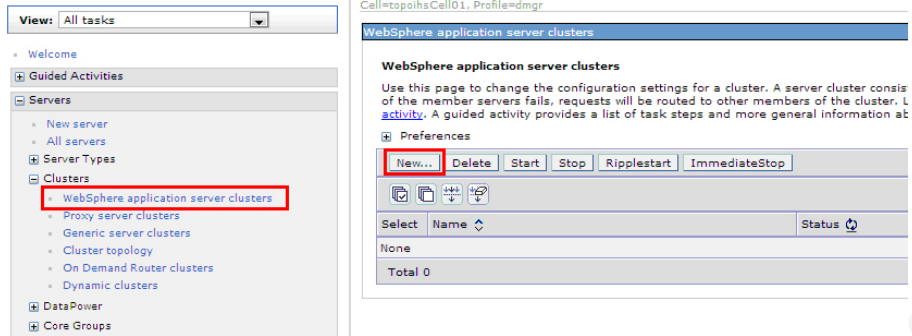
e. From the WebSphere Application Server administrative console, click **System administration > Nodes** and check that the nodes you added to the deployment manager are listed.

Select	Name	Host Name	Version	Discovery Protocol	Status
You can administer the following resources:					
	<a href="#">topoihsCellManager01</a>	topoihs	ND 8.5.5.0	TCP	↔
<input type="checkbox"/>	<a href="#">topoihsNode01</a>	topoihs	ND 8.5.5.0	TCP	↔
<input type="checkbox"/>	<a href="#">topowas1Node01</a>	topowas1	ND 8.5.5.0	TCP	↔
<input type="checkbox"/>	<a href="#">topowas2Node01</a>	topowas2	ND 8.5.5.0	TCP	↔

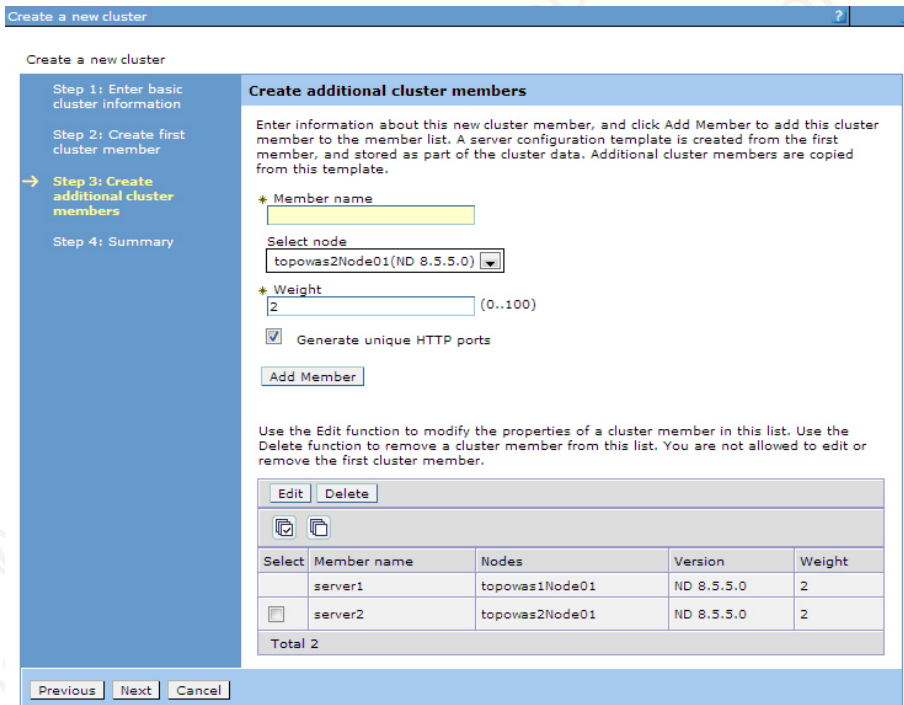
**Note:** Node names might be different from the profile names you specify because WebSphere Application Server automatically generates a display name for a new node.

3. Create a cluster and add Worklight nodes as members.

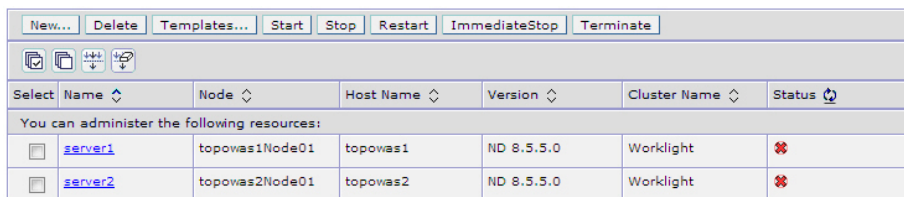
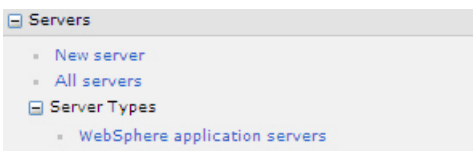
a. From the WebSphere Application Server administrative console, click **Servers > Clusters > WebSphere application server clusters**, and then click **New** to create a new cluster.



- b. For each Worklight node, add a member to the cluster: in the fields provided, enter the required information, and then click **Add Member**.



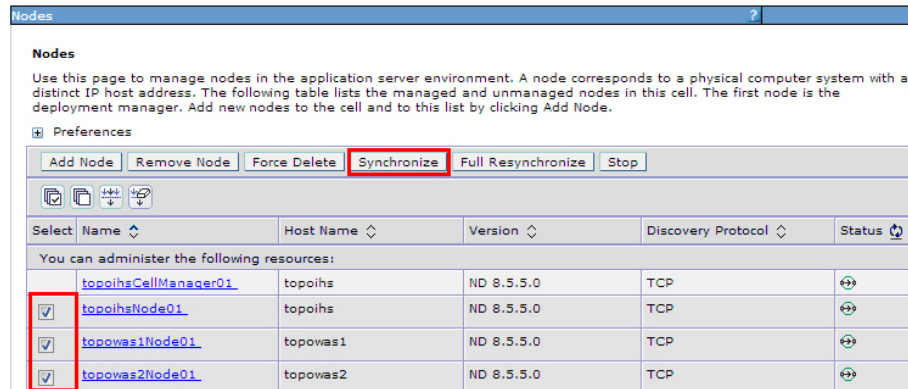
- c. From the WebSphere Application Server administrative console, click **Servers > Server Types > WebSphere application servers** to check that the cluster member servers are listed.



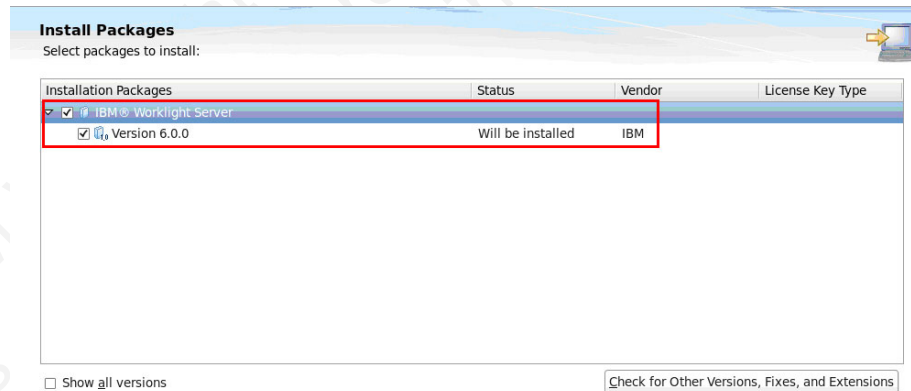
- d. If the status column indicates that nodes are not synchronized, click **System Administration > Nodes**, and then click **Synchronize** to



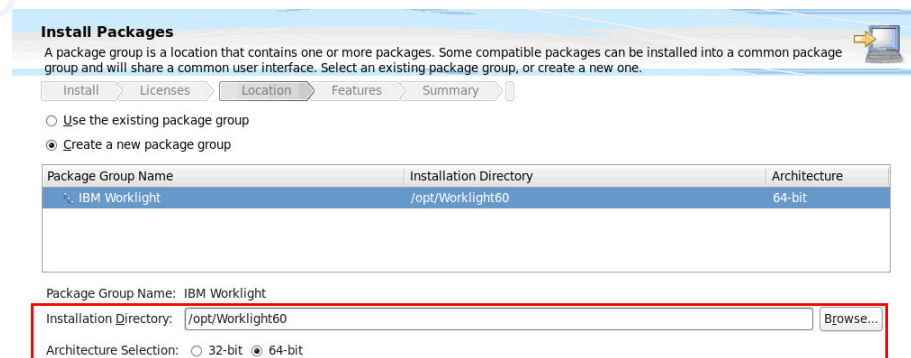
synchronize your nodes to the deployment manager.



4. Install IBM Worklight. Ensure that the WebSphere Application Server Network Deployment cluster is created without errors before you begin the installation.
  - a. On the Host1 machine. log on with the "root" user ID and run IBM Installation Manager. On Windows, log on with a user account in the Administrator group.
  - b. Go to **File > Preferences > Repositories** and add the IBM Worklight repository. If you are using Passport Advantage, you can skip this step. This procedure assumes that you are using the IBM Worklight repository.
  - c. In IBM Installation Manager, from the list of packages available for installation, select **IBM Worklight Server**.

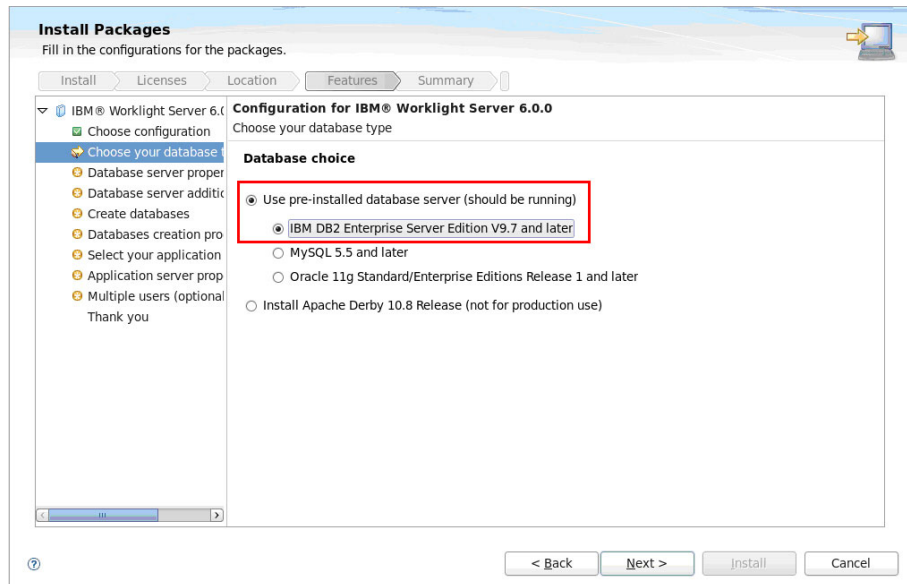


- d. In the **Installation Directory** field, accept the default installation directory, or enter the required directory.

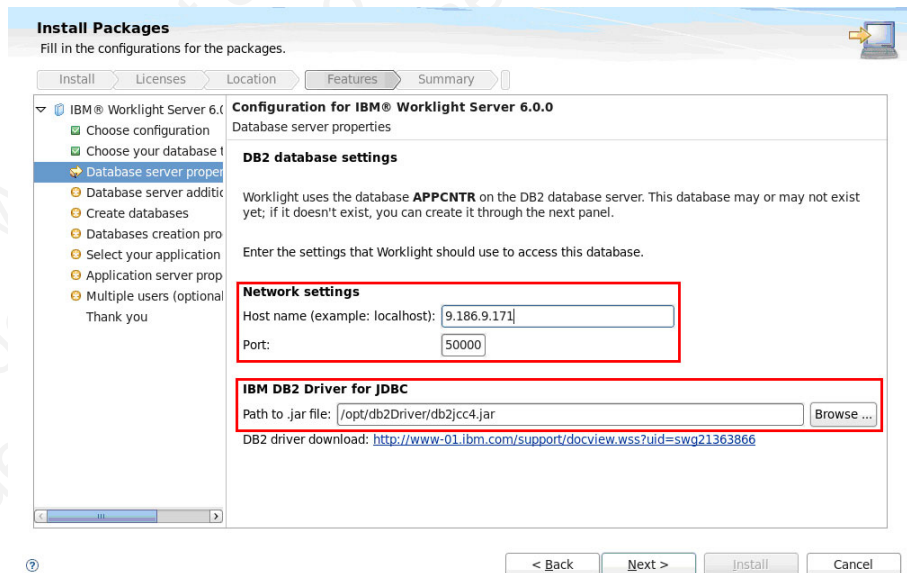


- e. Click **Next** repeatedly until the Features page is displayed.
  - f. In the "Choose configuration" panel, accept the default configuration.

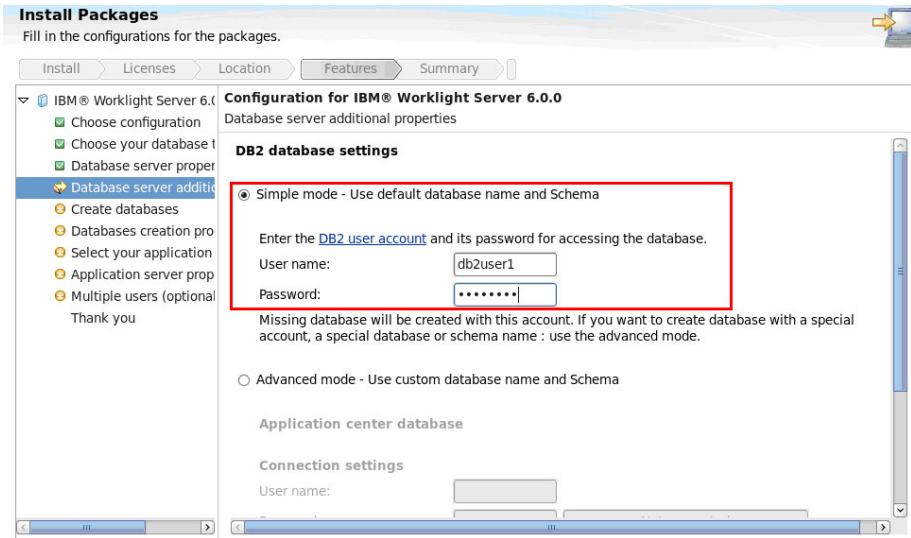
- g. In the "Database choice" panel, click **DB2 Enterprise Server Edition V9.7 and later**, and then click **Next**.



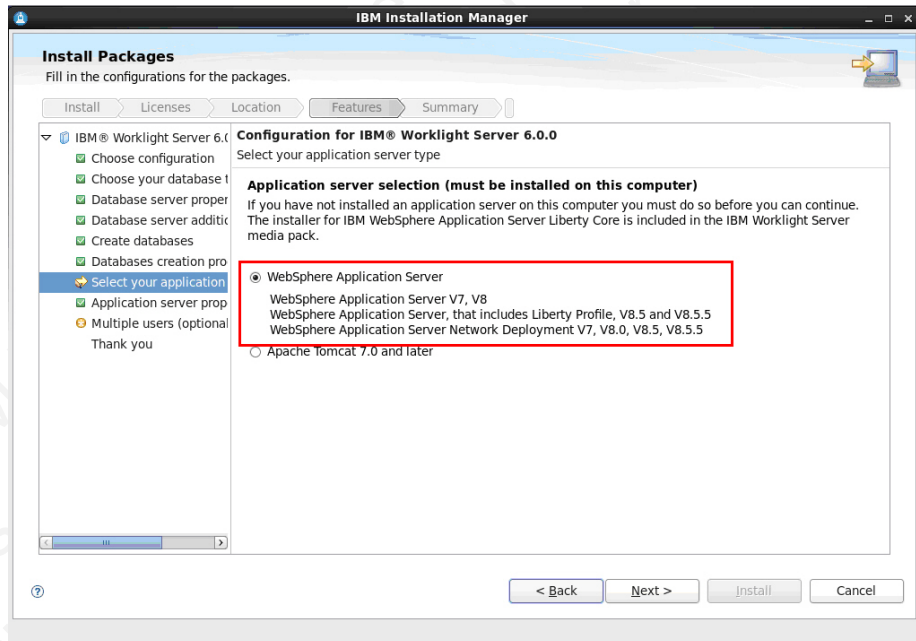
- h. In the "DB2 database settings" panel, specify network settings and the path to the JDBC driver for your Application Center database, and then click **Next**.



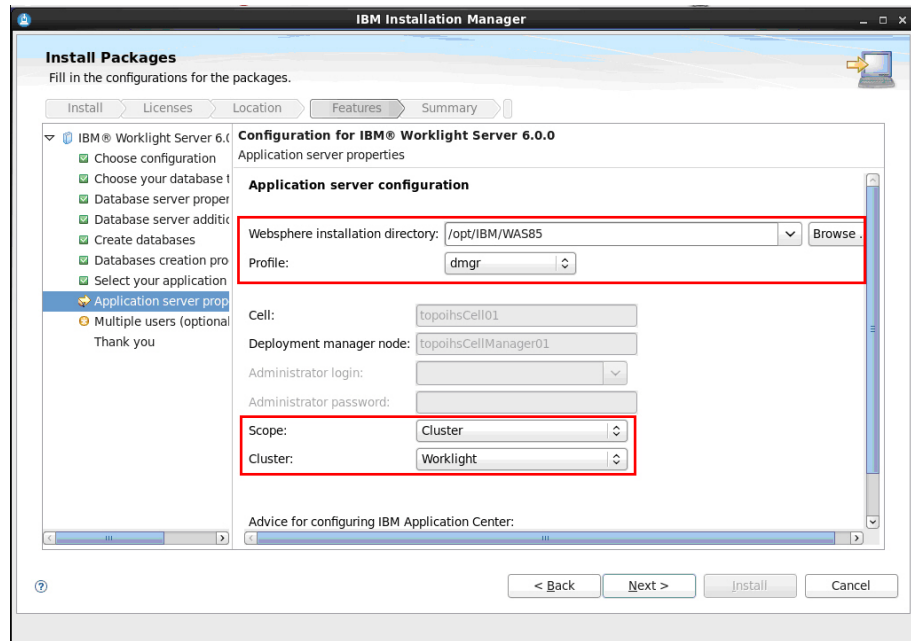
- i. Enter the user name and password of the DB2 user account for the Application Center database, and then click **Next** repeatedly until the "Configuration for IBM Worklight Server 6.0.0" panel is displayed.



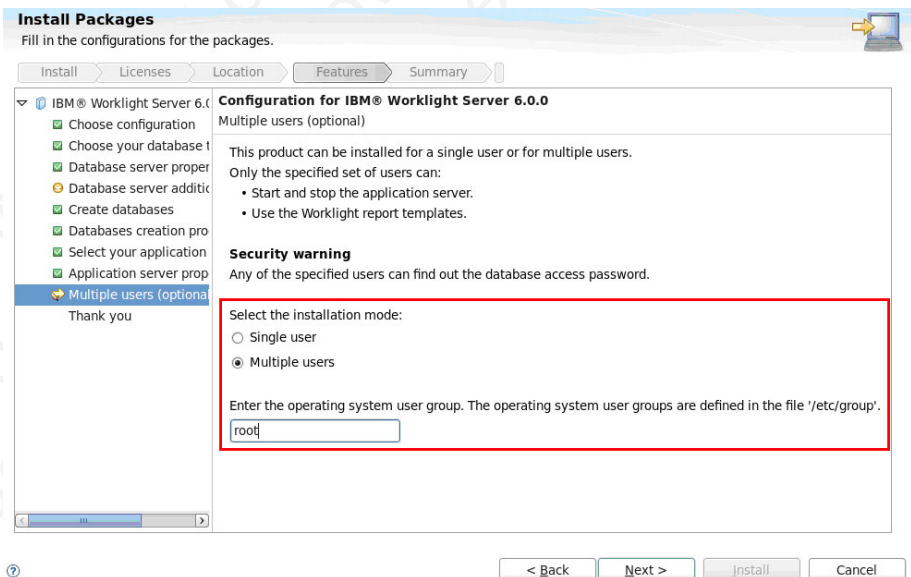
- j. In the "Configuration for IBM Worklight Server 6.0.0" panel, click **WebSphere Application Server**, and then click **Next**.



- k. In the "Application server configuration" panel, enter the WebSphere installation home directory and the deployment manager profile you specified in step 1, then from the **Scope** list, select **Cluster**, and select the cluster name.

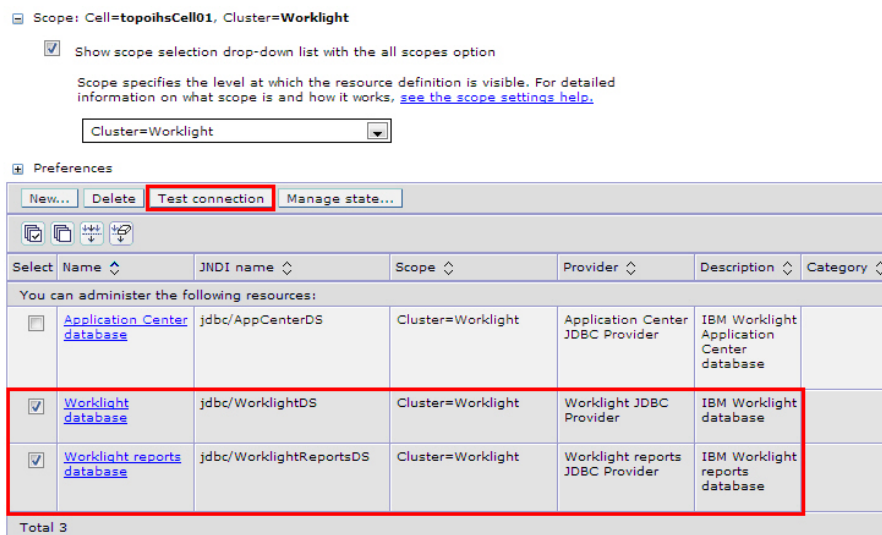


- i. In the "Multiple users" panel, click **Single user** or **Multiple users** according to whether you want to install IBM Worklight in single-user or multi-user mode, and then click **Next** repeatedly and then click **Install**.



5. Configure the databases. For instructions, see "Creating and configuring the databases with Ant tasks" on page 669.
6. In Worklight Studio, create an IBM Worklight project and build a Worklight project WAR file. See "Artifacts produced during development cycle" on page 238.
7. Configure IBM Worklight with the WebSphere Application Server Network Deployment cluster. For instructions, see "Deploying a project WAR file and configuring the application server with Ant tasks" on page 694. Modify the Ant template to match your WebSphere Application Server cluster and database server.
8. Verify the installation.

- a. Restart the WebSphere Application Server cluster.
- b. From the WebSphere Application Server administrative console, click **Resources > JDBC > Data sources**, and check that the data sources jdbc/WorklightDS and jdbc/WorklightReportsDS exist.

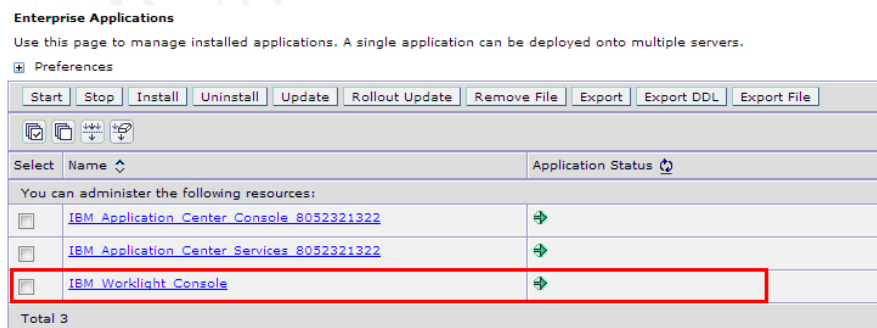


- c. Select the two data sources and click **Test connection** to verify the DB2 database connection. Confirmations similar to the ones in the following messages indicate a successful connection.

The test connection operation for data source Application Center database on server nodeagent at node topoas1Node01 was successful.

The test connection operation for data source Application Center database on server nodeagent at node topoas2Node01 was successful.

- d. Go to **Applications > Application Types > WebSphere enterprise applications** and check that the Worklight Console application is running.

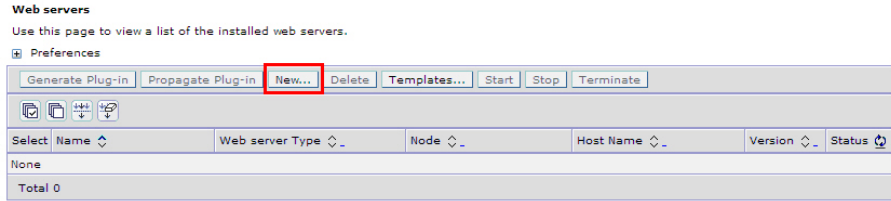


- e. Now that you have deployed IBM Worklight on the two node servers, you can access the Worklight Console on each host by browsing to the associated URLs:
  - http://Host2:9080/worklight/console
  - http://Host3:9080/worklight/console

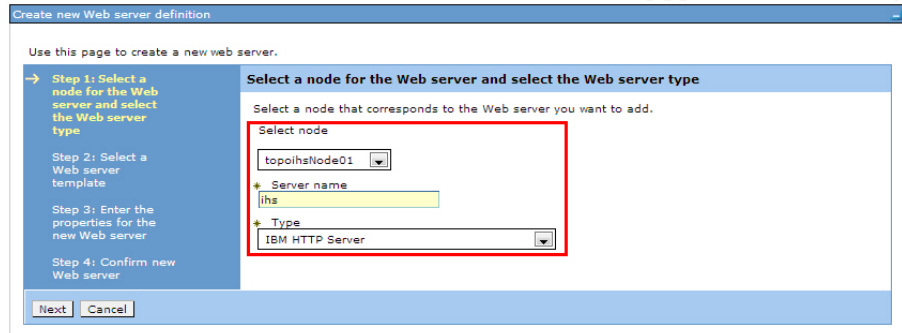
Check that both Worklight Consoles are running.

## 9. Configure the IBM HTTP Server.

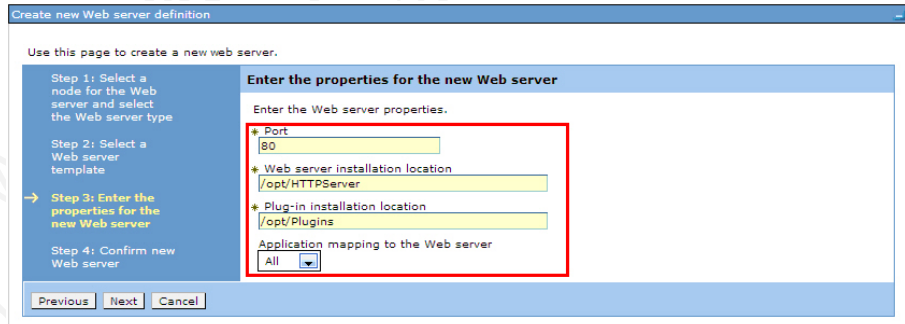
- a. From the WebSphere Application Server administrative console, go to **Servers > Server Types > Web servers**, and then click **New** to create a new IBM HTTP server.



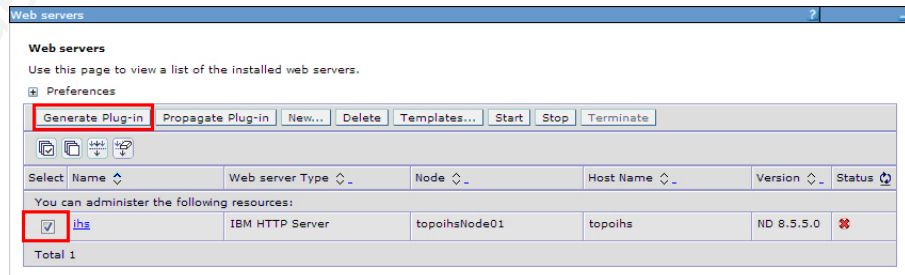
- b. Select the "ihs" node you previously created on Host1, then from the **Type** list, select **IBM HTTP Server**, and then click **Next**.



- c. Enter the IBM HTTP Server home and Web Server Plugins home you previously selected on Host1, and then click **Next** and save your configuration.

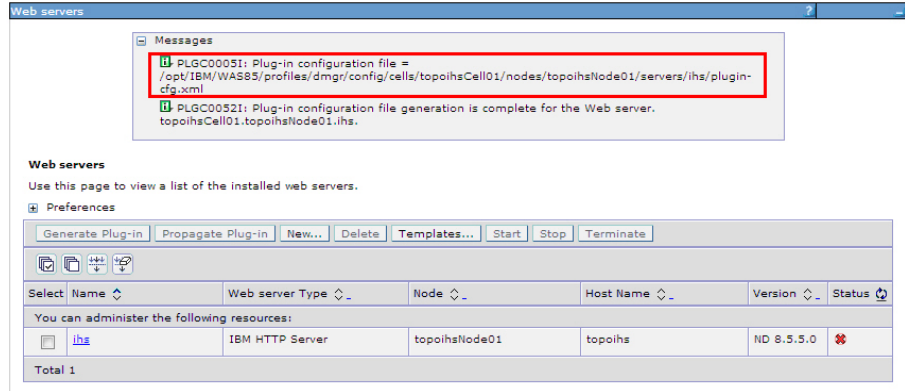


- d. In the administrative console, on the Web servers page, click **Generate Plug-in** to generate the plug-in configuration file.

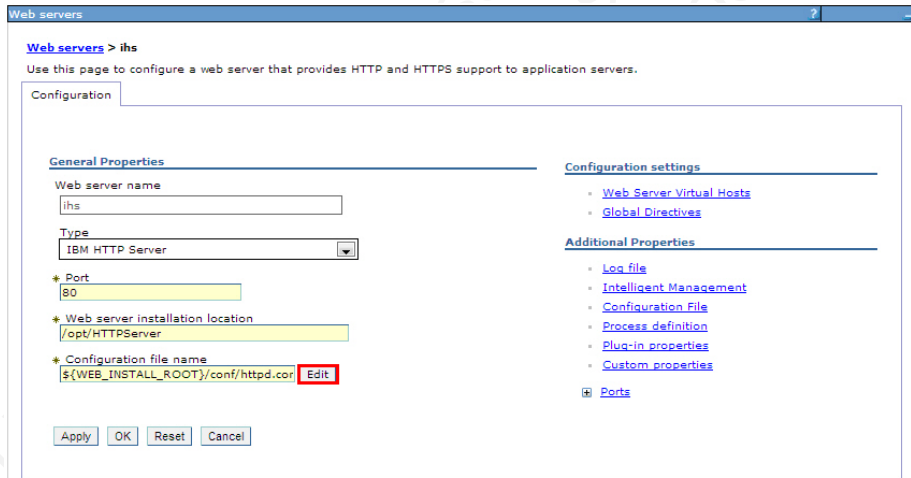


A confirmation message is displayed.





- e. Make a note of the plugin-cfg.xml location displayed in the confirmation message.
- f. In the administrative console, on the Web servers page, click "ihs", and then in the **Configuration file name** field, click **Edit**.



- g. In the editor, add a was\_ap22\_module and a WebSpherePluginConfig configuration to your http.conf file by adding the following text:

**On Windows:**

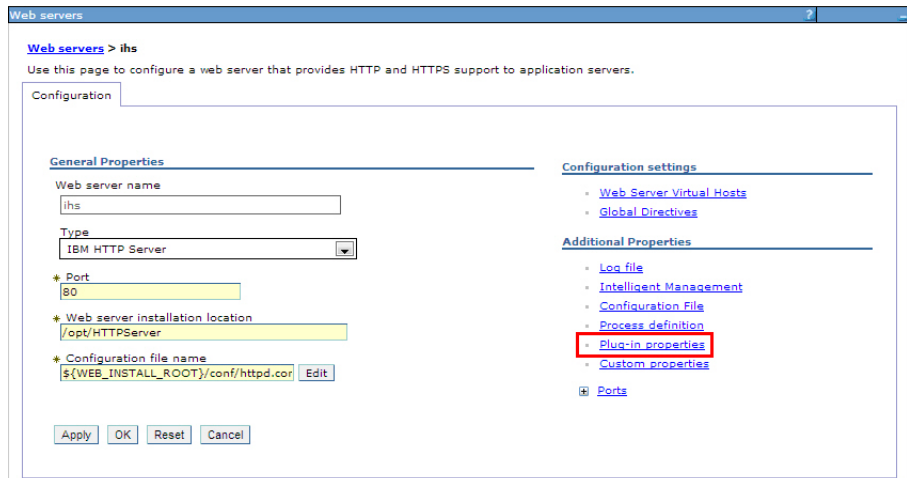
```
LoadModule was_ap22_module {IHS_Plugin_HOME}/bin/{64bits}/mod_was_ap22_http.dll
WebSpherePluginConfig {path to}/plugin-cfg.xml
```

**On Linux:**

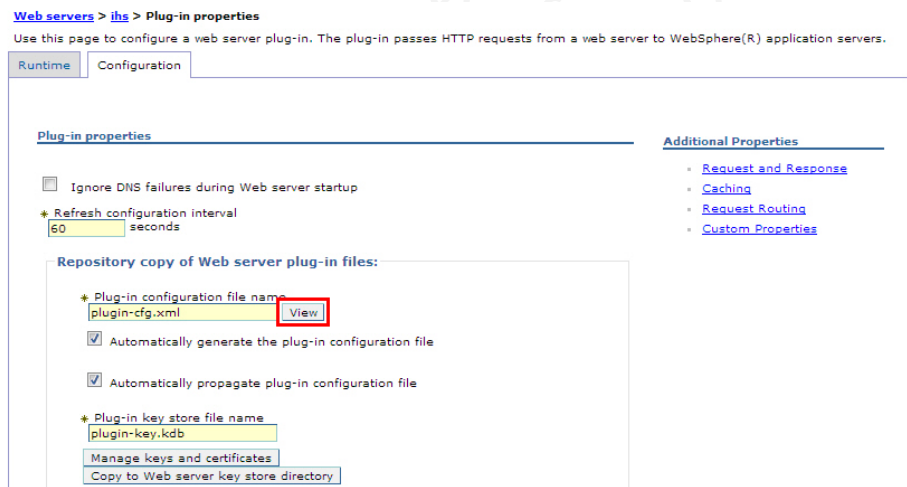
```
LoadModule was_ap22_module {IHS_Plugin_HOME}/bin/{64bits}/mod_was_ap22_http.so
WebSpherePluginConfig {path to}/plugin-cfg.xml
```

- h. In the administrative console, on the Web servers page for the "ihs" server, click **Plug-in properties**.





- i. In the **Plug-in Configuration file name** field, click **View**.



- j. Search for the cluster node and Worklight URI in the plugin-cfg.xml file.  
For example:

```
<ServerCluster CloneSeparatorChange="false" GetDWLMTTable="false" IgnoreAffinityRequests="t
  <Server CloneID="17oi91u2o" ConnectTimeout="5" ExtendedHandshake="false" LoadBalance
    <Transport Hostname="topowas1" Port="9080" Protocol="http"/>
    <Transport Hostname="topowas1" Port="9443" Protocol="https">
      <Property Name="keyring" Value="/opt/Plugins/config/ihs/plugin-key.kdb"/>
      <Property Name="stashfile" Value="/opt/Plugins/config/ihs/plugin-key.sth"/>
    </Transport>
  </Server>
  <Server CloneID="17oi9m7kg" ConnectTimeout="5" ExtendedHandshake="false" LoadBalance
    <Transport Hostname="topowas2" Port="9080" Protocol="http"/>
    <Transport Hostname="topowas2" Port="9443" Protocol="https">
      <Property Name="keyring" Value="/opt/Plugins/config/ihs/plugin-key.kdb"/>
      <Property Name="stashfile" Value="/opt/Plugins/config/ihs/plugin-key.sth"/>
    </Transport>
  </Server>
  <PrimaryServers>
    <Server Name="topowas1Node01_server1"/>
    <Server Name="topowas2Node01_server2"/>
  </PrimaryServers>
</ServerCluster>
<UriGroup Name="default_host_Worklight_URIs">
```

```

<Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/appcentercon
<Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/worklight/*"
<Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/application
</UriGroup>

```

If your configuration file does not include cluster servers and URIs, delete the "ihs" server and create it again.

- k. Optional: On the Plug-in properties page for the "ihs" server, click **Request Routing** if you want to set a load-balancing policy.

#### Additional Properties

- [Request and Response](#)
- [Caching](#)
- **[Request Routing](#)**
- [Custom Properties](#)

#### Request routing

Load balancing option  
 Round Robin ▾

\* Retry interval  
 1 seconds

Maximum size of request content

No Limit  
 Set Limit  KBytes

\* Maximum buffer size used when reading the HTTP request content  
 0 KBytes

Remove special headers  
 Clone separator change

Apply OK Reset Cancel

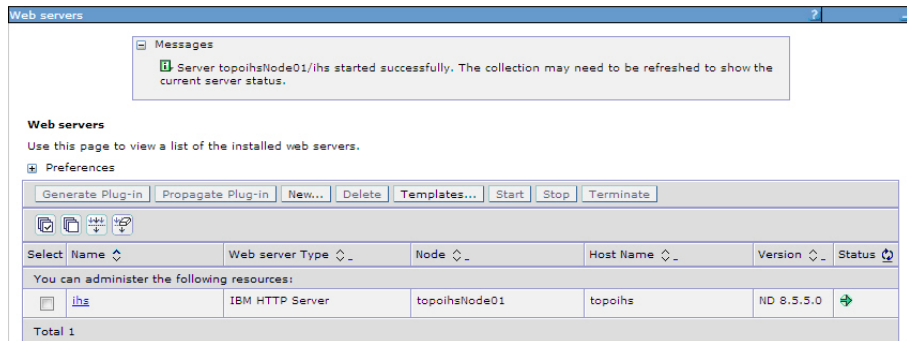
- l. Optional: On the Plug-in properties page for the "ihs" server, click **Caching** if you want to configure caching.

#### Caching

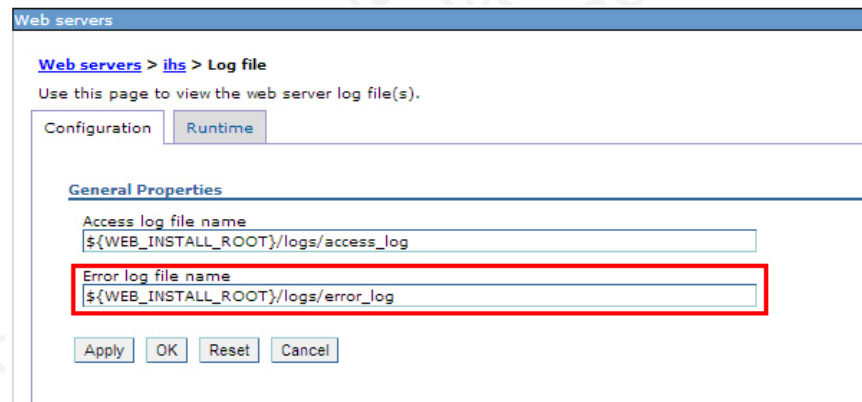
Enable Edge Side Include (ESI) processing to cache the responses  
 Enable invalidation monitor to receive notifications  
 Maximum cache size  
 1024 KB

Apply OK Reset Cancel

10. Start the IBM HTTP server and verify that the server is running.
  - a. In the WebSphere Application Server administrative console, go to **Servers > Server Types > Web servers**.
  - b. Select the IBM HTTP server you created (in this example, named "ihs"), and then click **Start**.



- c. If the server fails to start, check the log file. To find the location of the log file:
  - 1) In the administrative console, on the Web servers page for the "ihs" server, click **Log file**.
  - 2) On the log file page, click the **Configuration** tab.
  - 3) The location of the log file is displayed in the **Error log file name** field.



- d. To verify that the IBM HTTP server is running, enter the URL for the Worklight Console in a web browser. For example: `http://Host1:80/worklight/console`.

## Results

IBM Worklight is now installed on an IBM WebSphere Application Server Network Deployment cluster, and is ready for use.

## Setting up IBM Worklight in an IBM WebSphere Application Server Liberty Profile farm

You can set up an IBM Worklight cluster environment with Liberty Profile.

### About this task

This procedure explains how to set up IBM Worklight in the topology similar to the one shown in Figure 13 on page 162:

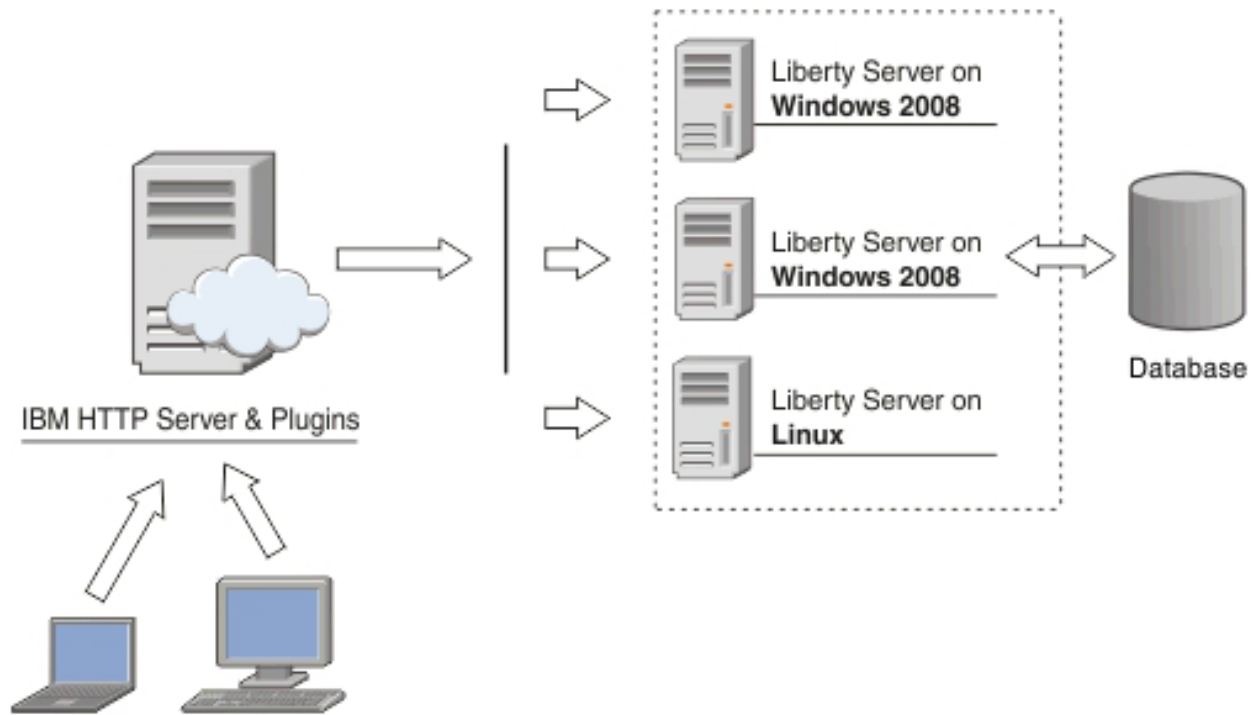


Figure 13. IBM Worklight cluster topology with Liberty Profile

This procedure uses the hardware listed in Table 37 and the software listed in Table 38.

Table 37. Hardware

Hostname	Operating system	Description
Host1	RHEL 6.2	IBM HTTP server with Web Server plug-in, acting as load balancer.
Host2	RHEL 6.2	Liberty farm server 1
Host3	RHEL 6.2	Liberty farm server 2
Host4	RHEL 6.2	DB2 server

Table 38. Software

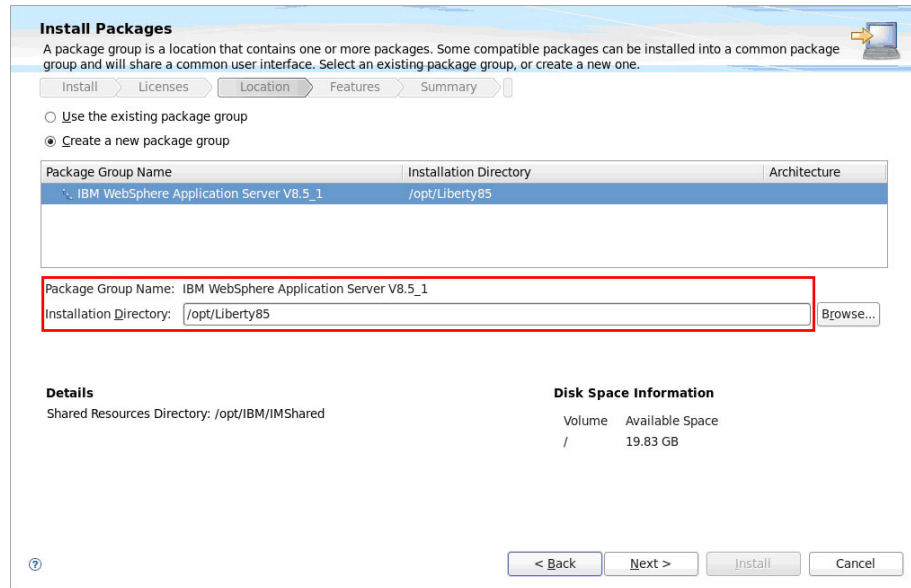
Name	Description
IBM Installation Manager 1.6	Install IBM HTTP Server , Liberty server, and IBM Worklight.
IBM HTTP Server 8.5	IBM HTTP Server. You need to get the installation repository before you start this procedure. IBM HTTP Server is also included in the WebSphere Application Server installation repository.
Web Server Plug-ins 8.5	IBM HTTP Server Plugin. You need to get the installation repository before you start this procedure. IBM HTTP Server Plugin is also included in the WebSphere Application Server installation repository.

Table 38. Software (continued)

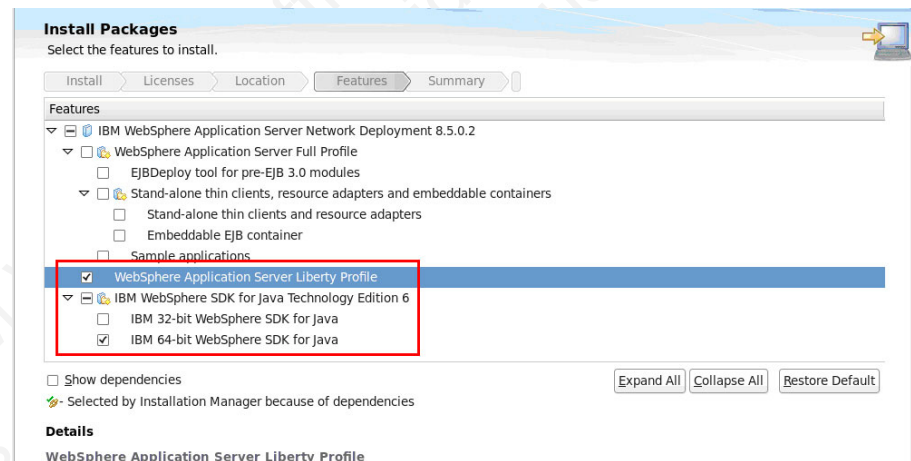
Name	Description
IBM Liberty Profile 8.5	The application server to be installed. You need to get the installation repository before you start this procedure. IBM Liberty Profile is also included in the WebSphere Application Server installation repository.
IBM Worklight V6.0.0	IBM Worklight. You need to get access to the installation repository before you start this procedure.
IBM DB2 V9.7 or later	DB2 Database. Your DB2 server must be available before you start the IBM Worklight installation.
Ant 1.8.3	Configure IBM Worklight with Liberty Profile Server.

## Procedure

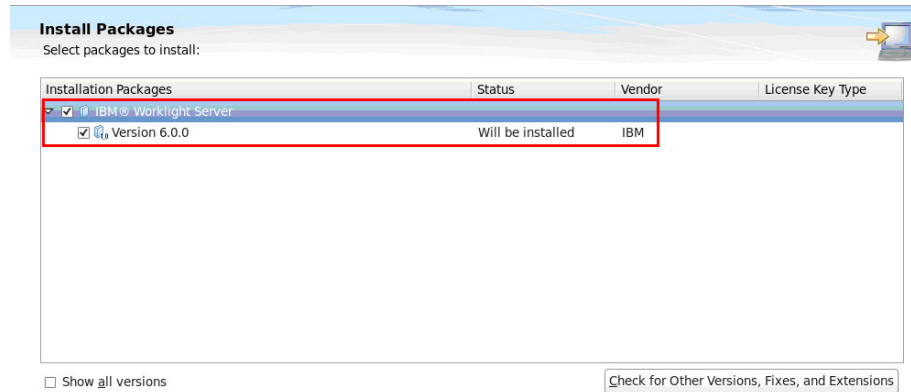
1. Install IBM HTTP Server and Web Server Plugins.
  - a. On the Host1 machine, log on with the "root" user ID and run IBM Installation Manager to install the IBM HTTP server and Web Server Plugins. This documentation assumes that the applications are installed in the following places:
    - IBM HTTP Server home**  
/opt/HTTPServer
    - Web Server Plugins home**  
/opt/Plugins
2. Install IBM WebSphere Liberty Profile.
  - a. On the Host2 machine, log in with the "root" user ID and run IBM Installation Manager in GUI mode.
  - b. Go to **File > Preferences > Repositories**, and then add the Liberty repository. If you are using the WebSphere Application Server installation repository, select IBM WebSphere Application Server Network Deployment.
  - c. In the **Installation Directory** field, accept the default installation directory or enter an alternative directory, and then click **Next**.



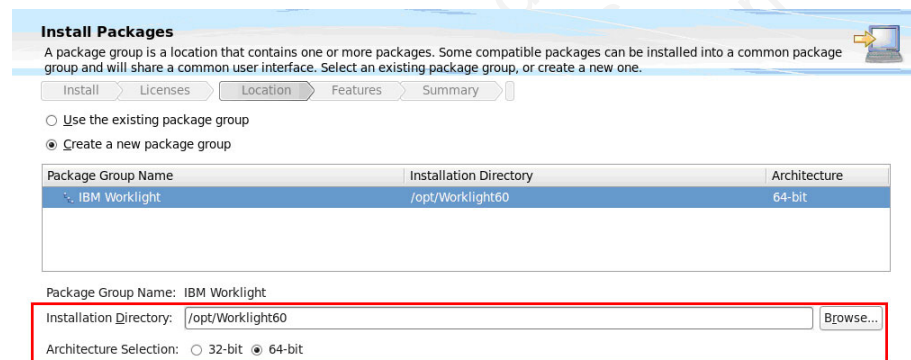
- d. Select the **WebSphere Application Server Liberty Profile** check box and clear the **WebSphere Application Server Full Profile** check box, and then click **Next** repeatedly until the installation is complete.



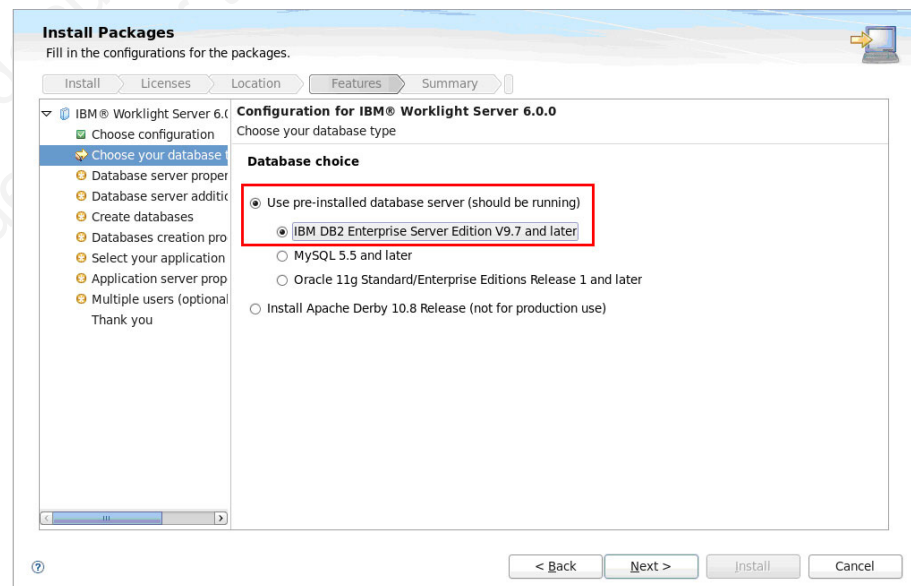
- e. Repeat the previous steps to install another Liberty Profile server on Host3.
3. Install IBM Worklight
    - a. On the Host2 machine. log on with the "root" user ID and run IBM Installation Manager. On Windows, log on with a user account in the Administrator group.
    - b. Go to **File > Preferences > Repositories** and add the IBM Worklight repository. If you are using Passport Advantage, you can skip this step. This procedure assumes that you are using the IBM Worklight repository.
    - c. In IBM Installation Manager, from the list of packages available for installation, select **IBM Worklight Server**.



- d. In the **Installation Directory** field, accept the default installation directory, or enter the required directory.

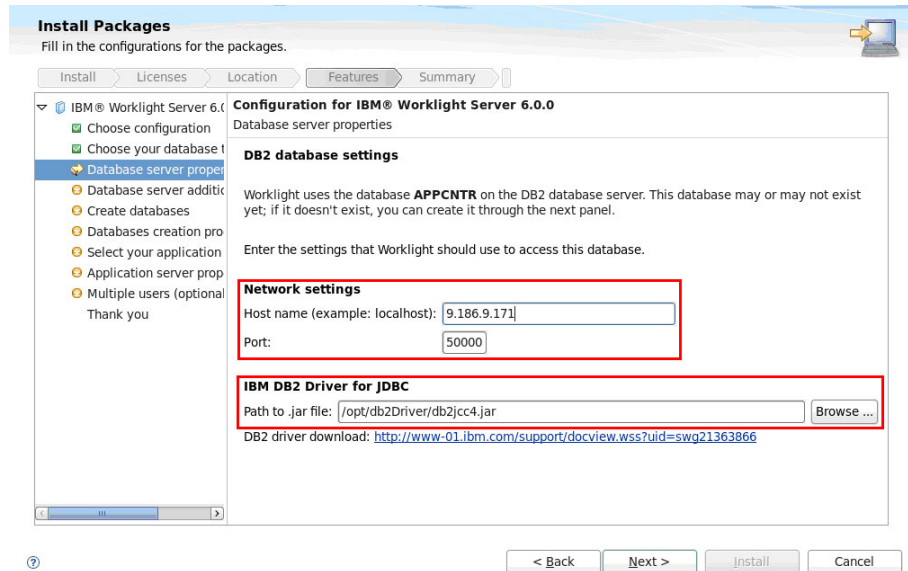


- e. Click **Next** repeatedly until the Features page is displayed.
- f. In the "Choose configuration" panel, accept the default configuration.
- g. In the "Database choice" panel, click **DB2 Enterprise Server Edition V9.7 and later**, and then click **Next**.

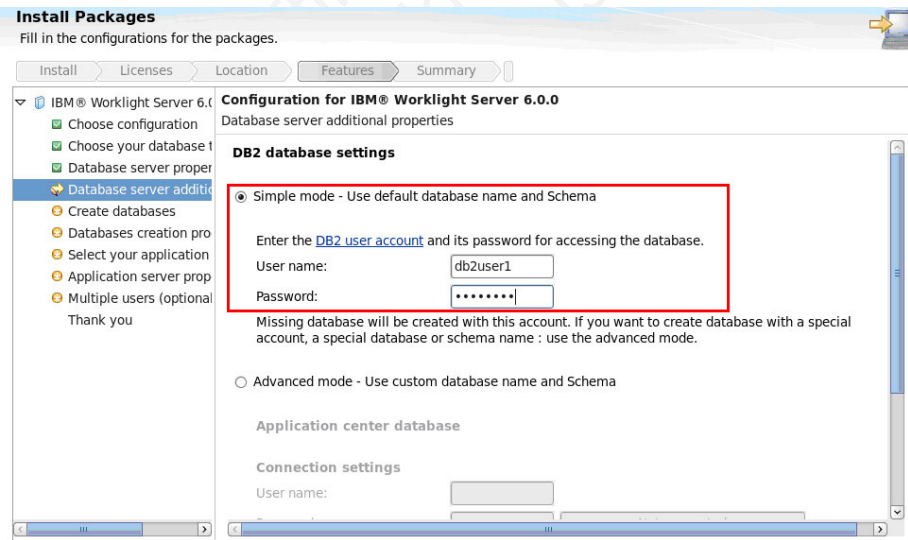


- h. In the "DB2 database settings" panel, specify network settings and the path to the JDBC driver for your Application Center database, and then click **Next**.

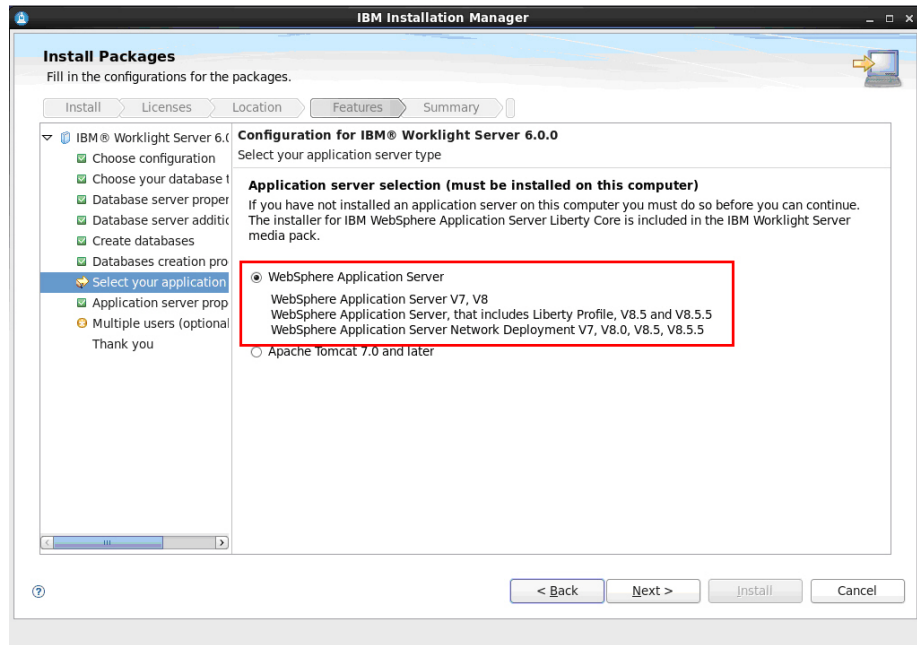




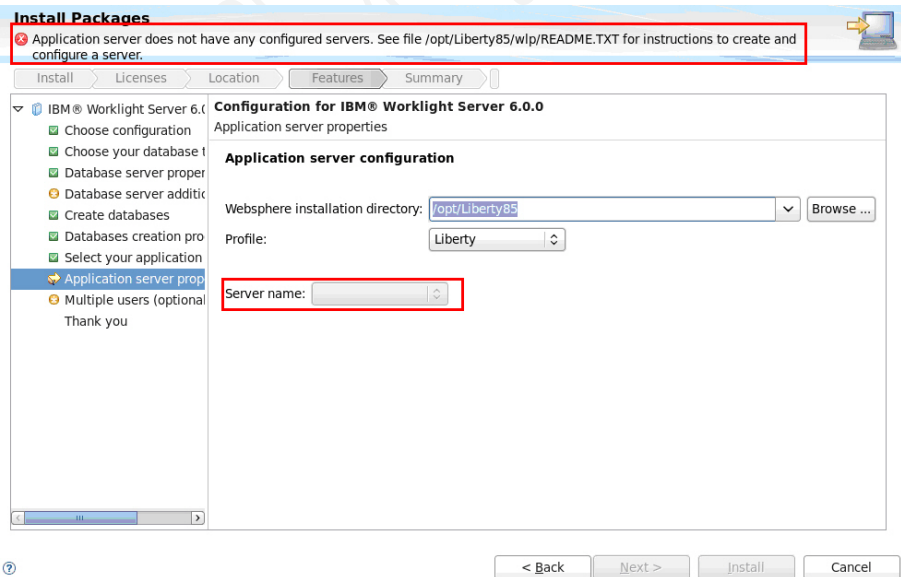
- i. Enter the user name and password of the DB2 user account for the Application Center database, and then click **Next** repeatedly until the "Configuration for IBM Worklight Server 6.0.0" panel is displayed.



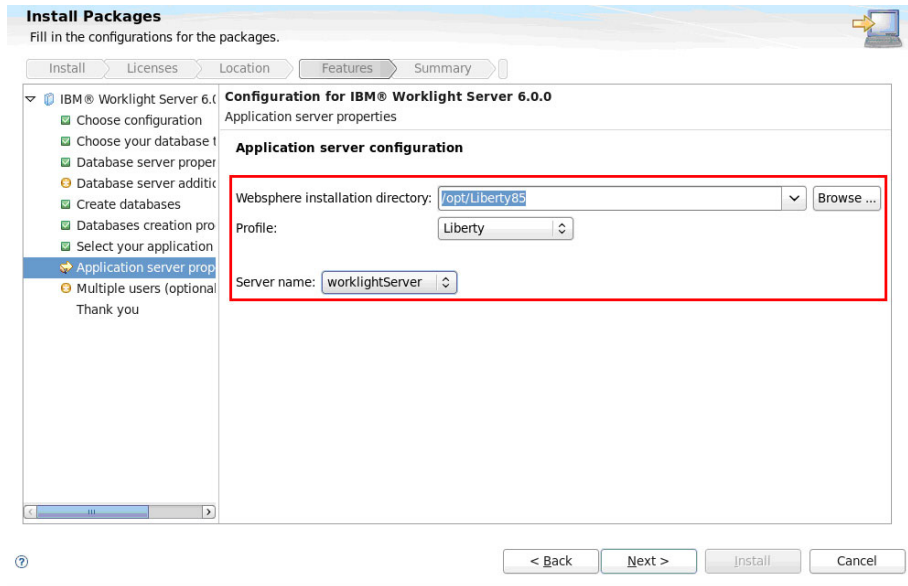
- j. In the "Configuration for IBM Worklight Server 6.0.0" panel, click **WebSphere Application Server**, and then click **Next**.



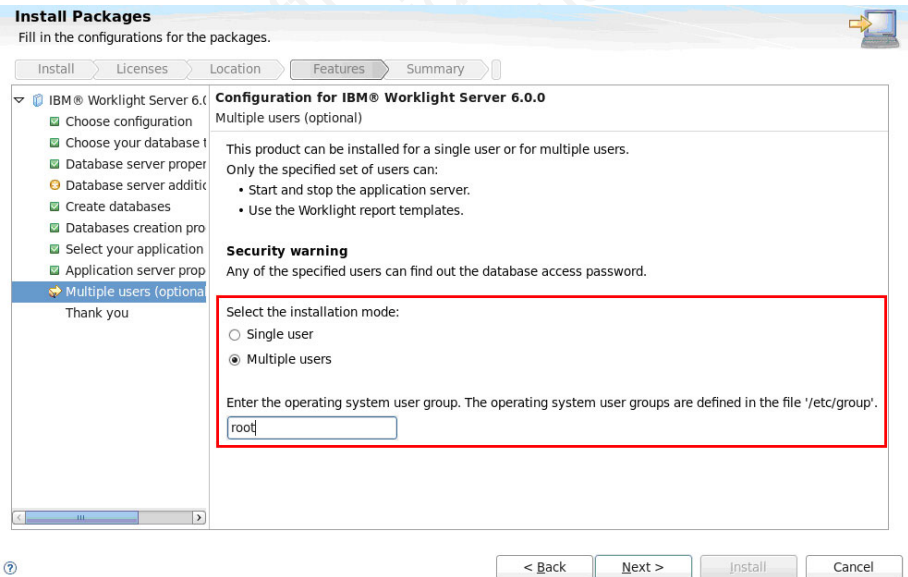
- k. In the "Application server configuration" panel, enter the Liberty Profile installation home directory you specified in step 1.



If the **Server name** field is empty, follow the instructions in `{Liberty_HOME}/wlp/README.TXT`. When the Liberty server is available, the installation wizard displays the information shown in the following picture.



- i. In the "Multiple users" panel, click **Single user** or **Multiple users** according to whether you want to install IBM Worklight in single-user or multi-user mode, and then click **Next** repeatedly and then click **Install**.



- m. Repeat the previous steps to install Worklight Server on Host3.
4. Configure the IBM Worklight databases. For instructions, see "Creating and configuring the databases with Ant tasks" on page 669.
5. In Worklight Studio, create an IBM Worklight project and build a Worklight project WAR file. See "Artifacts produced during development cycle" on page 238.
6. Configure IBM Worklight with Liberty Profile on Host 2 and Host 3. For instructions, see "Deploying a project WAR file and configuring the application server with Ant tasks" on page 694. Modify the Ant template to match your Liberty Profile server and database server.
7. Start the Liberty Profile servers to test whether you can access the Worklight Console on Host2 and Host3 by browsing to the associated URLs:

- `http://Host2:9080/worklight/console`
- `http://Host3:9080/worklight/console`

Check that both Worklight Consoles are running.

8. Run the following command on Host1 to start the IBM HTTP server.

```
/opt/HTTPServer/bin/httpd -d /opt/HTTPServer -k start -f /opt/HTTPServer/conf/httpd.conf
```

If you encounter problems during IBM HTTP server startup, see “Troubleshooting IBM HTTP Server startup” on page 172.

9. Ensure that the IBM HTTP Server can be accessed at the following URL in a web browser:

```
http://<hostname>:<port>
```

10. For each Liberty server, generate a web server plug-in configuration file named `plugin-cfg.xml`. The web server plug-in is used to forward HTTP requests from the web server to the application server.

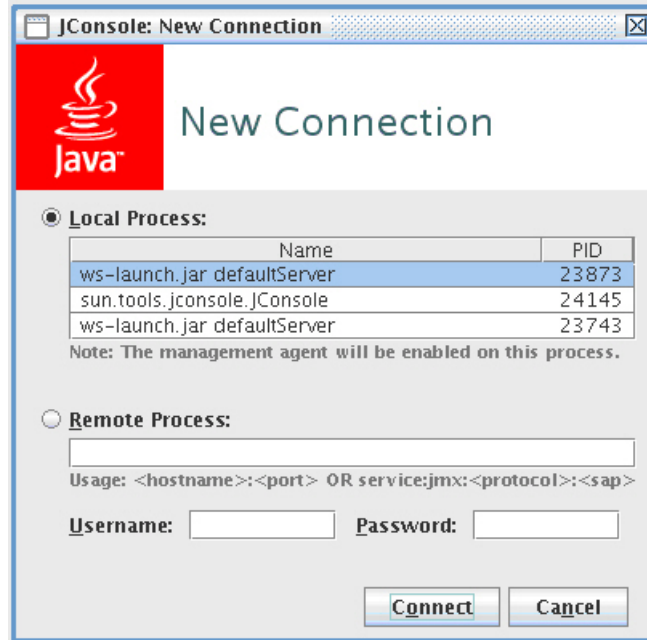
- a. Start the server that hosts your applications and ensure that the `localConnector-1.0` feature and other required features are included in the server configuration. Use the `pluginConfiguration` element in the server configuration file to specify the `webserverPort` and `webserverSecurePort` attributes for requests that are forwarded from the web server. By default, the value of `webserverPort` is 80 and the value of `webserverSecurePort` is 443. Assign the value `*` to the `host` attribute to ensure that applications on the Liberty server can be accessed from a remote browser. Here is an example of a `server.xml` server configuration file:

```
<server description="new server">
  <featureManager>
    <feature>localConnector-1.0</feature>
    <feature>jsp-2.2</feature>
  </featureManager>
  <httpEndpoint id="defaultHttpEndpoint" host="*" httpPort="9080">
    <tcpOptions soReuseAddr="true" />
  </httpEndpoint>
  <pluginConfiguration webserverPort="80" webserverSecurePort="443"/>
</server>
```

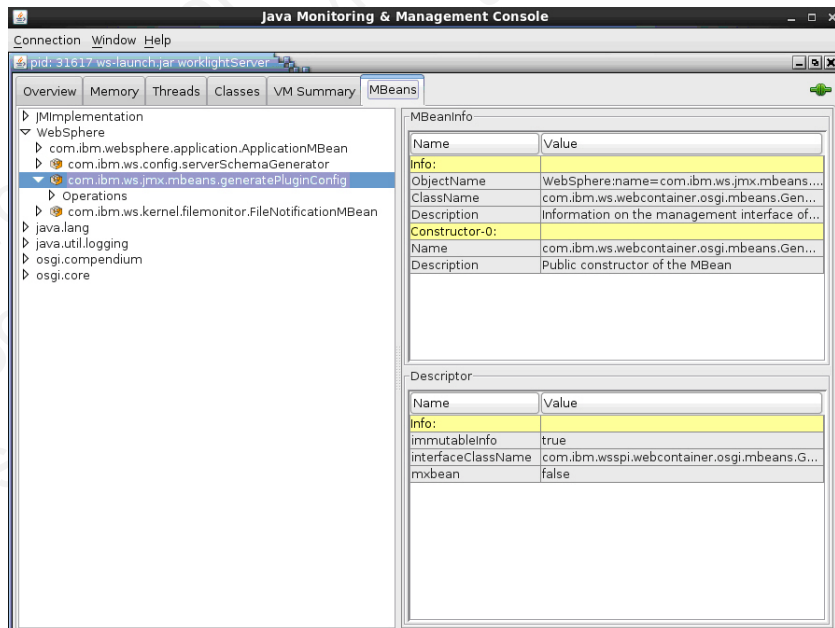
- b. Use one of the following methods to generate the `plugin-cfg.xml` file for the Liberty server running your application.

- jConsole:

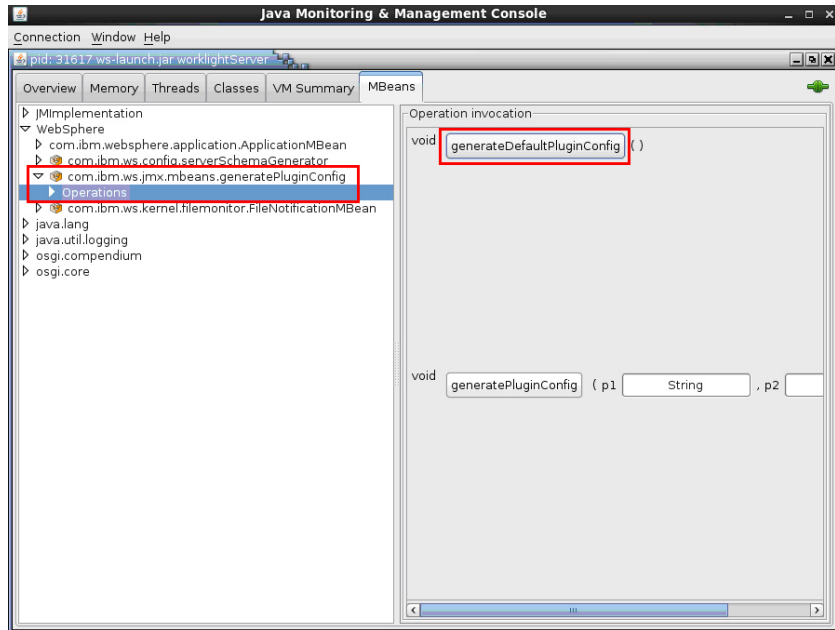
- 1) Using the same JDK as the server, run the jConsole Java utility from a command prompt. For example, run the following command:  
`C:\java\bin\jconsole`
- 2) In the jConsole window, click **Local Process**, click the server process in the list of local processes, and then click **Connect**.



- 3) In the Java Monitoring & Management Console, click the MBeans tab.



- 4) Select and invoke the defaultPluginConfig generation MBean operation to generate the plugin-cfg.xml file.



You can find the generated file in the `\wlp\usr\servers\  
<server_name>` directory. Here is an example:

```
<?xml version="1.0" encoding="UTF-8"?>
<Config ASDisableNagle="false" AcceptAllContent="false" AppServerPortPreference="Ho
  <Log LogLevel="Error" Name="String\logs\String\http_plugin.log"/>
  <Property Name="ESIEnable" Value="true"/>
  <Property Name="ESIMaxCacheSize" Value="1024"/>
  <Property Name="ESIInvalidationMonitor" Value="false"/>
  <Property Name="ESIEnableToPassCookies" Value="false"/>
  <Property Name="PluginInstallRoot" Value="String"/>
  <VirtualHostGroup Name="default_host">
    <VirtualHost Name="*:443"/>
    <VirtualHost Name="*:80"/>
    <VirtualHost Name="*:9080"/>
  </VirtualHostGroup>
  <ServerCluster CloneSeparatorChange="false" GetDWLMTable="false" IgnoreAffinityRe
    <Server CloneID="s56" ConnectTimeout="0" ExtendedHandshake="false" MaxConnection
      <Transport Hostname="wasvm56" Port="9080" Protocol="http"/>
    </Server>
  <PrimaryServers>
    <Server Name="default_node_String0"/>
  </PrimaryServers>
</ServerCluster>
<UriGroup Name="default_host_String_default_node_Cluster_URIs">
  <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/tri-
</UriGroup>
<Route ServerCluster="String_default_node_Cluster" UriGroup="default_host_String_c
</Config>
```

- Eclipse:
  - 1) Make sure your Liberty server is started.
  - 2) In Eclipse, in the servers panel, right-click the Liberty server, and then click **Utilities > Generate Plugin Config**.
- c. Copy the `plugin-cfg.xml` file to the machine that hosts the IBM HTTP Server web server, and then restart the web server to activate the settings in the file. Typically, you must enable the plug-in within the `httpd.conf` file of the web server by using the `LoadModule` phrase, and you must specify the location of the `plugin-cfg.xml` file using the `WebSpherePluginConfig` phrase.

### On Windows:

```
LoadModule was_ap22_module "path\to\mod_was_ap22_http.dll"  
WebSpherePluginConfig "path\to\plugin-cfg.xml"
```

### On other distributed systems:

```
LoadModule was_ap22_module "path\to\mod_was_ap22_http.so"  
WebSpherePluginConfig "path\to\plugin-cfg.xml"
```

11. Use one of the following methods to merge the plugin-cfg.xml files for all the Liberty servers in the cluster.

- Manually merge the files using a text editor.
- Use the job manager to submit a **Generate merged plugin configuration for Liberty servers** job.

For more information about the job manager, see [http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.nd.multiplatform.doc%2Fae%2Ftagt\\_jobmgr\\_liberty\\_plugin\\_merge.html](http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.nd.multiplatform.doc%2Fae%2Ftagt_jobmgr_liberty_plugin_merge.html).

12. Verify that workloads are distributed to multiple Liberty servers via the IBM HTTP Server and Web Server Plugins.

- a. Ensure that session affinity is enabled. To do so, check that a CloneID attribute is included for each server in the plugin-cfg.xml file of the IBM HTTP Server and Web Server Plugins. Note that you can generate CloneID values automatically or specify particular strings. The following example shows CloneID attributes specified for three servers:

```
<ServerCluster CloneSeparatorChange="false" GetDWLMTable="false" IgnoreAffinityRequests="true"  
  <Server CloneID="s59" ConnectTimeout="0" ExtendedHandshake="false" MaxConnections="-1" Name="s59"  
    <Transport Hostname="wasvm59.example.com" Port="9080" Protocol="http"/>  
  </Server>  
  <Server CloneID="s56" ConnectTimeout="0" ExtendedHandshake="false" MaxConnections="-1" Name="s56"  
    <Transport Hostname="wasvm56.example.com" Port="9080" Protocol="http"/>  
  </Server>  
  <Server CloneID="vm28" ConnectTimeout="0" ExtendedHandshake="false" MaxConnections="-1" Name="vm28"  
    <Transport Hostname="wasvm28.example.com" Port="9080" Protocol="http"/>  
  </Server>  
  <PrimaryServers>  
    <Server Name="default_node_String1"/>  
    <Server Name="default_node_String2"/>  
    <Server Name="default_node_String3"/>  
  </PrimaryServers>  
</ServerCluster>
```

- b. Ensure that each Liberty server is started.
- c. Verify that round-robin load-balancing is successfully routing application requests to each of the backend Liberty servers.

## Troubleshooting IBM HTTP Server startup

Problems starting the IBM HTTP Server during deployment of a Worklight Server on a WebSphere Application Server Liberty Profile farm might be caused by an exception in the runtime library.

### About this task

While setting up IBM Worklight on a WebSphere Application Server Liberty Profile farm, you are instructed to start the IBM HTTP Server by running the following command:

```
/opt/HTTPServer/bin/httpd -d /opt/HTTPServer -k start -f /opt/HTTPServer/conf/httpd.conf
```



If the attempt fails with the following message, the problem might be due to attempting to start IBM HTTP Server outside a WebSphere Application Server environment in which certain libraries cannot be found.

```
/opt/HTTPServer/bin/httpd: error while loading shared libraries: libexpat.so.0: cannot open shared
```

If a message similar to this is displayed, use the following procedure to make the required libraries available.

## Procedure

1. Check the IBM HTTP Server libraries:

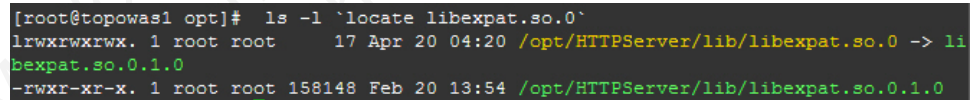
```
ldd /opt/HTTPServer/bin/httpd
```

The output shows that `libexpat.so.0` cannot be found:

```
linux-vdso.so.1 => (0x00007fff8c9d3000)
libm.so.6 => /lib64/libm.so.6 (0x00000039fb000000)
libaprutil-1.so.0 => /usr/lib64/libaprutil-1.so.0 (0x00007fc371a7d000)
librt.so.1 => /lib64/librt.so.1 (0x00000039fac00000)
libcrypt.so.1 => /lib64/libcrypt.so.1 (0x0000003a07c00000)
libpthread.so.0 => /lib64/libpthread.so.0 (0x00000039fa800000)
libdl.so.2 => /lib64/libdl.so.2 (0x00000039fa000000)
libexpat.so.0 => not found
libapr-1.so.0 => /usr/lib64/libapr-1.so.0 (0x00007fc37184f000)
libc.so.6 => /lib64/libc.so.6 (0x00000039fa400000)
libuuid.so.1 => /lib64/libuuid.so.1 (0x0000003a04c00000)
libexpat.so.1 => /lib64/libexpat.so.1 (0x00000039ff400000)
libdb-4.7.so => /lib64/libdb-4.7.so (0x00000039fd800000)
/lib64/ld-linux-x86-64.so.2 (0x00000039f9c00000)
libfreebl3.so => /lib64/libfreebl3.so (0x0000003a08000000)
```

2. Find the library on the file system.

```
ls -l `locate libexpat.so.0`
```



```
[root@topowas1 opt]# ls -l `locate libexpat.so.0`
lrwxrwxrwx. 1 root root 17 Apr 20 04:20 /opt/HTTPServer/lib/libexpat.so.0 -> li
bexpat.so.0.1.0
-rwxr-xr-x. 1 root root 158148 Feb 20 13:54 /opt/HTTPServer/lib/libexpat.so.0.1.0
```

3. Check `/etc/ld.so.conf`.

```
cat /etc/ld.so.conf
```

The output shows it includes all conf files under `/etc/ld.so.conf.d/`.

```
include ld.so.conf.d/*.conf
```

4. Add the IBM HTTP Server library to the configuration.

- a. `cd /etc/ld.so.conf.d/`

- b. Add the `http` library to the system configuration. The location of the IBM HTTP Server `lib` is shown above.

```
echo /opt/HTTPServer/lib > httpd-lib.conf
```

- c. Remove the `ldd` cache.

```
rm /etc/ld.so.cache
```

- d. Reload the `ldd` configuration.

```
/sbin/ldconfig
```

5. Check the IBM HTTP Server libraries again:

```
ldd /opt/HTTPServer/bin/httpd
```

The output shows `libexpat.so.0` is available:

```

linux-vdso.so.1 => (0x00007fffd594a000)
libm.so.6 => /lib64/libm.so.6 (0x00000039fb000000)
libaprutil-1.so.0 => /opt/HTTPServer/lib/libaprutil-1.so.0 (0x00007f20474bf000)
librt.so.1 => /lib64/librt.so.1 (0x00000039fac00000)
libcrypt.so.1 => /lib64/libcrypt.so.1 (0x0000003a07c00000)
libpthread.so.0 => /lib64/libpthread.so.0 (0x00000039fa800000)
libdl.so.2 => /lib64/libdl.so.2 (0x00000039fa000000)
libexpat.so.0 => /opt/HTTPServer/lib/libexpat.so.0 (0x00007f204739c000)
libapr-1.so.0 => /opt/HTTPServer/lib/libapr-1.so.0 (0x00007f2047271000)
libc.so.6 => /lib64/libc.so.6 (0x00000039fa400000)
/lib64/ld-linux-x86-64.so.2 (0x00000039f9c00000)
libfreeb13.so => /lib64/libfreeb13.so (0x0000003a08000000)

```

6. Start the IBM HTTP Server.

---

## Installing and configuring the IBM WebSphere Analytics Platform

To run the analytics features, you must install the IBM WebSphere Analytics Platform (IWAP).

### About this task

The operational analytics feature requires installation of the IBM WebSphere Analytics Platform component, which is included in the IBM Worklight server installation. The best practice is to install this component on a separate system than your IBM Worklight server and IBM Application Center server to offload the necessary storage and analytics workload from these critical IBM Worklight production systems. Installation requires extracting a file and running an interactive shell script that is provided in the compressed file.

System Requirements:

- Red Hat Enterprise Linux 6.x (64-bit).
- 200 MB of disk space is required for the installation.
- 4 GB of RAM is required (8 GB of RAM is recommended).
- Local file system with at least 100 GB of disk space.
- Python 2.7.x-2.8.x.
- Root access for installation.
- Ability to open firewall ports.

Find the compressed file that contains the Analytics Engine at:

- For UNIX systems: /opt/IBM/Worklight/analytics.zip (or your installation location, if different)
- For Windows systems: c:\Program Files (x86)\IBM\Worklight\analytics.zip (or your installation location, if different)

### Procedure

1. Copy the analytics.zip file to the RHEL 64-bit system that you designated as the host for the IBM WebSphere Analytics Platform.
2. Extract the analytics.zip file.
3. Run the interactive shell script, which prompts for input such as installation paths.
  - a. Run ./setup.sh -t For instructions on how to use the script, use -h option
  - b. The first prompt asks the user to specify the mode of installation of the IBM WebSphere Analytics Platform. Specify the mode of installation and press ENTER. There are three different modes supported:

### Stand-alone

A stand-alone installation installs and configures the Analytics features and a search node in which data is stored and indexed. The search node is configured to be a master search node. Each search cluster has a single master node and more search nodes can be added to the cluster by specifying the same cluster name during the installation. One or more stand-alone and search installations with the same cluster name form a cluster of search nodes.

### Search

A search installation installs and configures a search node (master).

### Console

A console mode installation installs and configures the Analytics feature and a search node that is configured as a client. By configuring as a client, the node does not store any data locally. The node becomes part of the cluster (the name of the cluster is provided during installation) and the operations are redirected to the search node in the cluster that contains the relevant data

- c. Specify the location to install the analytics. If none is specified, the default location of `/opt/IBM/analytics` is used.
  - d. Specify the location to store data. If none is specified, the default location `<INSTALL_LOCATION>/data` is used.
  - e. Specify the location to store logs. If none is specified, the default location of `<INSTALL_LOCATION>/logs` is used.
  - f. Specify an available port for analytics. If none is specified, the default port 80 is used.
4. Note the installation summary (Analytics page URL, Instructions to start / stop / restart / uninstall analytics).
  5. Open firewall ports if necessary.
    - a. The port that you designated for analytics during installation.
    - b. 9500: For HTTP communication between search nodes in a cluster.
    - c. 9600: For discovery of search nodes in a cluster / unicast.
  6. Start the analytics feature by running `service analytics start`.
  7. Verify the successful completion of the installation by visiting the analytics tab in the IBM Worklight console in a web browser.

During the installation this summary appears, providing instructions about how to start, stop, or uninstall IWAP:

IBM Worklight Analytics Installation Summary

-----  
Worklight analytics console can be accessed using URL:  
`http://localhost:80/iwap/worklight/v1/index.html`

Use the below to start / stop / restart analytics.

```
service analytics start
service analytics stop
service analytics restart
```

Note: The port 9300 should be open for communication between search nodes. The ports 9500 & 9600 should always be reserved for use by analytics

Uninstall: To uninstall analytics,  
use `/usr/sbin/analyticsuninstall.sh`

### Related concepts:

“Operational analytics” on page 840

The operational analytics feature enables searching across apps, services, devices, and other sources to collect data about usage or detect problems.

## Configuring Worklight Server for analytics

To use the analytics feature, you must configure Worklight Server to forward analytics data to the IBM WebSphere Analytics Platform.

### Before you begin

To follow these instructions you must know how to create an IBM Worklight project, generate a WAR file from this project, and install it in Worklight Server. For more information, see the Getting Started module *Creating your first Worklight application* under category 2, *Hello Worklight*, in Chapter 3, “Tutorials and samples,” on page 25.

### About this task

To enable the Worklight Server WAR to forward data to the IBM WebSphere Analytics Platform, you must specify properties in the JNDI configuration for the WAR file that you deploy to the server. Starting with the URL that you noted during installation of the IBM WebSphere Analytics Platform, modify the JNDI configuration by completing these steps.

### Procedure

1. Find and modify the URL that you noted during the IBM WebSphere Analytics Platform installation. For example, if the URL that you noted is `http://localhost:80/iwap/worklight/v1/index.html`, you must enter `http://<IP>:80/iwap/v1/events/_bulk` as the `wl.analytics.url` property. The URL that is used for the IBM WebSphere Analytics Platform must have the same connectivity (public Internet or internal intranet) and use the same protocol (HTTP or HTTPS) as the Worklight Server.
2. Add the following value to the JNDI configuration: `wl.analytics.url=[the modified URL that you noted during installation of IBM WebSphere Analytics Platform]`. For more information about JNDI properties, see “Configuring an IBM Worklight project in production by using JNDI environment entries” on page 723.

When this WAR is installed and the JNDI properties are properly configured, data flows to the IBM WebSphere Analytics Platform. When the `wl.analytics.url` property is set, a tab in the IBM Worklight Console displays the Analytics Dashboard view. The presence of the Analytics tab confirms that you properly configured your Worklight Server to forward analytics data to the IBM WebSphere Analytics Platform. The user interface for the Worklight Console Analytics tab is rendered within an IFRAME that communicates directly with the IBM WebSphere Analytics Platform. The browser from which the analytics information is accessed must have direct connectivity to the analytics platform, just as it does with the Worklight Server.

**Note:** When `wl.analytics.*` JNDI properties are changed, you must restart the server for the changes to take effect.

## What to do next

For more information about operational analytics, see “Operational analytics” on page 840.

---

## 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.

### Related concepts:

“Troubleshooting analytics” on page 858

Find solutions to problems with IBM Worklight analytics features.

## Troubleshooting to find the cause of installation failure

You can troubleshoot to find the cause of installation failure.

### About this task

If installation failed but the cause is not obvious, you can troubleshoot by completing the following procedure:

### Procedure

See the failed-install.log file in the installation directory or, if this file does not exist, 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 “Deploying a project WAR file and configuring the application server manually” on page 710.

## Troubleshooting failure 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:

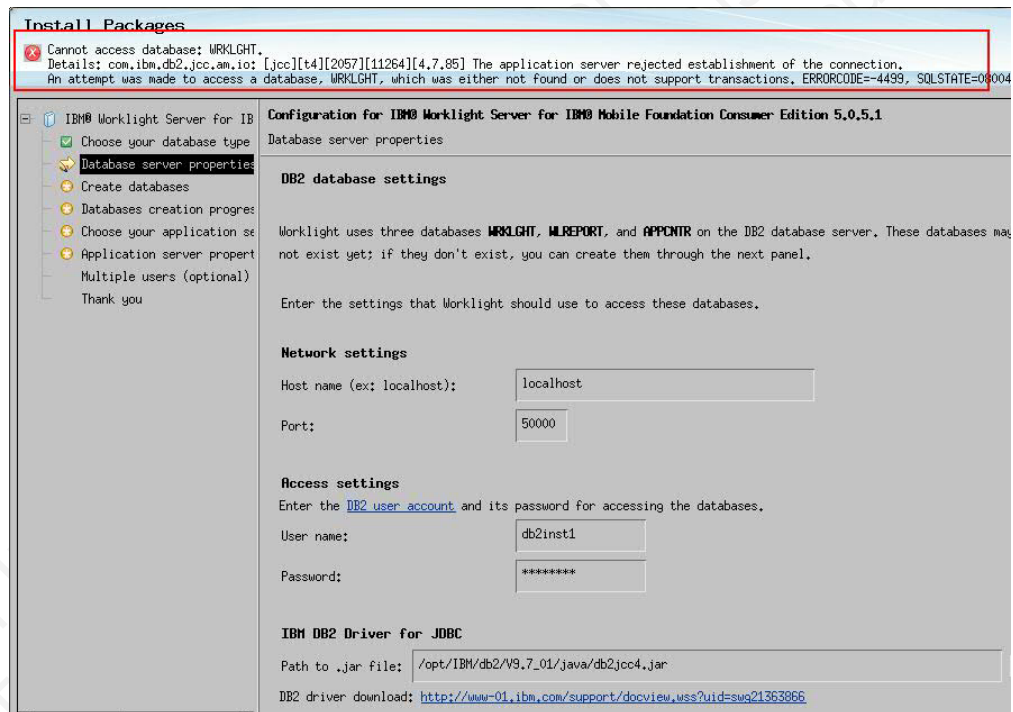
<http://pic.dhe.ibm.com/infocenter/db2luw/v10r1/index.jsp?topic=%2Fcom.ibm.db2.luw.messages.sql.doc%2Fdoc%2Fmsql0101035n.html>

## Troubleshooting an installation blocked by DB2 connection errors

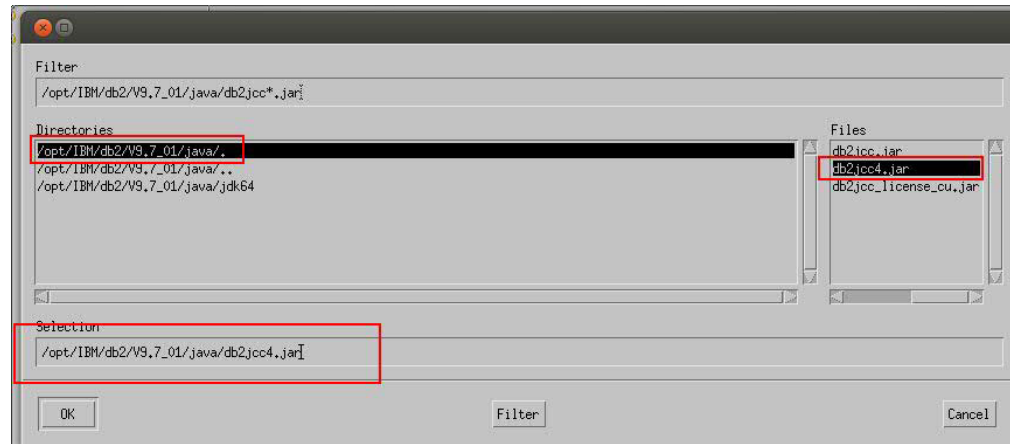
An incorrect DB2 JDBC driver can prevent connection to the database, and block the Worklight Server installation.

### About this task

During installation, the 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. An error similar to the following appears:



This error may be caused by an incorrect version of the DB2 JDBC driver. For example, you may have chosen the DB2 JDBC server that is included in a DB2 release:



## Procedure

1. If you receive this error, verify the current DB2 JDBC 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.



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

---

## Chapter 5. Upgrading from one version of IBM Worklight to another

You must take several actions when you upgrade from one version of IBM Worklight to another.

### About this task

Upgrading from one version of IBM Worklight to another involves upgrading the software, upgrading your database, if needed, and upgrading your apps. Most of this upgrade is automatically done for you when you use the Installation Manager installer. However, this upgrade might also involve some manual operations, such as setting various properties, using special command facilities and ANT tasks. The complete upgrade procedure is detailed in the following topics.

---

### Migrating from IBM Worklight V5.0.6 to V6.0.0

When you open your IBM Worklight project with Worklight Studio V6.0.0, 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 V5.0.6 to V6.0.0. If you upgrade from earlier versions, see also “Migrating from IBM Worklight V5.0.5 to V5.0.6” on page 224, and “Migrating from IBM Worklight V5.0.0.3 to V5.0.5” on page 229.

#### Worklight Studio

Depending on the state of your current development environment, the instructions vary for how to migrate to Worklight Studio V6.0.0:

- To migrate an existing Worklight Studio instance and its associated projects to V6.0.0, see “Migrating Worklight Studio to V6.0.0” on page 189.
- To migrate existing projects to a new instance of Worklight Studio, see “Migrating projects to a new Worklight Studio instance” on page 190.

#### Worklight Server

Within Worklight Studio, when Worklight Server restarts after an upgrade, it checks database table `WORKLIGHT_VERSION` to verify that the database schema is consistent with the new version of Worklight Server. If the IBM Worklight database schema changed in the new release, and the schema is found to be inconsistent with the new version, or empty, the existing schema is dropped. It is then re-created to be consistent with the new release, updating the `WORKLIGHT_VERSION` table appropriately.

This action deletes all data in your existing IBM Worklight development database. This deletion does not cause problems because only the tables that are related to IBM Worklight and its reports are dropped and re-created.

**Important:** If you upgrade Worklight Server to V6.0.0 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 Worklight Server in a production environment” on page 191.

## Usage of existing applications

- **Usage of existing applications with a new server version:** if you want to use existing applications with a new server version, see “Migrate your Worklight projects” on page 192.
- **Migration of projects using Tealeaf® libraries:** If you added Tealeaf libraries to iOS or Android projects that use IBM Worklight V5.0.6 or earlier, then those projects cannot be migrated automatically. Therefore, those applications must be manually upgraded into a new Worklight V6.0.0 project.

- You must manually remove the Tealeaf library and the configuration files from your application before you import your project into an IBM Worklight V6.0.0 workspace.
- You must add the following lines of code manually to the `/common/js/initOptions.js` file:

```
logger : {
  enabled: true,
  level: 'debug',
  stringify: true,
  tag: {
    level: false,
    pkg: true
  },
  whitelist: [],
  blacklist: []
},
analytics : {
  enabled: true
  //url : //
}
```

**Note:** You must also update the JNDI configuration for your WAR to set the other required properties, as explained in “Configuring Worklight Server for analytics” on page 176.

- **Usage of WL.App.close API:** starting with IBM Worklight V6.0.0, the WL.App.close API is deprecated to reflect a change in the iOS Human Interface Guidelines. Consider no longer using WL.App.close API in your apps. For more information, see “WL.App.close” on page 497

## Third-party libraries

**Cordova:** IBM Worklight V6.0.0 is now based on Cordova 2.6. Cordova 2.6 includes new, modified, deprecated, and removed items, compared to the earlier version. The upgrade process for the Cordova 2.6 configuration is automated when the IBM Worklight project is built in Worklight Studio or with the Ant tasks.

- **New items (for iOS only)**
  - The `AssetLibrary.frameworks` is added as a project resource.
  - In the `config.xml` file, the following attributes are added to the `<preference>` element:
    - **DisallowOverscroll**
    - **FadeSplashScreen**

- **FadeSplashScreenDuration**
  - **HideKeyboardFormAccessoryBar**
  - **KeyboardShrinksView**
  - **Modified item** For iOS, the config.xml root element is now <widget> instead of <cordova>.
  - **Deprecated items**
    - For iOS, the cellular network connection return of Connection.CELL\_2G is deprecated in Cordova 2.6. It may change and return Connection.CELL in a future release.
    - The **EnableLocation** preference in config.xml is deprecated in Cordova 2.6. Instead, use the **onload** attribute of the element.
- To know more about deprecation in Cordova, see <http://wiki.apache.org/cordova/DeprecationPolicy>.
- **Removed items** For iOS, the following attributes were removed from the <preference> element in the config.xml file:
    - **UIWebViewBounce**
    - **OpenAllWhiteListURLsInWebView**

**Worklight Dojo Library Project Setup:** Worklight V6.0.0 project setup changed for Worklight Dojo projects. A Worklight project that uses Dojo is now paired with a Dojo library project. Projects that were created with an earlier version of IBM Worklight had a mobile version of Dojo that was directly placed in the project. Worklight projects that are created with IBM Worklight V6.0.0 now have a small subset of Dojo resources inside the Worklight project, and a separate Dojo library project. To know how to migrate an earlier project to use the Dojo library, see “Migrating an IBM Worklight project to use the Dojo library” on page 184.

## Changes in projects

**Removed environments:** IBM Worklight no longer supports the following environments:

- iGoogle
- Facebook
- Apple OS X Dashboard
- Vista

For more information about these environments, see IBM Worklight V5.0.6 Information Center. If you use IBM Worklight applications that include the Facebook environment, you can migrate these apps to the Desktop Browser environment. For more information, see “Manually migrating Facebook apps” on page 188.

**Custom Cordova plug-ins for Windows Phone 8 applications:** if you wrote your own Cordova plug-ins in a Windows Phone 8 application, you must declare them in the config.xml file under the <plugin> element as follows:

```
<widget>
  <plugins>
    <!--Cordova Plugins-->
    <!--Worklight Plugins-->
    <plugin name=your plugin/>
  </plugins>
</widget>
```

## Migrating an IBM Worklight project to use the Dojo library

You can migrate a Dojo project that was created with an earlier version of IBM Worklight to use the new Dojo library project.

### About this task

A project that was created with an earlier version of Worklight Studio had a mobile distribution and a `build-dojox.xml` file that controlled which pieces of Dojo were built into the IBM Worklight application, as shown in Figure 14.

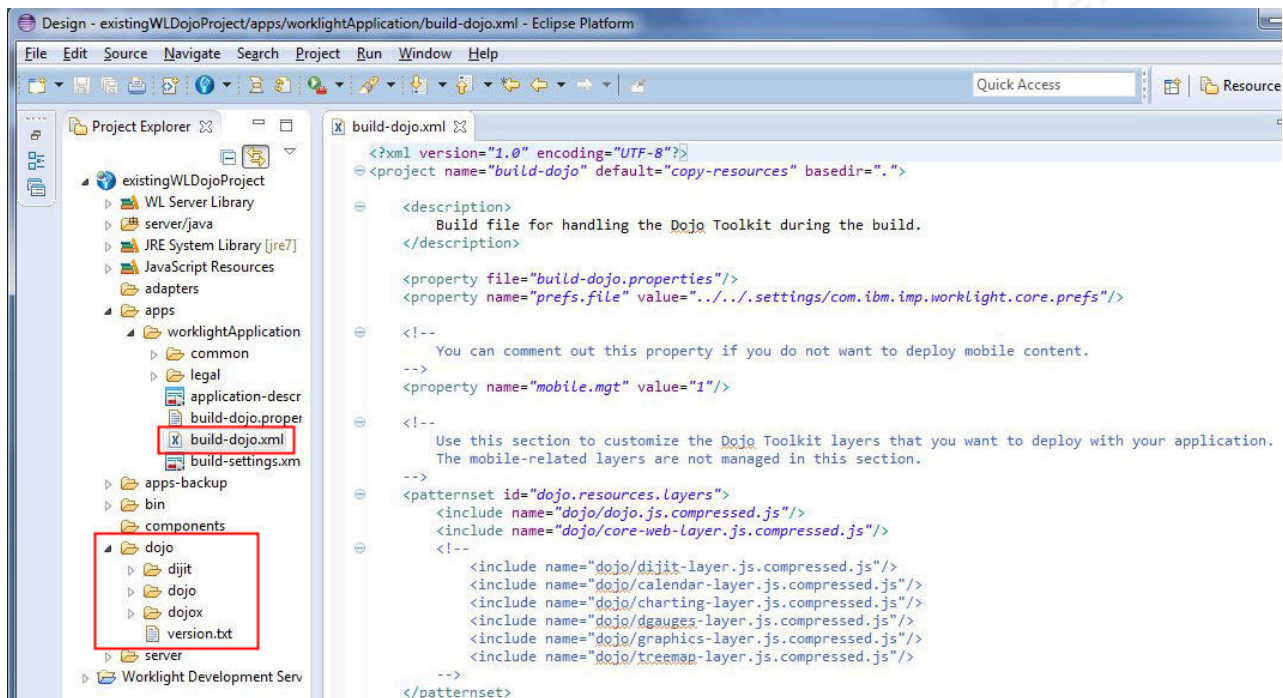


Figure 14. Structure of a Dojo project made with IBM Worklight V5.0.6 or earlier

Starting in Worklight Studio V6.0.0, projects that require Dojo are set up using a Dojo library project (see “Worklight Dojo library project setup” on page 281). To convert your existing Worklight Dojo project to a Worklight Dojo Library project, complete the following procedure.

### Procedure

1. Back up the project you want to migrate in case you want to roll back the migration.
2. Right-click your IBM Worklight project, and click **Properties**.
3. In the Properties window, click **Project Facets**.
4. Clear the **Dojo Toolkit** check box.

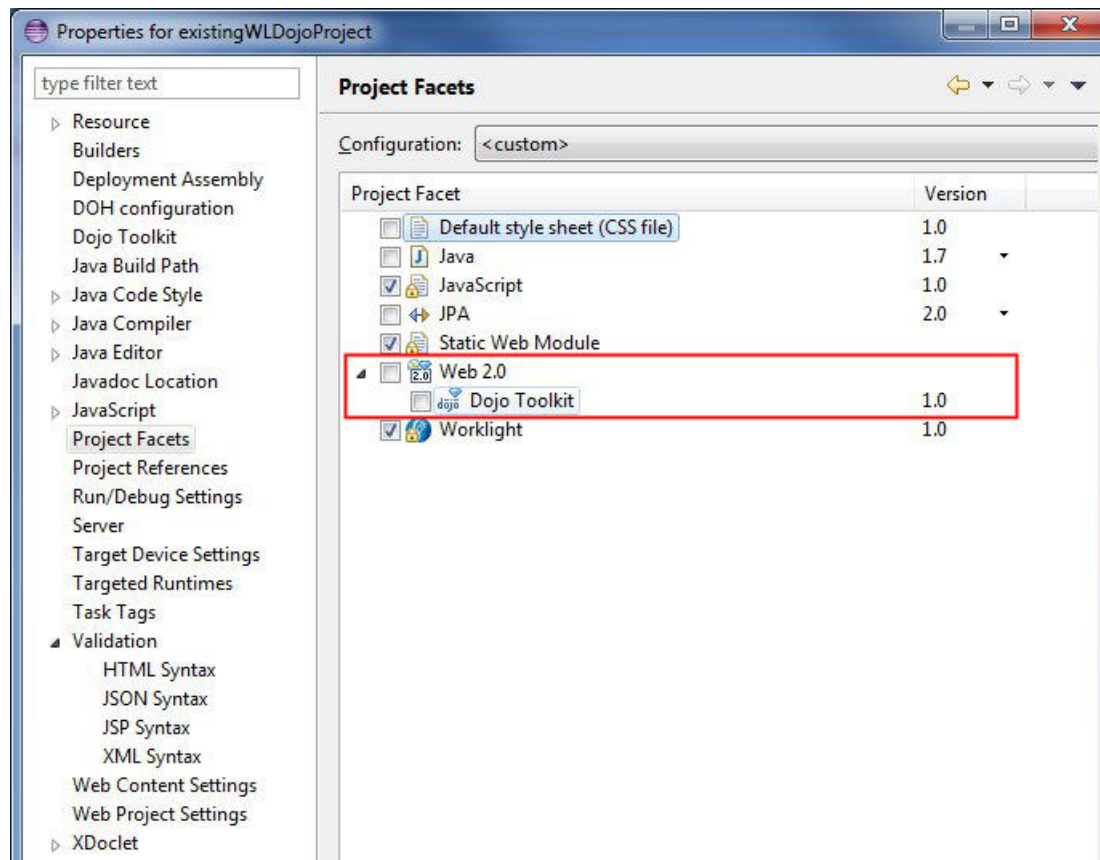


Figure 15. Project Facets window

5. Click **OK**.

Nothing is removed from your project. You must now create a placeholder application in your existing project to reinstall Dojo.

6. To create a placeholder application, go to **File > New > Worklight Hybrid Application**.
7. In the field **Application name**, set the name of your application, and select **Add Dojo Toolkit**.



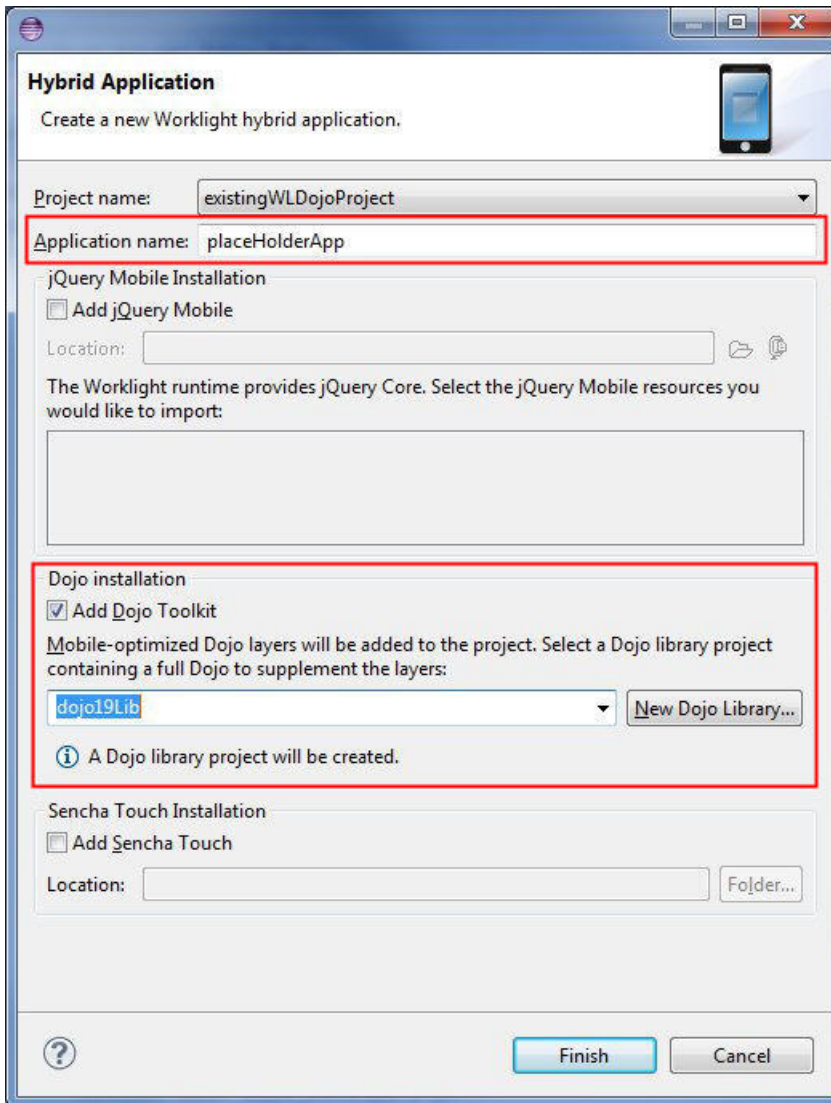


Figure 16. Adding Dojo toolkit to an application

8. Name and configure the Dojo Library. For more information, see “Worklight Dojo library project setup” on page 281.

9. Click **Finish**.

Now, there is a new Dojo Library Project in your workspace. There is also a new www folder, and your placeholder application is created. You must now migrate each existing application. Figure 17 on page 187 shows:

- The new Dojo library project
- The newly created placeholder application
- The mobile Dojo distribution of the earlier application (the dojo folder)
- The new Dojo layers (the www folder)



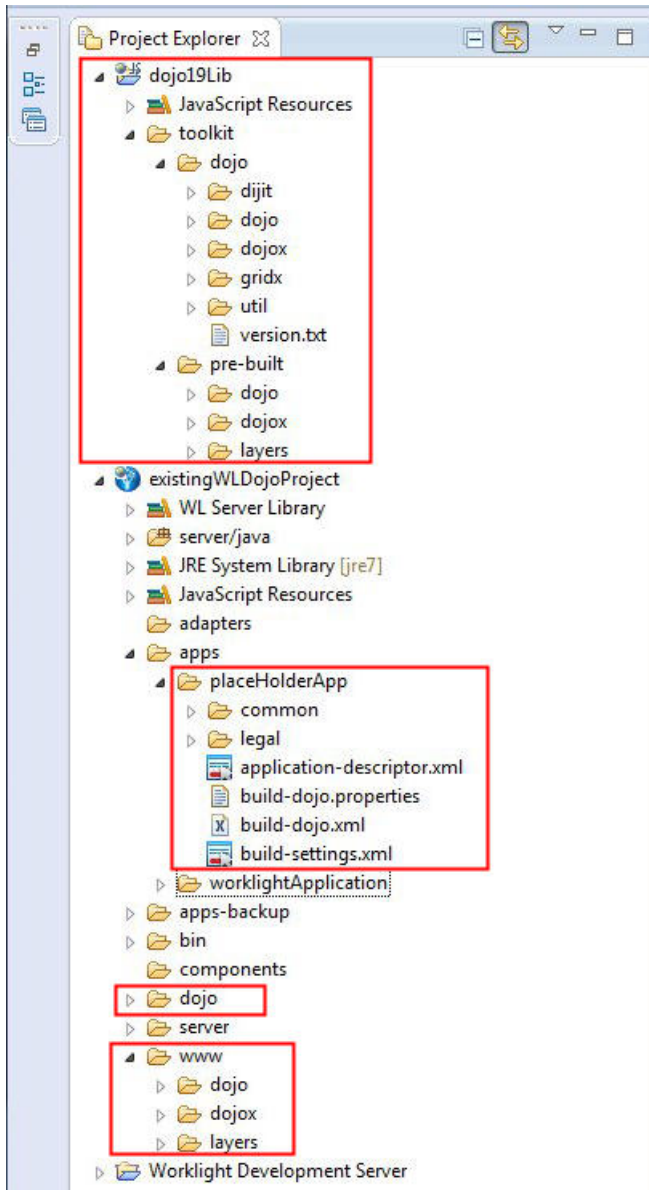


Figure 17. New application in an existing Dojo project

- Copy the build-dojo.xml and build-dojo.properties files from the placeholder application to the application you want to migrate.

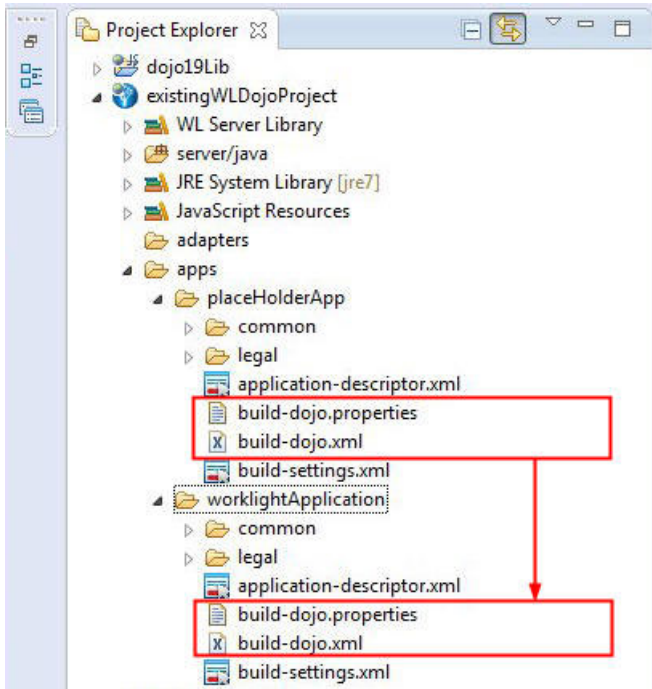


Figure 18. Copy of the *build-dojo.xml* and *build-dojo.properties* files

The build of Worklight Studio then picks up the new Dojo files.

11. If you did not add custom Dojo code to the folder *Worklight Project Name/dojo*, you can delete it. Otherwise, copy the Dojo code that you want to keep to the *www* folder.

### What to do next

Repeat this procedure for every application you want to migrate.

When the *build-dojo.xml* and the *build-dojo.properties* files are copied to every application you want to migrate, and when the custom Dojo code (if needed) is copied from the *dojo* folder to the *www* folder, the migration process is complete. You can delete the placeholder application.

## Manually migrating Facebook apps

You can migrate from a Facebook environment to a Desktop Browser environment by copying your Facebook files and folders to the Desktop Browser folder.

### Procedure

1. Create a **Desktop Browser web page** environment for your app.
2. Delete all the files and folders under this new environment.
3. Copy all the files and folders from the **facebook** environment folder to the **desktopbrowser** environment folder.
4. In the Facebook dashboard, change the field entry for the canvas URL from `http://host:port/apps/services/www/application_name/facebook/` to `http://host:port/apps/services/www/application_name/desktopbrowser/`.
5. Change the field entry for the secure canvas URL from `https://host:port/apps/services/www/application_name/facebook/` to `https://host:port/apps/services/www/application_name/desktopbrowser/`.

6. In the .html and .js files of the **desktopbrowser** environment folder, search for any occurrence of the string `http://host:port/apps/services/www/application_name/facebook/` and replace it with `http://host:port/apps/services/www/application_name/desktopbrowser/`.

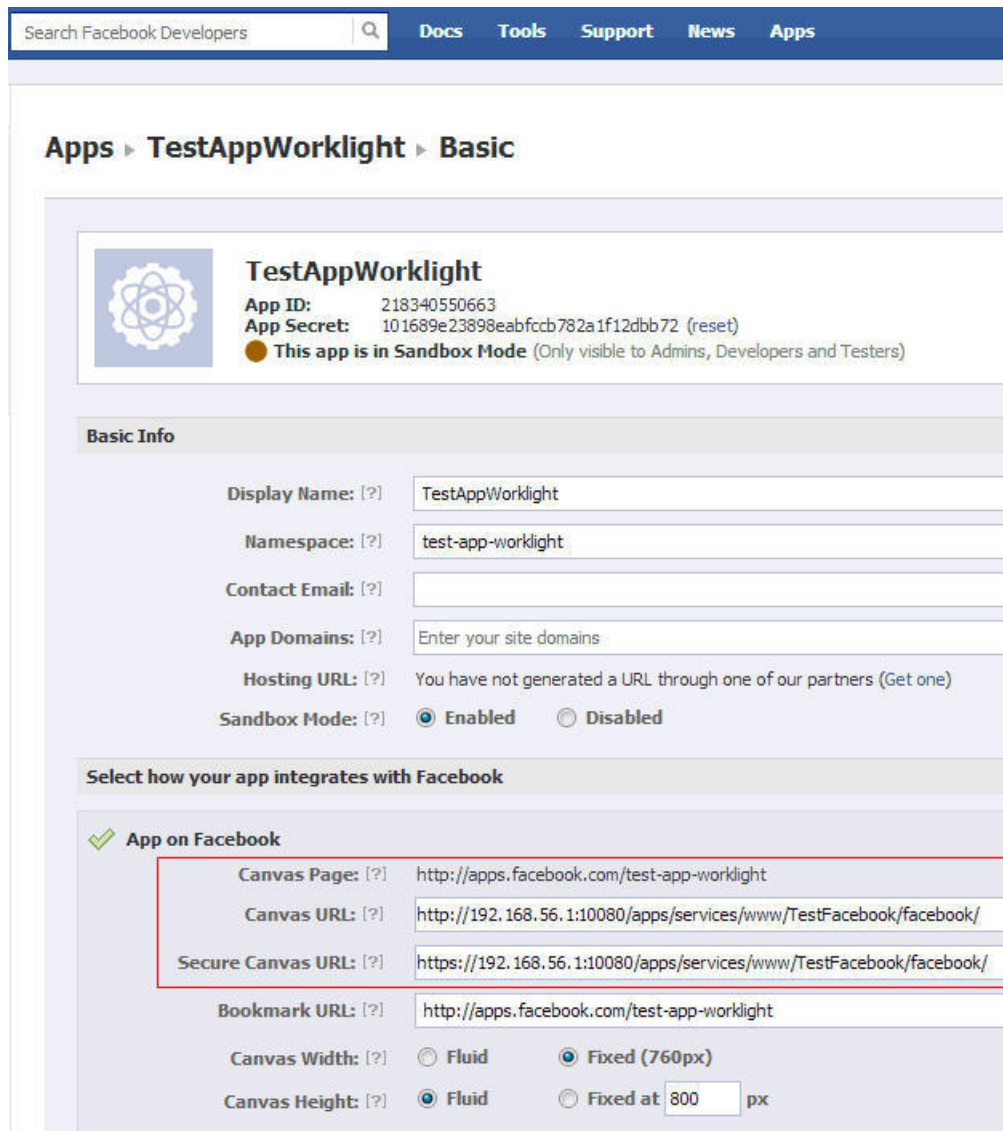


Figure 19. Facebook dashboard

7. Rebuild and deploy the new environment.

## Migrating Worklight Studio to V6.0.0

Worklight Studio V6.0.0 must be installed into Eclipse V4.2.2 (Juno) as a new installation. It cannot be upgraded, nor installed on top of an earlier version of Eclipse.

### About this task

Worklight Studio cannot be directly upgraded to V6.0.0 from earlier versions in a single update installation. If you want to install Worklight Studio V6.0.0 into the Eclipse instance where your current version of Worklight Studio is installed, you

must first uninstall your current version of Worklight Studio, and then upgrade Eclipse to the supported version. When Worklight Studio V6.0.0 is installed, you can then point to your earlier workspace and work with your existing projects.

## Procedure

1. Optional: Uninstall your existing instance of Worklight Studio (only if you install Worklight Studio V6.0.0 into the same Eclipse as the one where you have your current version of Worklight Studio).
  - a. Open the **Help** menu and click **About Eclipse**.
  - b. Click **Installation Details**.
  - c. Select all of the following if they are installed: **IBM Dojo Mobile Tools**, **IBM jQuery Mobile Tools**, **IBM Worklight Studio**.
  - d. Click **Uninstall** and follow the instructions to complete the uninstallation.
2. Upgrade Eclipse to the minimum supported version (V4.2.2). If you already have it, move to the next step.
  - a. Open the **Window** menu and click **Preferences**.
  - b. In the left panel, click **Install/Update** to display more options.
  - c. Click **Available Software Sites**, and click **Add**.
  - d. In the Add Site window, enter a value in the **Name** field (for example, *Eclipse Update*).
  - e. Enter `http://download.eclipse.org/releases/juno/` into the **Location** field and click **OK**.
  - f. Click **OK** to exit the Preferences window.
  - g. Open the **Help** menu and click **Check for Updates**.
  - h. In the Available Updates window, select the items that you want to install or update.
  - i. Click **Next** and follow the instructions to complete the installation.
3. Install Worklight Studio V6.0.0. For instructions on Worklight Studio installation, see “Installing Worklight Studio” on page 35.

## Results

Worklight Studio is now 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 solve the problem:

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

## Migrating projects to a new Worklight Studio instance

Follow these instructions if you want to migrate projects that were created with IBM Worklight V5.0.6 or earlier to Worklight Studio V6.0.0.

## Procedure

1. Export your projects from your current instance of Worklight Studio.
  - a. Open the **File** menu and click **Export...**
  - b. Expand the **General** menu, click **Archive File** and click **Next**.
  - c. Browse to the destination for the archive file and click **Finish**.

2. Install Worklight Studio V6.0.0. For instructions on Worklight Studio installation, see “Installing Worklight Studio” on page 35.
3. Import the projects that you exported in Step 1.
  - a. In Eclipse, open the **File** menu and click **Import...**
  - b. In the Import window, click **General** to expand more options.
  - c. Click **Archive File**, and click **Next**.
  - d. Browse to each project archive file and click **Finish**.

## Results

You can now find your existing projects in your new instance of Worklight Studio.

### 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 solve the problem:

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

## Upgrading Worklight Server in a production environment

Upgrading Worklight Server 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.x to V6.0.x in a production environment, the process can be more complicated than upgrading to a new version in your development environment. The upgrade procedure can also take longer if you have existing IBM Worklight applications that run in a production Worklight Server environment. For step-by-step instructions on how to upgrade your production Worklight Server to V6.0.x, see the following topics.

**Note:** The documentation in these topics assumes that:

- Your database type is IBM DB2, MySQL, or Oracle (not Apache Derby).
- Your application server type is WebSphere Application Server Full Profile, WebSphere Application Server Liberty Profile, or Apache Tomcat.

**Important:** The topics are in a specific order, and must be completed in the order shown.

The tasks under the “Prepare for the upgrade process” on page 192 topic can be completed before the actual installation of the new Worklight Server. The following topics, after “Start the upgrade process” on page 197, must be completed sequentially and in one session until you complete the full procedure. Naturally, the final topic, “Recovering from an unsuccessful upgrade” on page 224, must be performed only if the upgrade was not successful.

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.

The topics provide 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 preparatory 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 IBM Worklight applications.

Read the entire set of topics before you begin the actual upgrade process to become familiar with the tasks ahead of you, what must be done, and in what order.

## Prepare for the upgrade process

Several preparation tasks must be completed before you begin the actual installation of the new Worklight Server version.

Migrating to a new version of Worklight Server in a development environment is quick and easy, because in most cases no critical data must be preserved in the IBM Worklight databases. In a production environment, however, more time and effort is required for the upgrade, to minimize production downtime and inconvenience to users of existing applications.

The following topics cover preparation tasks to be performed before you begin the installation of the new Worklight Server version. These tasks can be performed at any time prior to the upgrade, but must be completed before you move to the next step, “Start the upgrade process” on page 197.

### Migrate your Worklight projects:

Before you upgrade your production Worklight Server, complete these steps to upgrade your development environment and migrate your existing IBM Worklight projects and apps.

#### About this task

Your projects in Worklight Studio (with their respective apps and adapters) must be migrated to the same version of Worklight Studio you want to install. This task is performed by the development team. It can take some time, and is therefore best started ahead of time, before you begin the next step, “Start the upgrade process” on page 197.

This task is necessary because the project configuration, which is encoded in the project WAR file, must match the IBM Worklight runtime library code (`worklight-jee-library.jar`). If you were to use the project WAR file you created with a previous release of Worklight Studio, the Worklight console code and project configuration would be from a previous version and would not be correct for the new `worklight-jee-library.jar`.

The existing project WAR file (from the previous release) must be replaced with a new project WAR file generated from the new release of Worklight Studio. The new project WAR contains the correct Worklight console code and the correct project configuration.

#### **Important:** Impact for Client Applications

The communication protocol of Worklight Server V6.0.0 supports the protocols of client applications that are built with IBM Worklight V5.0.0.3 or later. Device users who are using apps that were built with IBM Worklight V5.0.0.3 or later, and whose server-side artifacts are successfully ported to Worklight V6.0.0 and tested on a test server, should continue to work without requiring the device users to upload a new version of the application.



However, the Direct Update feature “Direct updates of app versions to mobile devices” on page 755 stops working on those versions. The Direct Update feature works only if the client-side artifacts of the application are built with the same version of Worklight Studio as the Worklight Server that delivers the direct updates. To deliver an update and activate Direct Update, you create an application with an incremented version number and publish it on its application store. To notify your users that a new version of the application is available, you can use the startup display notification feature “Displaying a notification message on application startup” on page 761. If the application update is mandatory, another alternative is to deny access to the old application version “Remotely disabling application connectivity” on page 758.

### Procedure

1. Back up the Worklight Studio workspace that contains the IBM Worklight project.  
When a new version of Worklight Studio is pointed to an older Worklight Studio project, it updates (modifies) some of the metadata files. Therefore, it is a good idea to keep a backup of the previous workspace.
2. Install the new version of Worklight Studio (the version corresponding to the new version of Worklight Server).  
See “Installing Worklight Studio” on page 35 for details.
3. Start Worklight Studio.  
See “Starting Worklight Studio” on page 37 for details.
4. Create a copy of your existing workspace, and then start with a fresh workspace:
  - First, in the old version of Worklight Studio, select the existing Worklight project and click **File > Export > General > Archive** to create an archive of it.
  - Then, in the new version of Worklight Studio, create a new empty workspace, click **File > Import > General > Existing Projects into Workspace**, and select the archive that you created. This also migrates the project to the new version.
5. Rebuild a project WAR file and deploy it to a test environment that contains the new version of Worklight Server.  
See “Deploying IBM Worklight applications to test and production environments” on page 663 for details.
6. Recompile the apps and adapters and deploy them to the same test environment.  
See “Deploying IBM Worklight applications to test and production environments” on page 663 for details.
7. Test the apps and their compatibilities with the client-side artifacts that have not been upgraded.
8. Repeat this process for each of the Worklight Studio projects you want to migrate to the new Worklight Server version.

### What to do next

Prepare the migration of the Application Center.

If you are using Application Center, the size limit for applications that are stored on Application Center with IBM DB2 is 1 GB. If you have applications larger than 1 GB in the Application Center, remove them before you start the upgrade process.



## **New packaging of WebSphere Application Server Liberty Profile and its impact on the Worklight Server upgrade:**

Important information about how WebSphere Application Server Liberty Profile is delivered in IBM Worklight V6.0.0, and how it impacts the upgrade of your production server.

**Important:** The information on this page applies to you if you previously installed Worklight Server version V5.x with the embedded WebSphere Application Server Liberty Profile option.

In Worklight Server V6.0.0, WebSphere Application Server Liberty Core is not embedded in the Installation Manager wizard of Worklight Server. Instead, it is provided as a separate Installation Manager wizard.

As a result, the upgrade process that follows does **not** upgrade your installed version of WebSphere Application Server Liberty Profile, and will not apply fix packs to it in the future. At the end of the upgrade process, your Liberty server remains installed in `<WorklightServerInstallationDirectory>/server/wlp`, but is considered as an external file from the perspective of upgrades, uninstall, and updates from the Installation Manager wizard of Worklight Server.

To prevent this existing server from being uninstalled during the upgrade process, the Installation Manager wizard temporarily renames its directory during the upgrade process. It is critical to apply the steps that are defined in section “Special steps for WebSphere Application Server Liberty Profile” on page 200 before you start the upgrade process. The result of not completing these steps can be a non-functional server.

### **Alternate Method: Migrate the Worklight apps and data to a new Liberty Server**

This alternate upgrade method migrates your Worklight applications and data to a new WebSphere Application Server Liberty Profile server installed by IBM Installation Manager. This server can be updated by IBM Installation Manager when new updates for Liberty are made available.

1. Stop the Liberty server that was installed with the previous version of IBM Worklight.
2. Install WebSphere Application Server Liberty Core with IBM Installation Manager. The installer for IBM WebSphere Application Server Liberty Core is part of the IBM Worklight package and of the IBM Mobile Platform Foundation package.
3. Create a server in this new WebSphere Application Server Liberty Profile installation. If you are not familiar with the creation of a server for Liberty, see the “Overview of the Worklight Server installation process” on page 43.
4. Configure the Liberty server for your production environment.
5. Install IBM Worklight with IBM Installation Manager. In this step, if you request IBM Installation Manager to install Application Center, and if you specify the Application Center databases of the previous version, the Application Center is migrated.
6. Continue to step “Upgrading the databases and deploying the upgraded IBM Worklight project” on page 203.

### **Deciding between in-place upgrade and rolling upgrade:**

There are two ways to perform an upgrade: *in-place upgrade* or *rolling upgrade*.

- An in-place upgrade is an upgrade by which the old version of IBM Worklight is no longer installed after the new version of IBM Worklight has been installed.
- A rolling upgrade is an upgrade that installs the new version of IBM Worklight such that it runs side-by-side with the old version of IBM Worklight in the same application server or in a different application server.

The in-place upgrade is the normal way to upgrade. It has the advantage that it is simpler to perform.

The rolling upgrade has the advantage that it minimizes the downtime of the application server, in case an unexpected problem with the upgrade occurs. If there is a problem, you can restart the application server in the old configuration while investigating the problem.

**Important:** This release supports the rolling upgrade for IBM Worklight projects. However, the rolling upgrade of Application Center is not supported.

This chapter focuses on the in-place upgrade. It mentions specific instructions for rolling upgrade. But the rolling upgrade is complex and not yet fully explained in this documentation.

### **Familiarizing yourself with IBM Installation Manager:**

Before you start the actual installation, verify that you have all the products that you want to install and that you are familiar with IBM Installation Manager procedures.

#### **About this task**

You use IBM Installation Manager to complete the actual upgrade. Before you start, verify that you have all of the necessary installation components, and that you understand the installation procedure.

#### **Procedure**

1. Verify that your hardware and software meet the installation requirements: “Installation prerequisites” on page 42.
2. Make sure that you have the appropriate version of IBM Installation Manager installed on the installation workstation.

Use Installation Manager V1.6.3.1 or later, especially on Windows. For more information about Installation Manager procedures, see the IBM Installation Manager user documentation.

3. Download the repositories that are required for the update from Passport Advantage, or have them available if they are on physical media.

For more information about the types of upgrade repositories available, see “Information about the repositories” on page 196.

4. Verify that the products that you want to update are contained in the Installation Manager repositories.

#### **CAUTION:**

**The following steps are not the actual installation. They are preparatory tasks to ensure that you have everything that is required for the upgrade, so be sure to click Cancel in the last step.**

- a. Start Installation Manager.

- b. Click **File** > **Preferences** > **Repositories** to add references to the repositories that you downloaded and extracted on a local disk, or that you can access through the internet. See Repository preferences for details.
- c. Click **Install**.
- d. Verify that the products list contains everything that you need.
- e. Click **Cancel**. Do not proceed with the installation.

#### Information about the repositories:

Information about the types of repositories that are used by IBM Installation Manager.

#### About this task

There are two types of repositories: base repositories and delta repositories.

- A *base repository* is an installation package that is available on Passport Advantage or on physical media. It is self-contained.
- A *delta repository* is an installation package that is available from FixCentral and is labeled as an *update pack*. It requires a base repository to be functional.

To install a major release (for example, Worklight Server V6.0.0), you need only:

- The base repository V6.0.0 installation package from Passport Advantage or physical media.

To install a fix pack release (for example, Worklight Server V6.0.0.1), you need:

- The base repository (such as Worklight Server V6.0.0) installation package from Passport Advantage or physical media.
- The appropriate V6.0.0.1 installation package from FixCentral.

For a fix pack installation, you must add both repositories to the list known to Installation Manager. Then, in the example given, IBM Installation Manager recognizes the V6.0.0 release as an **Install** choice and the V6.0.0.1 release as an **Update** choice.

To install an interim fix release, you can need up to three repositories:

- The repositories for the release to which the fix applies.
- The repository for the fix.

For installing an interim fix, you must add all these repositories to the list known to IBM Installation Manager. Then IBM Installation Manager recognizes the interim fix as an **Update** choice.

#### Gathering the information you need for the update:

To avoid having to stop the upgrade process to look up required information, gather it in advance and have it handy.

#### About this task

One of the purposes of these instructions is to minimize the time that is required for the Worklight Server upgrade. You do not want to start the procedure and then discover that you are missing some piece of information that is required by the installer.

To avoid this situation, prepare a list of information you are likely to be asked for, and keep it handy during the actual installation procedure.

In addition, it is often necessary to pre-plan certain aspects of the upgrade and clear them with your application server administrator and database administrator. For example, you must know which user name to use when you install the Worklight Server upgrade. Similarly, you must either have sufficient permissions to create or update databases, or have your database administrator do it for you.

### **Procedure**

Go through the following checklist to make sure that you have all of the necessary information to begin the upgrade:

- Make a list of the host names and IP addresses of all servers that must be upgraded.
- Make a similar list of all database names and locations.
- Ensure that the correct JDBC drivers are installed for your target databases.
- The upgrade procedure requires the credentials of the Worklight, Worklight reports, and Application Center databases. Therefore, you must either know the correct schemas, user names, and passwords, or have your database administrator assist you.
- The upgrade procedure requires you to stop and restart the application server and verify its configuration. Therefore, you must be familiar enough with your application server to complete these tasks, or have a system administrator do them.
- For IBM Installation Manager, consult your system administrators and decide whether to run it in administrator mode or in single-user mode. See “Single-user versus multi-user installations” on page 50. This choice can influence how Installation Manager works, and the success or failure of the installation. For example, if you install Worklight Server as root but then try to run it as a different user, problems can arise.

### **Start the upgrade process**

In this phase of the upgrade process, you shut down and back up the application server and IBM Worklight databases and complete other pre-installation tasks.

When you finish the tasks that are listed in “Prepare for the upgrade process” on page 192, you can begin the actual upgrade process.

**Note:** Once you begin this phase of the upgrade process, your Worklight Server, database, and application server are offline. They are no longer available to support existing apps or provide service to existing users of those apps. The upgrade process itself can take several hours. Therefore, you must plan the timing of this process for non-critical hours to have minimal impact on users.

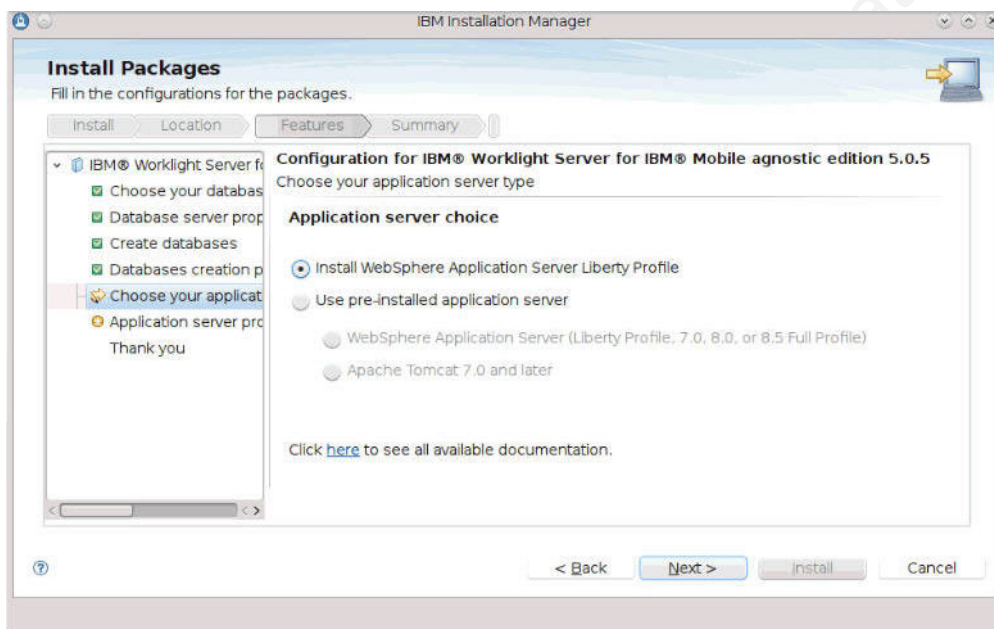
The following topics present the steps, in the order in which they must be completed:

#### **Shutting down the Liberty Application Server:**

If you use the embedded WebSphere Application Server Liberty Profile that can be installed with Worklight Server, you must shut it down.

## About this task

If you installed Worklight Server in your production environment with the embedded WebSphere Application Server Liberty Profile option, you begin the actual upgrade process by shutting down this server. During your original Worklight Server installation, the panel of IBM Installation Manager on which this decision is made looks like the following screen capture:



If you use this embedded server, you must ensure that the Liberty application server is not running before you continue with the upgrade. Specific instructions are shown in the following procedure.

### Procedure

1. Ensure that the `JAVA_HOME` environment variable points to the installation directory of a Java 6 or 7 implementation (JRE or JDK), or that the `PATH` environment variable contains a java program from a Java 6 or 7 implementation.
2. Shut down the server.
  - a. On UNIX, enter the following commands, changing the installation location if necessary:

```
cd /opt/IBM/Worklight
cd server/wlp/bin
./server stop worklightServer
```
  - b. On Windows, enter the following commands, changing the installation location if necessary:

```
cd C:\Program Files (x86)\IBM\Worklight
cd server\wlp\bin
server.bat stop worklightServer
```
3. Verify that no other runaway Liberty server processes are running in the same directory. On Linux and AIX®, you can list such processes with the following command:

```
ps auxww | grep java | grep /wlp/
```

### **Back up the databases:**

Back up the contents of your project databases.

As a safety measure, and to provide a fallback strategy in the case of an unsuccessful upgrade, back up your Worklight Server project databases.

This strategy covers the rare cases in which the new version fails to work correctly in your environment. In this case, you must roll back the upgrade, and return to the previous version of Worklight Server. You must then restore the contents of the databases from the backup.

The reason for this precaution is that the new server version can store its data in a slightly different way than the previous version. There is no assurance that the old server version can operate after the new server version stores data in the databases.

Consult the documentation for your database management system for backup and restore procedures.

### **Back up the Application Server:**

Back up the directory that contains the application server and its configuration.

As an extra safety measure, back up the directory that contains the application server and its configuration. This strategy covers the rare cases in which the new application server version fails to work correctly if errors occur in the forthcoming configuration changes.

If the application server is the embedded Liberty server included with a previous release of Worklight Server, the directories to back up are as follows:

- `WL_INSTALL_DIR/server/wlp`
- On Windows, also back up `C:\ProgramData\IBM\Worklight\WAS851liberty-server\wlp`.

See the documentation for WebSphere Application Server Full Profile or Apache Tomcat to determine the directories to back up for those application servers.

If you later must roll back the upgrade, and return to the previous version of Worklight Server, you must restore the application server. This means, for each of the directories you backed up:

1. Remove or rename the directory or directories to be replaced on disk.
2. Restore the previous contents of the directory or directories, from its respective backup.

### **Test the application server backup**

It is a good idea to test your backup of the application server to make sure you can restart it. For example, for a WebSphere Application Server Liberty Profile backup:

1. Unpack the backup directory.
2. Start the server.



If the server fails to start and load the applications, delete the server's workarea before starting it again. The workarea is the directory `<LibertyInstallDir>/usr/servers/<serverName>/workarea`.

### Applying configuration changes for a rolling upgrade:

When you perform a rolling upgrade and you want to reuse the same application server, and the application server is WebSphere Application Server Liberty Profile or Apache Tomcat, you need to make some specific configuration changes.

### Before you begin

For information about in-place and rolling upgrades, see “Deciding between in-place upgrade and rolling upgrade” on page 194.

### Procedure

1. Shut down the application server.
2. Edit the `server.xml` configuration file by hand to make sure that the old configuration of the IBM Worklight project is preserved. During the next steps, the Worklight Server installer removes from this configuration file the contents between the marker comments:
  - `<!-- Begin of configuration added by IBM Worklight installer. -->`  
`...`  
`<!-- End of configuration added by IBM Worklight installer. -->`
  - For Apache Tomcat:  
`<!-- Begin of Context and Realm configuration added by IBM Worklight installer. -->`  
`...`  
`<!-- End of Context and Realm configuration added by IBM Worklight installer. -->`

and replaces it with contents specific to the new Worklight version. Since you want to preserve the configuration of the Worklight project, you need to move this part of the configuration outside these marker comments.

The declarations that you need to move are:

- The declaration of the Worklight console application.
- The declaration of the data sources named `jdbc/WorklightDS` and `jdbc/WorklightReportsDS`.
- For WebSphere Application Server Liberty Profile, the `<webContainer invokeFlushAfterService="false"/>` element.

### Special steps for WebSphere Application Server Liberty Profile:

If you use WebSphere Application Server Liberty Profile, you must complete these tasks before you continue with the upgrade process.

### About this task

If you installed Worklight Server with the embedded WebSphere Application Server Liberty Profile option, you must stop all servers that were created with this installation of Liberty before you start the upgrade process.

### Procedure

1. Stop the Worklight Server that was created by the Installation Manager wizard of Worklight Server V5.x. In the `<Worklight Installation Directory>/server/wlp/bin` directory, issue the appropriate command:
  - On Windows: `server.bat stop worklightServer`



- On UNIX or Linux: `server stop worklightServer`
2. If you created other server instances, stop them as well by issuing the same commands:
    - On Windows: `server.bat stop <nameOfServer>`
    - On UNIX or Linux: `server stop <nameOfServer>`
  3. Exit from the `<Worklight Installation Directory>/server` directory. For example, by issuing the command `cd ../../../../`.
  4. Stop any other process that uses a file or that can write to files in directory `<Worklight Installation Directory>/server`.

### Verify the ownership of files:

Before you begin the actual installation, check the ownership of all Worklight Server files.

The upcoming “Run IBM Installation Manager to perform the upgrade” step attempts to remove and replace many files in the Worklight Server installation directory. This step can fail if the single-user mode of IBM Installation Manager is used and some of the files or directories are not owned by that user. Therefore, it is useful to guard against this case.

If you installed Worklight Server with the single-user mode of IBM Installation Manager, check whether all files and directories in `WL_INSTALL_DIR` are owned by the current user. For more information about Installation Manager's administrator mode, see Administrator, nonadministrator, and group mode.

On UNIX, you can list the files and directories that do not fulfill this condition with the following commands:

```
cd WL_INSTALL_DIR
find . '!' -user "$USER" -print
```

This command should produce no output.

### Shut down conflicting processes:

Before you start the actual upgrade in the next step, shut down any conflicting Windows processes.

On Windows only, shut down all processes that have their current working directory inside the `WL_INSTALL_DIR` or the `C:\ProgramData\IBM\Worklight\WAS851liberty-server` directory hierarchy. These processes include:

- Console windows
- Windows Explorer windows

You can check for these processes with the Microsoft Sysinternals Process Explorer. For more information, see Process Explorer.

## Run IBM Installation Manager to perform the upgrade

Use IBM Installation Manager to install the new Worklight Server version.

In this step, you use IBM Installation Manager to upgrade your Worklight Server instance.

**Note:** Before you continue, make sure that you completed all of the steps in the “Prepare for the upgrade process” on page 192 and “Start the upgrade process” on page 197 sections that preceded this step.

### **Upgrading on WebSphere Application Server Network Deployment servers:**

Use this procedure to upgrade Worklight Server in a WebSphere Application Server Network Deployment environment.

#### **About this task**

Follow this procedure if you originally installed Worklight Server on an application server of type WebSphere Application Server Network Deployment.

That is, if your original installation was to one or more servers under the control of a deployment manager, and not a single stand-alone server, use the following procedure.

To upgrade stand-alone servers, see “Upgrading on a stand-alone WebSphere Application Server or Apache Tomcat server.”

#### **Procedure**

1. Uninstall Worklight Server on all server nodes that you want to upgrade.
2. Install the new version of Worklight Server. If you want to install the Application Center with IBM Installation Manager, choose:
  - **WebSphere Application Server** as your application server.
  - A deployment manager profile (such as **Dmgr01**) as your profile.
  - Either a **cluster**, a **node**, or the **entire cell** as your scope.For details of this step, see “Installing Worklight Server into WebSphere Application Server Network Deployment” on page 51.
3. Step through the installation wizard.
4. IBM Installation Manager installs on your disk the files and tools that are required to deploy Worklight in your application server. If you requested IBM Installation Manager to install Application Center, it upgrades the existing databases and then deploys Application Center to your application server.
5. When the installation completes, close Installation Manager. Then, complete the steps that are described in “Upgrading the databases and deploying the upgraded IBM Worklight project” on page 203.

### **Upgrading on a stand-alone WebSphere Application Server or Apache Tomcat server:**

Use this procedure to upgrade Worklight Server in a stand-alone WebSphere Application Server or Apache Tomcat environment.

#### **About this task**

If you originally installed Worklight Server on:

- A stand-alone WebSphere Application Server Liberty Profile server,
- A stand-alone WebSphere Application Server Full Profile server, or
- A stand-alone Apache Tomcat server,

use the following procedure, with the IBM Installation Manager **Update** function.

## Procedure

1. Start Installation Manager.
2. Click **Install**. The package name for IBM Worklight Server has changed between Worklight Server V5.x and V6.0.0, so the upgrade must be done with the 'Install' process.
3. If you are doing an in-place upgrade (see “Deciding between in-place upgrade and rolling upgrade” on page 194), select the package group that contains your Worklight Server installation. If you are doing a rolling upgrade, select **Create a new package group**.
4. Step through the installation wizard. If you are doing an in-place upgrade, most choices are disabled (displayed in gray). But you can change the passwords for the database or for WebSphere Application Server access if they are different from the original installation.
5. IBM Installation Manager completes the following tasks:
  - It installs on your disk the files and tools that are required to deploy Worklight in your application server.
  - It undeploys the previous version of Worklight from the Application Server.
  - It removes the application server configurations that were set by the previous installer of IBM Worklight Server.
  - If Application Center was installed in the previous version of Worklight Server, the installer also:
    - Undeploys the previous version of the Application Center from the application server.
    - Upgrades the databases of Application Center to the format used by IBM Worklight V6.0.0. To see a copy of the upgrade scripts, you can install IBM Worklight Server in a new package group and review a copy of the upgrade scripts in `<WorkLightInstallDir>/ApplicationCenter/databases`.
    - Deploys the new version of Application Center to the application server and connects it to the upgraded database.
    - Configure the application server for running the Application Center.
6. When the installation completes, close Installation Manager. Then, complete the steps that are described in “Upgrading the databases and deploying the upgraded IBM Worklight project.”

### Upgrading the databases and deploying the upgraded IBM Worklight project:

After you run IBM Installation Manager, follow these instructions to deploy your IBM Worklight projects.

### Upgrading the databases and deploying the upgraded IBM Worklight project with Ant tasks

After you install IBM Worklight Server with IBM Installation Manager, you must upgrade the databases.

**Note:** This operation cannot be undone. If the upgrade process fails, you must restore the databases from your backup of them to return to a version of the databases compatible with your previous version of Worklight. To then upgrade the restored databases to this new version of Worklight, use the Ant tasks that are described in “Creating and configuring the databases with Ant tasks” on page 669.

After you upgrade the database, you must deploy the upgraded Worklight project that you create in step “Migrate your Worklight projects” on page 192. Review the configuration of the upgraded Worklight project and the properties that are defined in `worklight.properties`.

Finally, to deploy the upgraded Worklight project, use the Ant tasks that are described in “Deploying a project WAR file and configuring the application server with Ant tasks” on page 694. It is possible to override properties of the `worklight.properties` file with this Ant task. The new property values are specified through `<property>` elements in the `<configureapplicationserver>` Ant task invocation.

When these steps are completed, the Worklight Console is available in the application server. In most cases, you must restart the application server before you open the Worklight Console.

### **Upgrading the databases and deploying the upgraded IBM Worklight project manually**

To upgrade the databases manually, follow the steps that are described in “Manually updating the databases.”

To manually deploy the Worklight project that you create in step “Migrate your Worklight projects” on page 192, follow the steps that are defined in “Deploying a project WAR file and configuring the application server with Ant tasks” on page 694.

#### *Manually updating the databases:*

Follow these instructions to manually update the Worklight databases.

If you prefer to update databases manually instead of using the Ant tasks, you must update their sets of tables and columns manually. For example, you can have test or pre-production databases as part of your production environment, each served by a different Worklight database.

Updating these alternative databases is done by running a sequence of database scripts. The upgrade scripts are contained in the just-installed Worklight Server directory.

#### **Scripts for DB2**

For an upgrade from Worklight Server V5.0.0, first upgrade to V5.0.5:

- `WorklightServer/databases/upgrade-worklightreports-50-505-db2.sql` (for WLREPORT)

For an upgrade from Worklight Server V5.0.5 to V5.0.6:

- `WorklightServer/databases/upgrade-worklight-505-506-db2.sql` (for WRKLGHT)
- `ApplicationCenter/databases/upgrade-appcenter-505-506-db2.sql` (for APPCNTR).

For an upgrade from Worklight Server V5.0.6 to V6.0.0:

- `WorklightServer/databases/upgrade-worklight-506-60-db2.sql` (for WRKLGHT)

- WorklightServer/databases/upgrade-worklightreports-506-60-db2.sql (for REPORTS)
- ApplicationCenter/databases/upgrade-appcenter-506-60-db2.sql (for APPCNTR)

These scripts are applied similarly to steps 4 and 6 in “Setting up your DB2 databases manually” on page 678

**Note:** If you are using Application Center, the size limit for applications stored on Application Center with IBM DB2 is 1 GB. If you have applications larger than 1 GB in the Application Center, remove them before starting the upgrade process.

### Scripts for MySQL

For an upgrade from Worklight Server V5.0.0, first upgrade to V5.0.5:

- WorklightServer/databases/upgrade-worklightreports-50-505-mysql.sql (for WLREPORT)

For an upgrade from Worklight Server V5.0.5 to V5.0.6:

- WorklightServer/databases/upgrade-worklight-505-506-mysql.sql (for WRKLGHT)
- ApplicationCenter/databases/upgrade-appcenter-505-506-mysql.sql (for APPCNTR)

For an upgrade from Worklight Server V5.0.6 to V6.0.0:

- WorklightServer/databases/upgrade-worklight-506-60-mysql.sql (for WRKLGHT)
- WorklightServer/databases/upgrade-worklightreports-506-60-mysql.sql (for REPORTS)
- ApplicationCenter/databases/upgrade-appcenter-506-60-mysql.sql (for APPCNTR)

These scripts are applied similarly to step 1.b in “Setting up your MySQL databases manually” on page 686.

### Scripts for Oracle

For an upgrade from Worklight Server V5.0.0, first upgrade to V5.0.5:

- WorklightServer/databases/upgrade-worklightreports-50-505-oracle.sql (for WLREPORT)

For an upgrade from Worklight Server V5.0.5 to V5.0.6:

- WorklightServer/databases/upgrade-worklight-505-506-oracle.sql (for WRKLGHT)
- ApplicationCenter/databases/upgrade-appcenter-505-506-oracle.sql (for APPCNTR)

For an upgrade from Worklight Server V5.0.6 to V6.0.0:

- WorklightServer/databases/upgrade-worklight-506-60-oracle.sql (for WRKLGHT)
- WorklightServer/databases/upgrade-worklightreports-506-60-oracle.sql (for REPORTS)

- ApplicationCenter/databases/upgrade-appcenter-506-60-oracle.sql (for APPCNTR)

These scripts are applied similarly to step 3 in “Setting up your Oracle databases manually” on page 690.

## Verify the Worklight Server

Installation of the new IBM Worklight version modifies the Worklight Server configuration. Verify that the new configuration meets your expectations.

The Worklight Server installation modified the application server's configuration to match the new Worklight Server version. It is a good idea to verify that the results meet your expectations.

The following procedures give instructions for how to complete these verifications for your server type.

### Expected server configuration for WebSphere Application Server Liberty Profile:

Procedures to help you verify your Liberty server configuration after the upgrade.

The file to be verified is `server.xml` of the particular WebSphere Application Server Liberty Profile instance. If you are using the Liberty server that is embedded in Worklight Server, it is the file `wlp/usr/servers/worklightServer/server.xml` inside the directory:

- `WL_INSTALL_DIR/server` on UNIX
- `C:\ProgramData\IBM\Worklight\WAS85liberty-server` on Windows

This file still contains modifications that you added **outside** the blocks that are delimited by comments such as:

```
<!-- Begin of features added by IBM Worklight installer. -->
...
<!-- End of features added by IBM Worklight installer. -->

and

<!-- Begin of configuration added by IBM Worklight installer. -->
...
<!-- End of configuration added by IBM Worklight installer. -->
```

However, the contents **inside** these blocks have been replaced with code that is suitable for the new Worklight Server version. If you made changes inside these blocks, the upgrade has removed them. You must adapt and reinstall them, as appropriate. Among these blocks, you should see code similar to the following examples:

In the `<featureManager>` element:

```
<feature>ssl-1.0</feature>
<feature>servlet-3.0</feature>
<feature>jdbc-4.0</feature>
<feature>security-1.0</feature>
<feature>appSecurity-1.0</feature>
```

In the `<server>` element, for the Worklight run time and the Worklight Console:

```
<!-- Declare the 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 Worklight Server application. -->
<webContainer invokeFlushAfterService="false"/>

```

Similarly, for the Application Center:

```

<!-- Declare the IBM Application Center Console application. -->
<application id="appcenterconsole" name="appcenterconsole" location="appcenterconsole.war" type="war">
    <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" type="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>

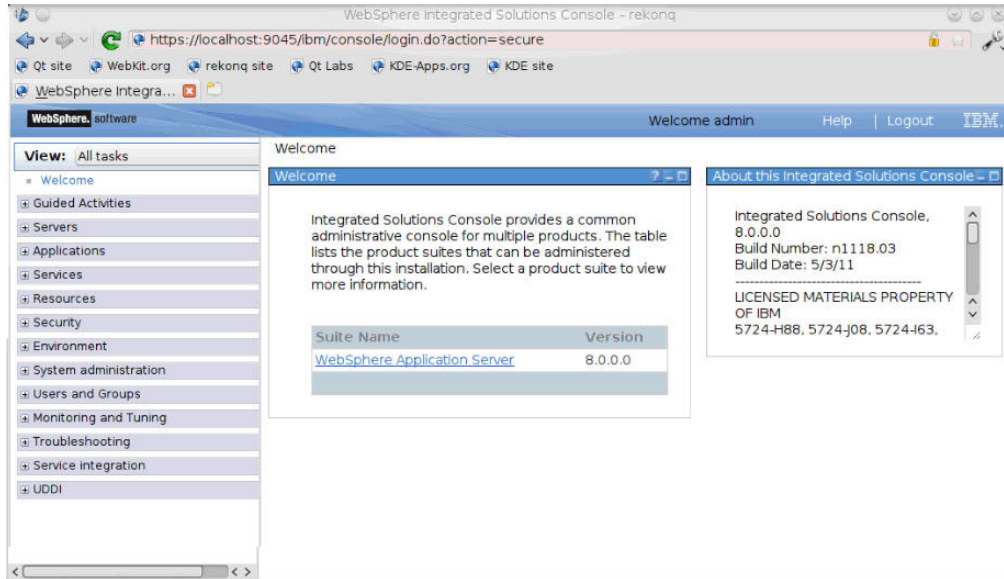
```

### Expected server configuration for WebSphere Application Server Full Profile:

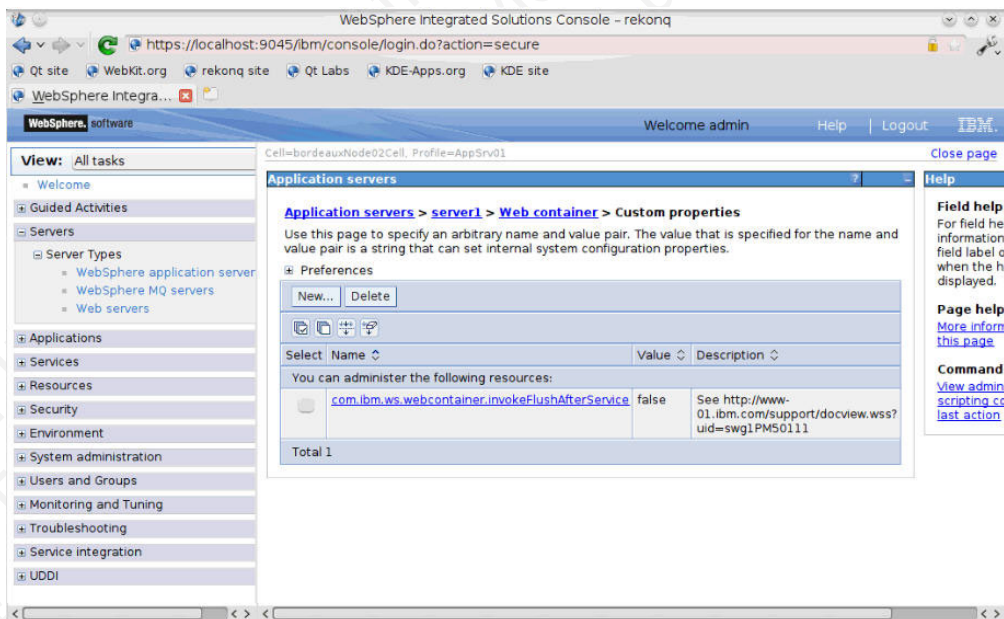
Procedures to help you verify your WebSphere Application Server Full Profile server configuration after the upgrade.

To check the server configuration, use the WebSphere Application Server administration console. The common location of this console is <https://localhost:9043/ibm/console>. (The following screen captures use a different port.)

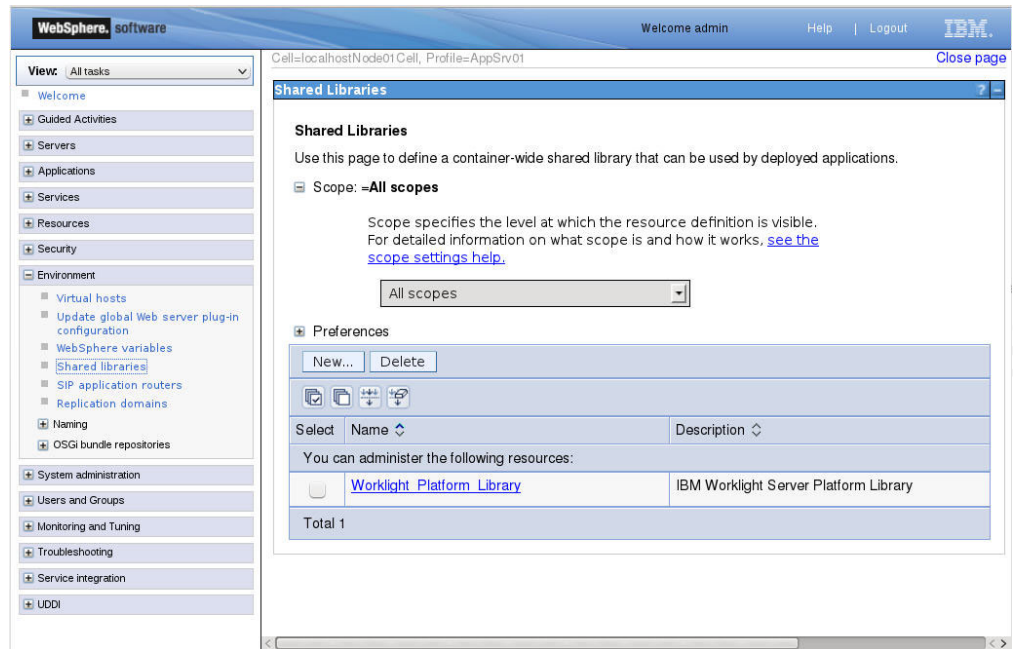




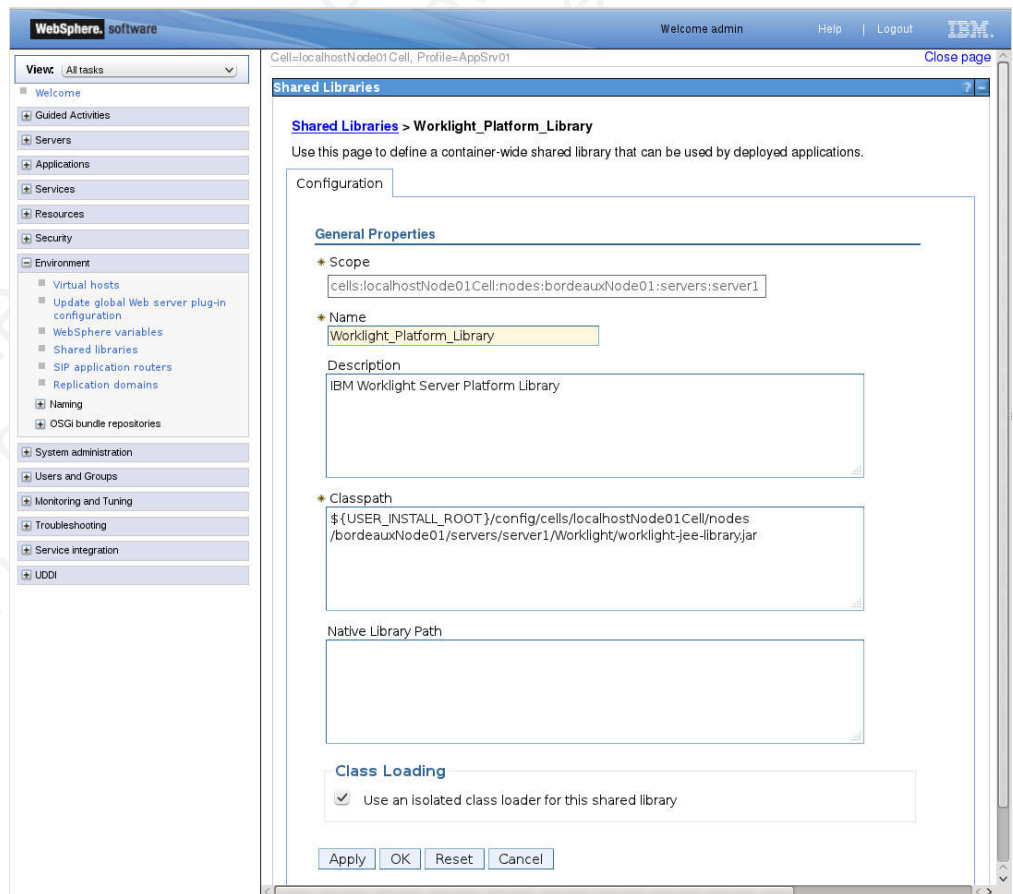
When you check the custom properties of the web container of the server, you should see that property `com.ibm.ws.webcontainer.invokeFlushAfterService` has the value `false`.



When you check the Shared Libraries, you should find one that is named `Worklight_Platform_Library`.

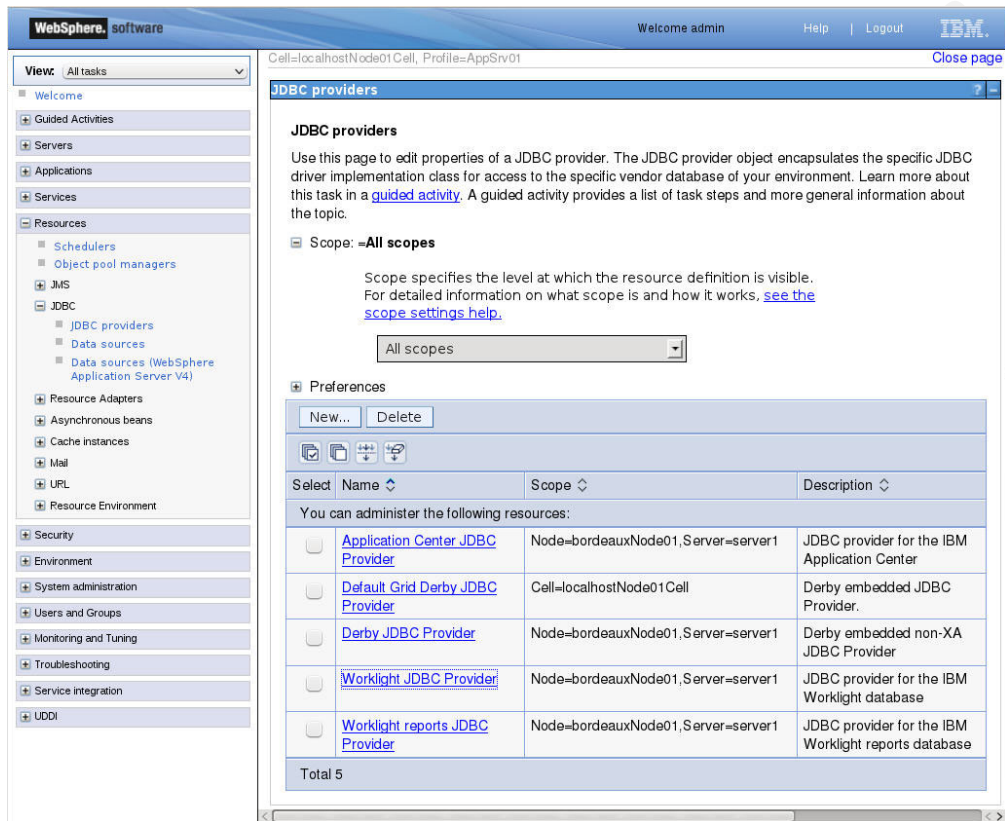


The detail configuration of this shared library should show a classpath that consists of the `worklight-jee-library.jar`:

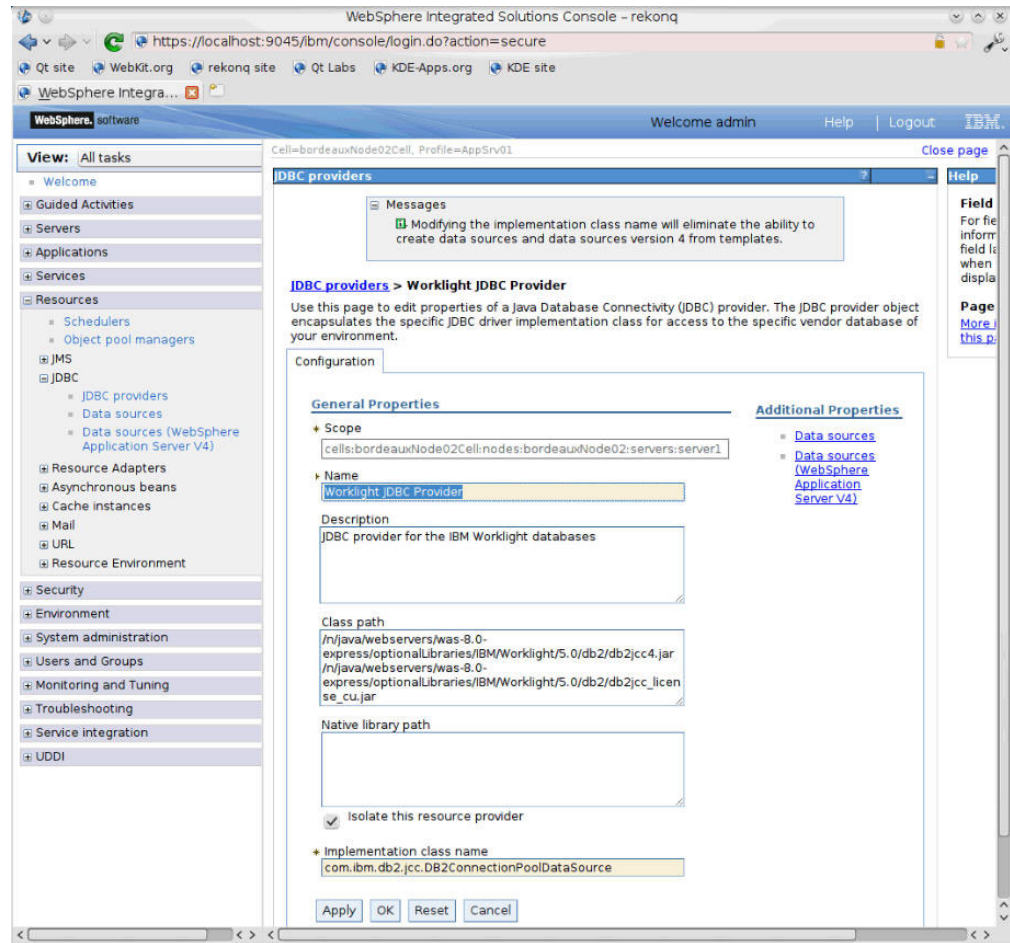


When you check the JDBC providers, you should see three of them:

- Application Center JDBC Provider
- Worklight JDBC provider
- Worklight reports JDBC Provider

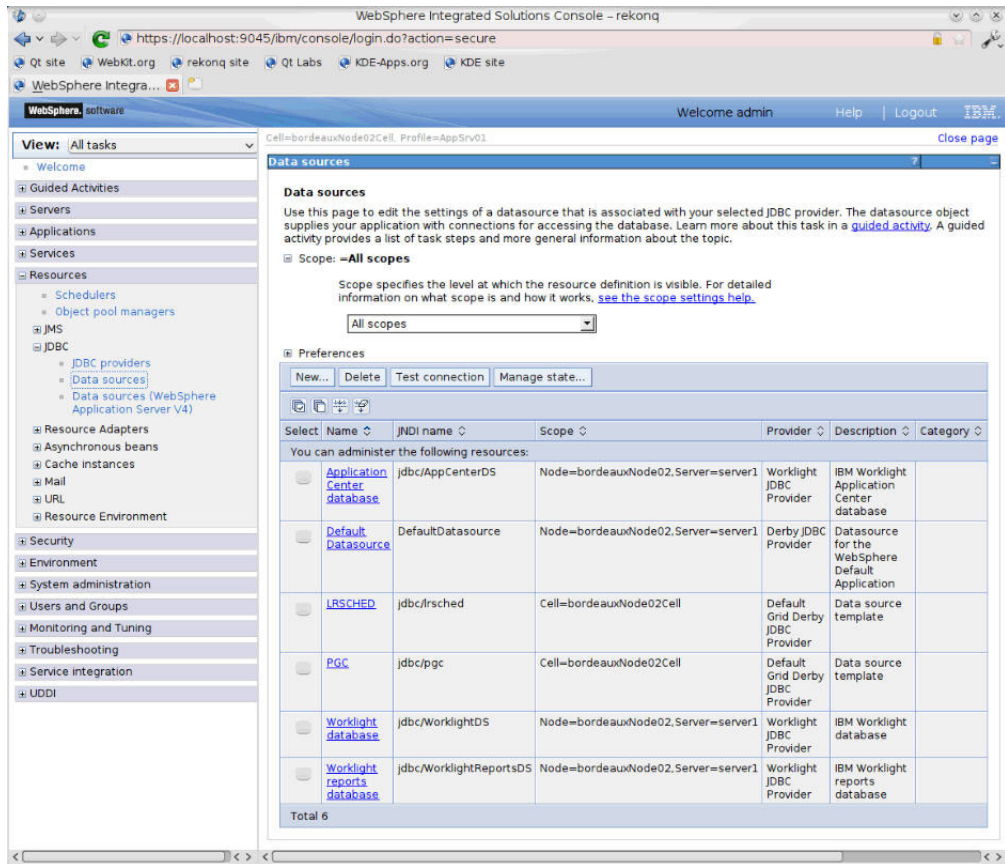


The details of the JDBC provider can look similar to the following screen capture, if the database type is DB2:



When you check the data sources, you should see three of them:

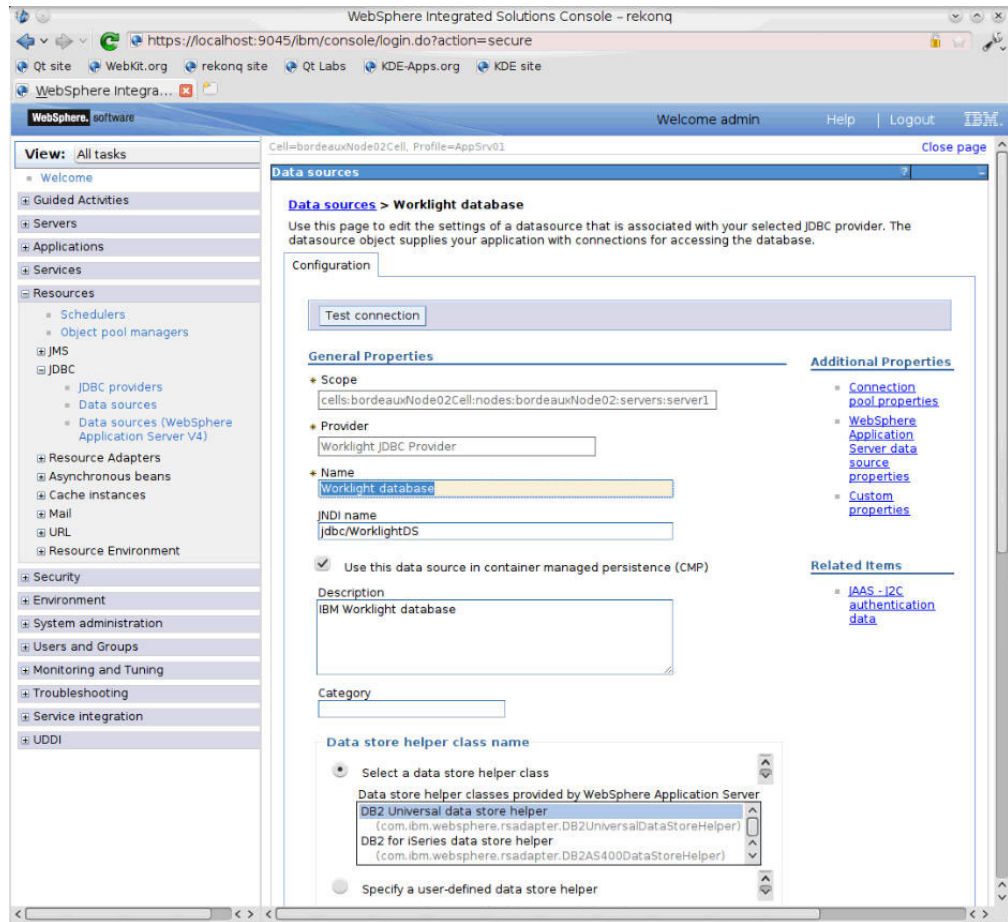
- Worklight database
- Worklight reports database
- Application Center database



When you look at the configuration of each database, you should see the following JNDI names:

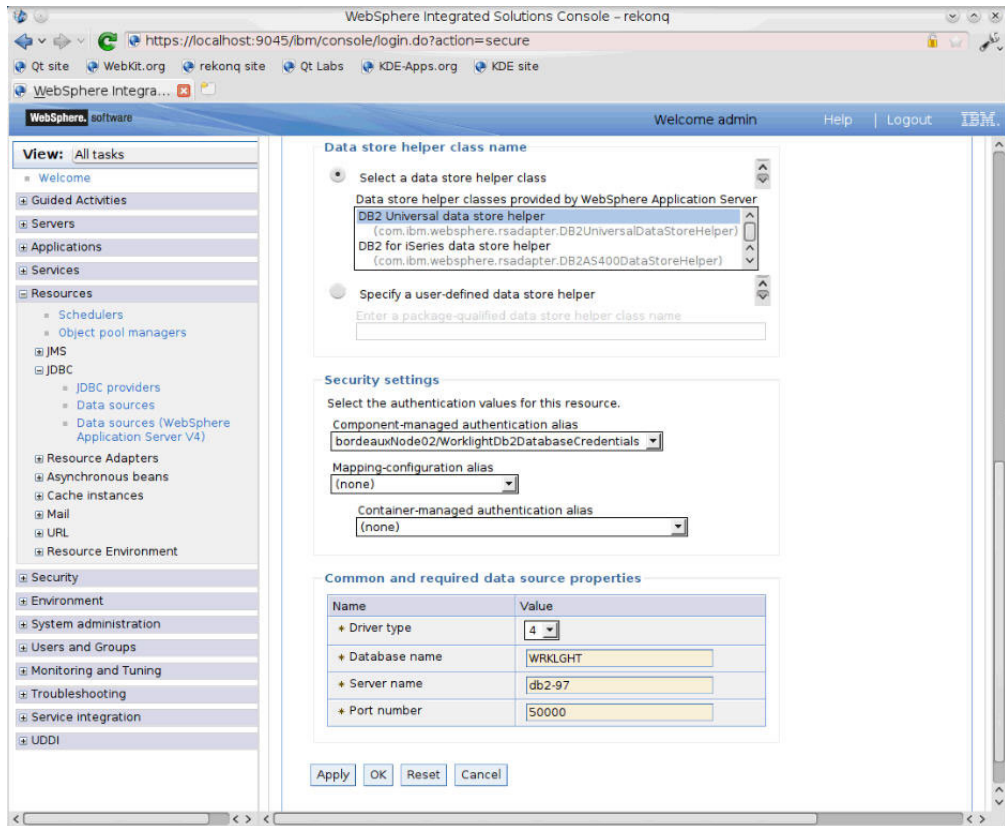
- jdbc/WorklightDS
- jdbc/WorklightReportsDS
- jdbc/AppCenterDS

**Note:** The **Test connection** button does not work in a WebSphere Application Server Network Deployment configuration because the scope of the data source definition normally does not include the deployment manager server.

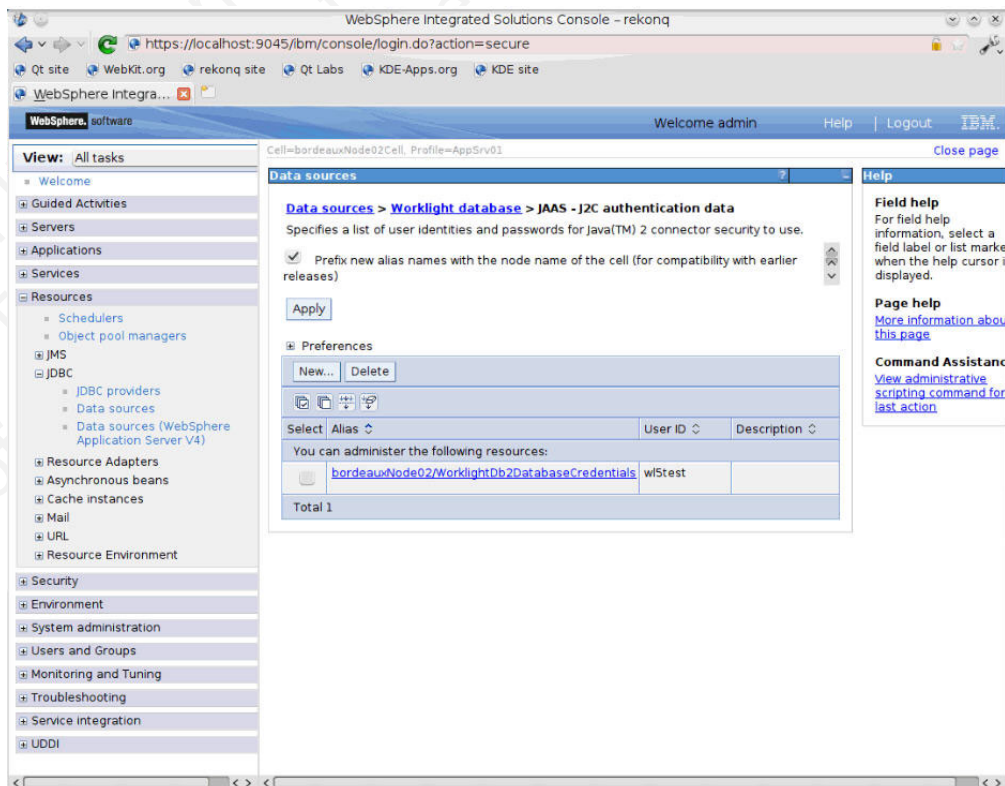


For a DB2 data source, the configuration should refer to a JAAS authentication alias, called `WorklightDb2DatabaseCredentials` (possibly with a suffix). For other types of databases, the credentials are part of the custom properties of the data source.

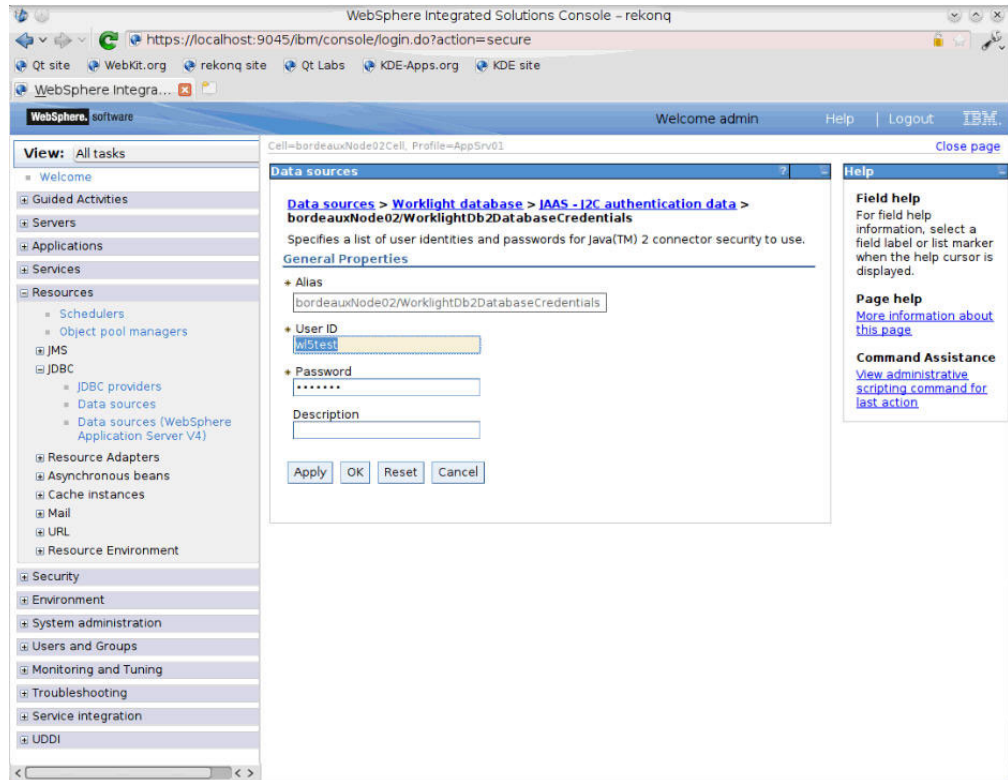




The JAAS authentication alias for DB2 can look like the following screen captures.



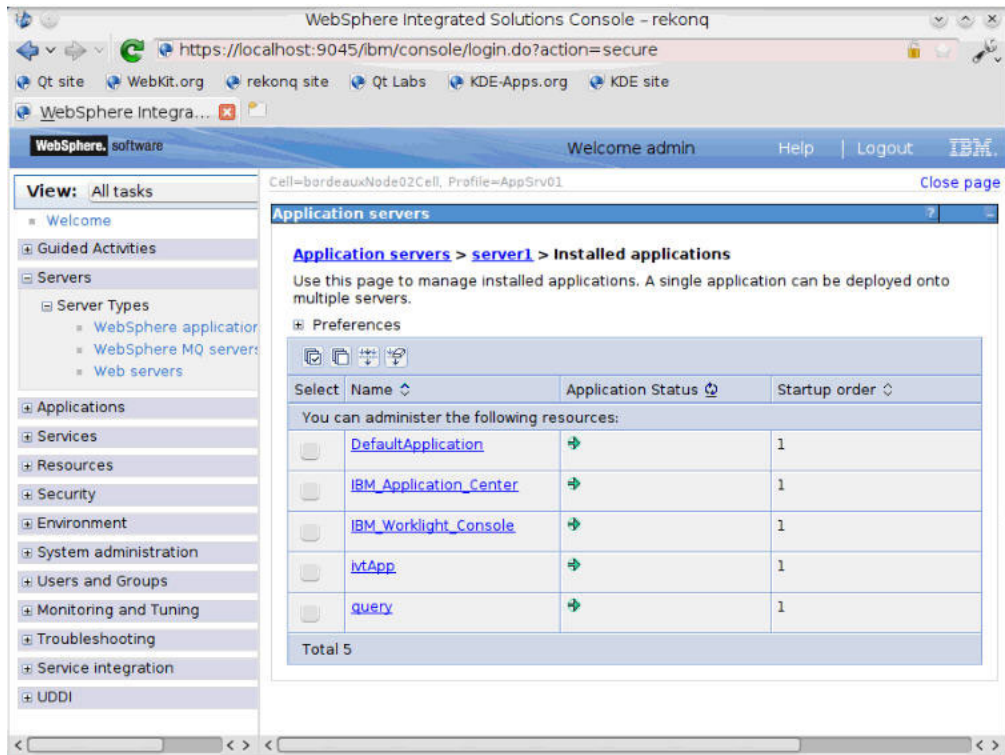




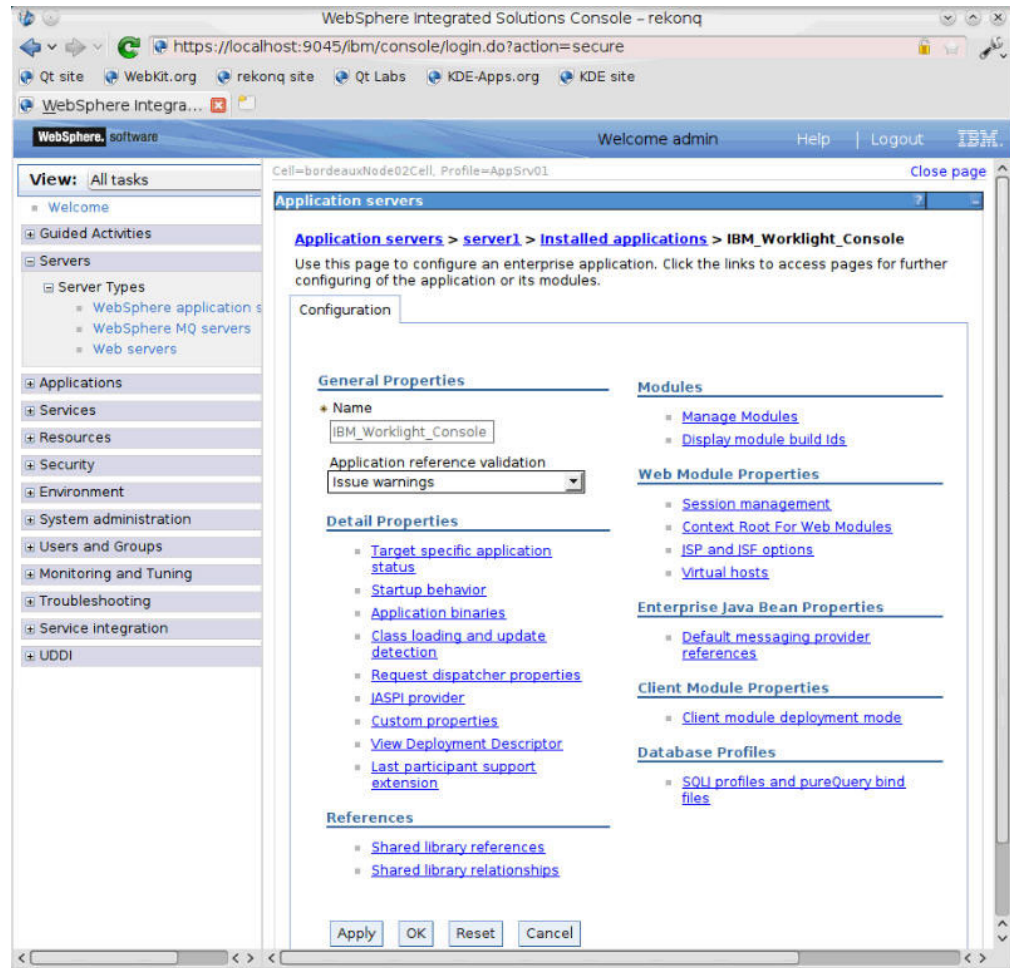
When you look at the installed applications list of the server, you should see:

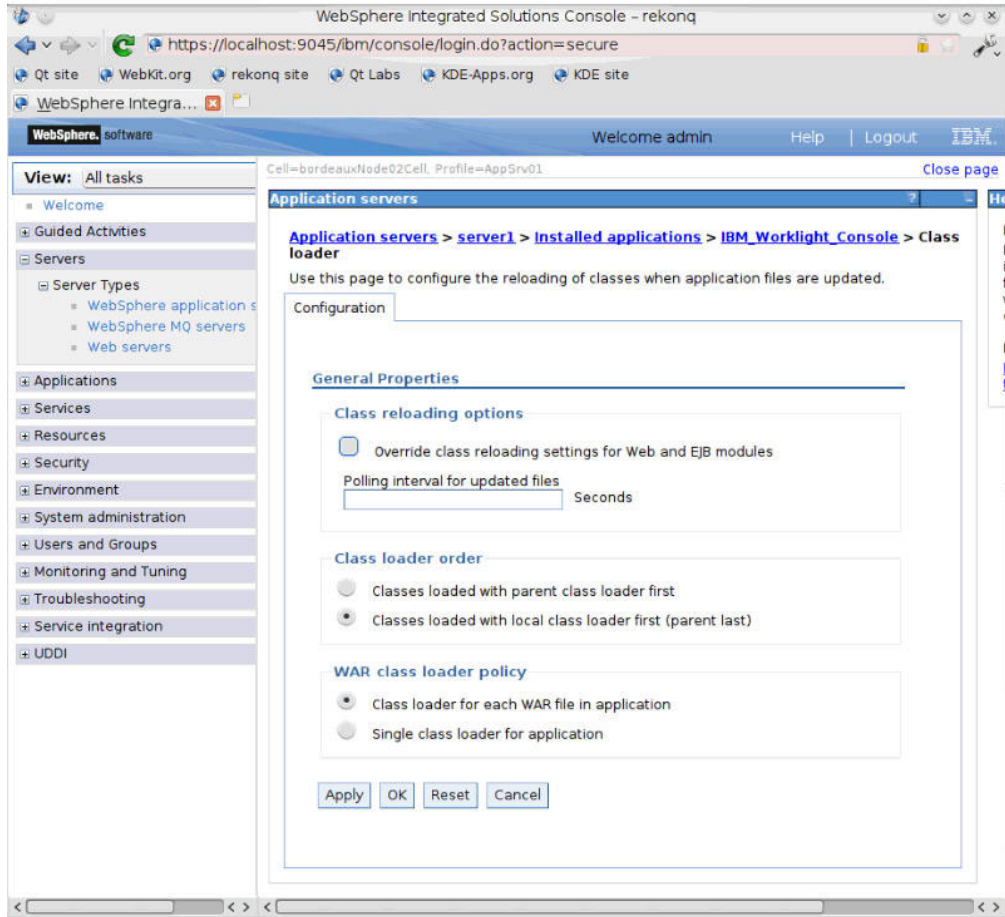
- IBM\_Worklight\_Console
- IBM\_Application\_Center\_Console
- IBM\_Application\_Center\_Services

These entries can appear differently from the following screen capture, possibly displaying a numeric suffix for uniqueness (in case of WebSphere Application Server Network Deployment).

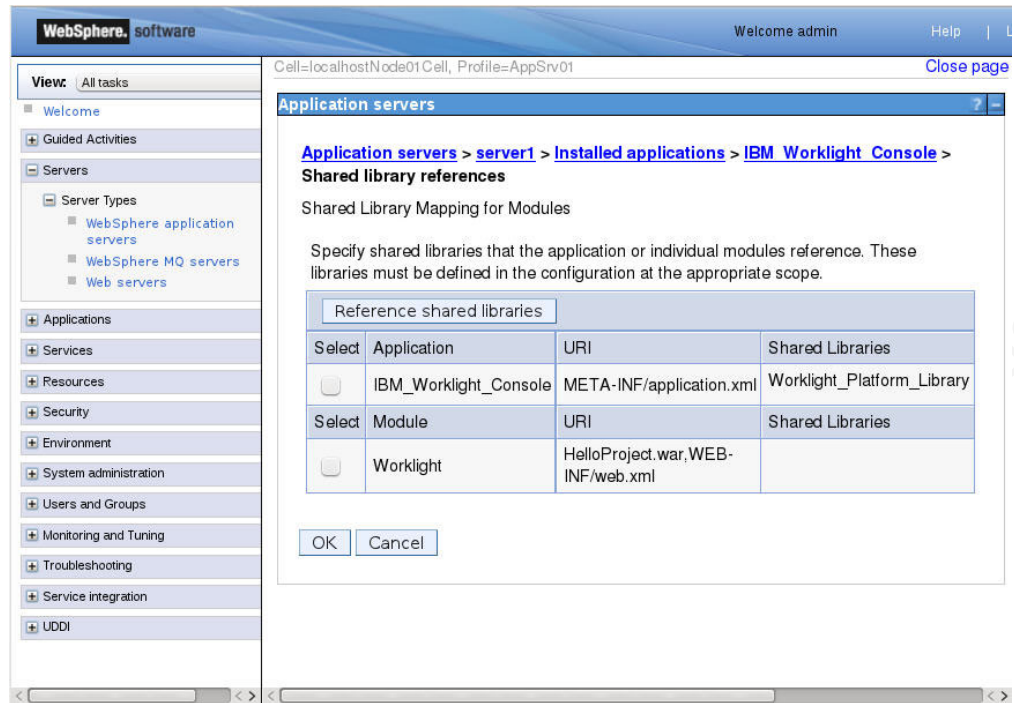


The configuration of each of the web applications should have classloader settings of parent last and Class loader for each WAR file in application, as shown in the following screen captures.

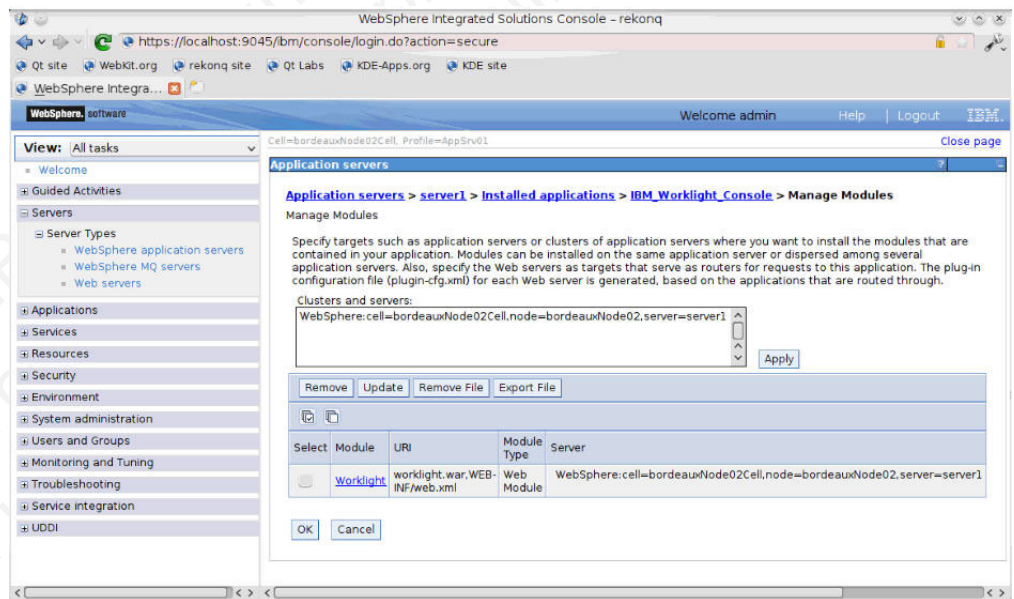




When you look at the shared library references of the application IBM\_Worklight\_Console, you should see a reference to the Worklight\_Platform\_Library.

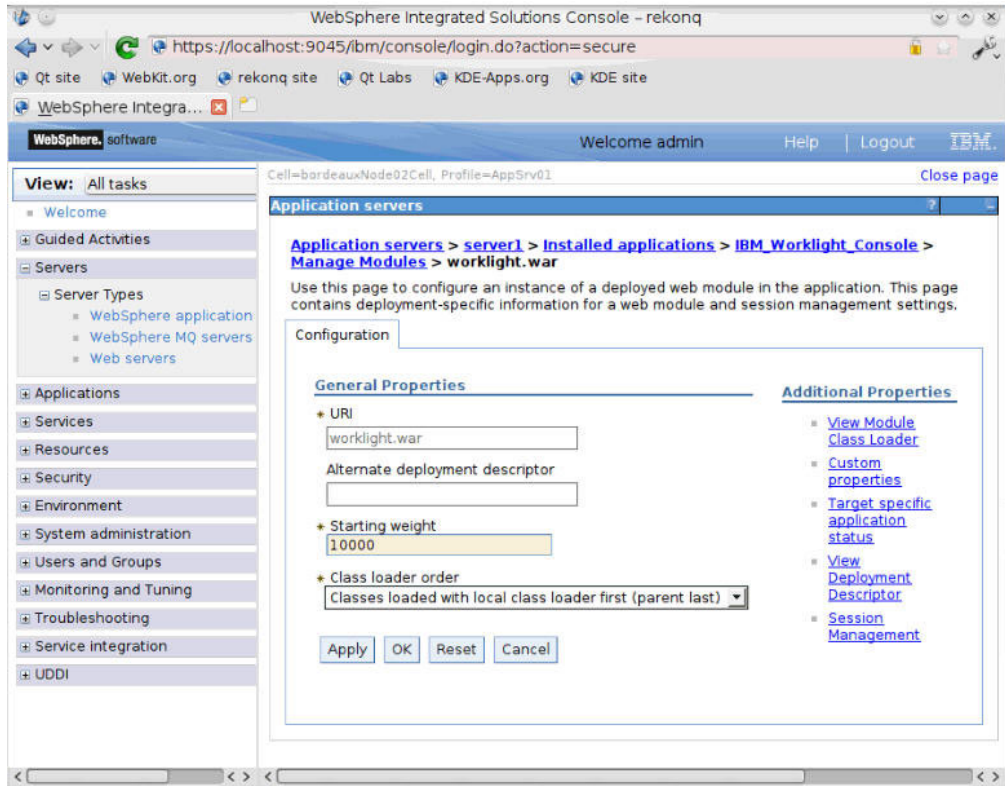


The IBM\_Worklight\_Console application has a single module, Worklight, corresponding to the project WAR file.

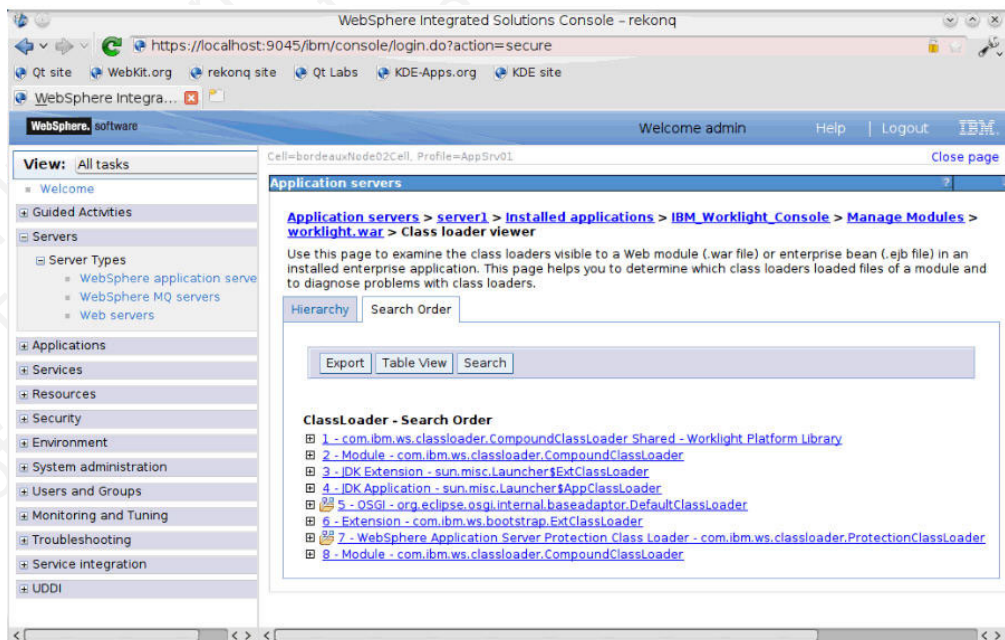


The module settings for each module among the Worklight Server applications should contain a class loader order of parent last.





The class loader viewer for module worklight.war in the IBM Worklight Console should mention the Worklight\_Platform\_Library.



### Expected server configuration for Apache Tomcat:

Procedures to help you verify your Apache Tomcat application server configuration after the upgrade.



The main file to be verified is `conf/server.xml` of the particular Apache Tomcat server.

This file still contains modifications that you added **outside** the block that is delimited by comments such as:

```
<!-- Begin of Context and Realm configuration added by IBM Worklight installer. -->
...
<!-- End of Context and Realm configuration added by IBM Worklight installer. -->
```

However, the contents **inside** these blocks have been replaced with code that is suitable for the new Worklight Server version. If you made changes inside this block, the upgrade has removed them. You must adapt and reinstall them, as appropriate.

In this block, you should see code similar to the following examples:

```
<!-- Declare the IBM Worklight Console application. -->
<Context path="/worklight" docBase="worklight">

    <!-- Declare the IBM Worklight Console 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. -->
    <Resource name="jdbc/WorklightDS" type="javax.sql.DataSource"
    driverClassName="com.ibm.db2.jcc.DB2Driver"
    url="jdbc:db2:// db2server:50000/WRKLGHT" username=" db2username"
    password=" password" auth="Container" maxActive="8" maxIdle="4"
    maxWait="5000"/>

    <!-- Declare the IBM Worklight Console Reports database. Used through
    property wl.reports.db.jndi.name. 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. -->
    <Resource name="jdbc/WorklightReportsDS" type="javax.sql.DataSource"
    driverClassName="com.ibm.db2.jcc.DB2Driver" url="jdbc:db2://
    db2server:50000/WLREPORT" username=" db2username"
    password=" password" auth="Container" maxActive="8" maxIdle="4"
    maxWait="5000"/>
</Context>

<!-- Declare the IBM Application Center applications. -->

<!-- Declare the IBM Application Center Console application. -->
<Context path="/appcenterconsole" docBase="appcenterconsole"/>

<!-- Declare the IBM Application Center Services application. -->
<Context path="/applicationcenter" docBase="applicationcenter">

    <!-- Declare the IBM Application Center Services database. -->
    <Resource name="jdbc/AppCenterDS" type="javax.sql.DataSource"
    driverClassName="com.ibm.db2.jcc.DB2Driver" url="jdbc:db2://
    db2server:50000/APPCNTR" username=" db2username"
    password=" password" auth="Container" maxActive="8"
    maxIdle="4" maxWait="5000"/>

</Context>

<!-- 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/realms-howto.html. -->
<Realm className="org.apache.catalina.realm.MemoryRealm"/>
```

Similarly, the file `conf/tomcat-users.xml` should contain a block that is delimited by comments such as:

```
<!-- Begin of configuration added by IBM Worklight installer. -->
...
<!-- End of configuration added by IBM Worklight installer. -->
```

This file still contains modifications that you added **outside** this block. However, the contents **inside** these blocks have been replaced with a definition for the role `appcenteradmin`, similar to the following example:

```
<!-- Define roles and users for the IBM Application Center. -->
<role name="appcenteradmin"/>
<user name="appcenteradmin" password="admin" roles="appcenteradmin"/>
<user name="demo" password="demo" roles="appcenteradmin"/>
<user name="guest" password="guest" roles="appcenteradmin"/>
```

Finally, the file `conf/catalina.properties` should contain property definitions similar to the following example:

```
# Added by the IBM Worklight installer.
# The directory with binary files of the 'aapt' program, from the Android SDK's
# platform-tools package.
android.aapt.dir= WL_INSTALL_DIR/ApplicationCenter/tools/android-sdk

# Added by the IBM Worklight installer.
# Define the AppCenter services endpoint in order for the AppCenter
# console to be able to invoke the REST service.
# You need to enable this property only if the server is behind a
# reverse proxy.
#ibm.appcenter.services.endpoint=http://<proxy>/applicationcenter
```

### Verify and update the HTTP redirections:

If you are upgrading Worklight Server on a clustered application server environment, you should also update IBM HTTP Server after you install IBM Worklight V6.0.0.

If your Worklight Server upgrade is to be installed on a WebSphere Application Server Network Deployment clustered environment or a WebSphere Application Server Liberty Profile farm, you may need to update IHS after you install Worklight Server V6.0.0. For general information about installing these types of application server, see:

- “Setting up IBM Worklight in an IBM WebSphere Application Server Network Deployment V8.5 cluster environment” on page 147
- “Setting up IBM Worklight in an IBM WebSphere Application Server Liberty Profile farm” on page 161

If your application server receives HTTP requests forwarded by an HTTP server, the HTTP server configuration may require updating.

For IBM HTTP Server, in the IHS plugin file the context root of the applications must be updated especially for the session affinity configuration section. The following example is a configuration for Application Center that is deployed with its default settings, and a project that is deployed with a root context of `/worklight`.

```
<UriGroup Name="default_host_defaultServer_default_node_Cluster_URIs">
  <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid"
    Name="/worklight/*"/>
  <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid"
```

```
        Name="/applicationcenter/*"/>
    <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionId"
        Name="/appcenterconsole/*"/>
</UriGroup>
```

## Configure the Application Center

Set configuration options for the Worklight Application Center.

In Worklight Server V5.0.6.x, the Application Center supports new configuration options.

The Application Center security should be configured as documented in “Configuring the Application Center after installation” on page 118 and in:

- For WebSphere Application Server Liberty Profile: “Configuring WebSphere Application Server Liberty Profile” on page 120
- For WebSphere Application Server Full Profile: “Configuring WebSphere Application Server full profile” on page 119
- For Apache Tomcat: “Configuring Apache Tomcat” on page 121

The endpoint URI should be configured as documented in “Defining the endpoint of the application resources” on page 139 and in:

- For WebSphere Application Server Liberty Profile: “Configuring the endpoint of the application resources (Liberty profile)” on page 141
- For WebSphere Application Server Full Profile: “Configuring the endpoint of the application resources (Full Profile)” on page 140
- For Apache Tomcat: “Configuring the endpoint of the application resources (Apache Tomcat)” on page 142

## Final configuration tasks

Complete the topics that follow to set security options for the new version of the Worklight Application Center and to restart the Application Center.

### Other configuration (security):

Update the configuration of the Worklight Application Center to use new security features in this release.

Consider updating the security configuration to use the new features of Worklight V5.0.6. For more information, see Improved security and user experience.

### Start the Application Server:

Restart the Worklight Application Center.

If all previous tasks are completed, now you can start the application server.

## Finish migrating the IBM Worklight projects

Complete the migration of your IBM Worklight projects and applications by updating the Worklight Server WAR file.

Using the Worklight Console, deploy the upgraded server-side application artifacts (the .wlap file and adapters) that you create in step “Migrate your Worklight projects” on page 192.

Applications that are built with IBM Worklight V5.0.0.3 or later, and whose server-side artifacts are successfully ported to Worklight V6.0.0 and tested on a test server, should continue to work without requiring the device users to upload a new version of the application. However, the Direct Update feature (“Direct updates of app versions to mobile devices” on page 755) stops working on those versions. For instructions about how to re-enable direct update for those applications, see “Migrate your Worklight projects” on page 192.

### Recovering from an unsuccessful upgrade

Instructions for how to recover from a failed installation or to revert to the previous version of Worklight Server.

#### About this task

If the Worklight Server upgrade fails for any reason, use the following procedure to restore the previous Worklight Server version.

#### Procedure

The **Roll Back** button of IBM Installation Manager is not supported for Worklight Server. Therefore, to return to the previous version:

1. Uninstall Worklight Server, with Installation Manager.
2. Install the old version of Worklight Server with Installation Manager, specifying the same installation parameters that you used previously.
3. Restore the databases. For more information, see “Back up the databases” on page 199.
4. Restore the application server. For more information, see “Back up the Application Server” on page 199.
5. If the server fails to start and load the applications, delete the server's workarea before starting it again. For example, for a WebSphere Application Server Liberty Profile backup, the workarea is the directory `<LibertyInstallDir>/usr/servers/<serverName>/workarea`.

---

## Migrating from IBM Worklight V5.0.5 to V5.0.6

When you open your IBM Worklight project with 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. There are also some modifications that you must be aware of, 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 IBM Worklight V5.0.0.3 to V5.0.5” on page 229.

### Worklight Server

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

When Worklight Server restarts after an upgrade, it checks database table `WORKLIGHT_VERSION` to verify that the database schema is consistent with the new version of Worklight Server. If the IBM Worklight database schema changed in the new release, and the schema is found to be inconsistent or empty, the existing

schema is dropped. It is then re-created to be consistent with the new release, updating the WORKLIGHT\_VERSION table appropriately.

This action deletes all data in your existing IBM Worklight development database. In most cases, this deletion does not cause problems because only the tables that are related to IBM Worklight and its reports are dropped and re-created.

**Important:** If you are upgrading Worklight Server to V6.0.x 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 Worklight Server in a production environment” on page 191.

## Usage of existing applications

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 “Installing Worklight Server” on page 41 topics). When you do so, the existing data in the server's project 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 project .war, .wlap, and .adapter files to the new server.

## 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 Worklight 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 IBM Worklight upgrader:

- 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:** 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 228.

If you created your current project with an earlier version of 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 228.

## 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 331.

**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 322.

**iOS apps:** add the following piece of 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 620 and the module *Using Worklight API in Native Java ME applications*, under category 7, *Developing native applications with Worklight*, in the Chapter 3, “Tutorials and samples,” on page 25.

**Windows Phone 8 applicationBar folder:** when you migrate a Windows Phone 8 project to IBM Worklight V5.0.6, the `images/applicationBar` folder under the root of the IBM Worklight project becomes the `nativeResources/applicationBar` folder and stays under the root of the IBM Worklight project. See “`WL.OptionsMenu.addItem`” on page 601.

## 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:

- The Worklight 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 `createIdentity` 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.
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(doj, dijit) {
            dojo.ready(function() {

                });
        });
}
```

---

## Migrating from IBM Worklight V5.0.0.3 to V5.0.5

When you open your IBM Worklight project with a newer version of Worklight 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.

### 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 have 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 317.

## IBM Worklight Application Center

When you migrate from V5.0.0.3 to V5.0.5, the Derby database for the Application Center is migrated to the new format as part of the installation.

When you migrate 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 Java Platform, Enterprise Edition security role named **appcenteruser**, which consists of the group of users that are authorized to use the mobile client. See “Configuring the Application Center after installation” on page 118 to learn how to set up these security roles.

## Cordova

IBM Worklight V5.0.5 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 V5.0.0.3 was bundled with Cordova 1.6.1. For information about Cordova changes, review the release notes in the Apache Cordova Change Log site: The Apache Software Foundation.

## jQuery

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

## Dojo

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 228.

If you have existing IBM Worklight Dojo projects created with a previous version, consider migrating the code to the new architecture. This action ensures that the code performs more reliably and that the page continues to work when it makes further changes in Rational Publishing Engine, 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.

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. Developing IBM Worklight applications

You use Worklight Studio, the IBM Worklight client, and the server-side API to develop cross-platform mobile applications, desktop applications, or web applications.

### About this task

This information is designed to help users develop applications for various channels by using the IBM Worklight. It is intended for developers who are familiar with web, or native application development.

This section covers client-side development and server-side development topics, such as the integration with back-end services, and push notifications.

IBM Worklight 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.

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

---

### Worklight Studio overview

Worklight Studio is an Eclipse-based integrated development environment (IDE). You can use Worklight Studio to create mobile applications for various mobile operating systems, and to integrate applications with existing services.

With Worklight Studio, you can add custom plug-ins to Eclipse. For instance, you can use a Rational Team Concert plug-in to control your source code, track changes, and create daily builds without installing an extra development application. You can also build server applications, and applications for different mobile device operating systems, from a single IDE.

**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.

## Native and web development technologies

Worklight Studio supports native and web development technologies such as HTML5, Apache Cordova, and Java. With these development technologies, you can use the following capabilities:

- Develop mobile applications with pure HTML5.
- Use a compatible JavaScript framework, such as jQuery Mobile, Dojo Mobile, or Sencha Touch. You can use the user interface widgets and functions that are provided by these frameworks.
- Use Apache Cordova so that your mobile application can access native device functionality. To access a special device module, such as one for near field communication (NFC), you can develop a native extension that you expose to JavaScript through an Apache Cordova plug-in, which is a small native-to-JavaScript wrapper.

## Shell development

For hybrid mobile applications, Worklight Studio uses a default hybrid shell that provides you with capabilities to use web and native technologies. With shell development, you can use the following capabilities:

- Separate native-component implementation from web-based implementation, and split this work between different developers. For example, you can create a custom shell, and add third-party native libraries, implement custom security, or provide extended features that are specific to your company.
- Use shells to restrict or enforce specific corporate guidelines, such as design or security rules. For example, you can use a shell to add a default style to your mobile application, or to disable the camera of the device.

## Runtime skinning

With Worklight Studio, you use a common environment as a basic development point and all environments can share base code. You can then create a version of this environment that is specific to a device, for example an iPad, by creating a variant of the base and implementing only the required changes. At run time, an extra function that is called runtime skinning makes your mobile application switch between different sets of customization.

## Integration of device-specific SDKs

Each vendor of mobile devices supplies its own development environment as part of a software development kit (SDK). Worklight Studio generates a project for each supported SDK, such as Xcode for iOS development. Some vendors require that you use their SDK for specific tasks, such as building the binary application. The integration of device-specific SDKs within Worklight Studio links your Worklight Studio project with the native development environment (such as Xcode). You can then switch between a native development environment and Worklight Studio. Any change in the native development environment is reflected to your Worklight Studio project, which reduces manual copying steps.

## Third-party library integration

Depending on your programming approach, your mobile application can include several JavaScript frameworks, such as Sencha Touch, jQuery Mobile, or Dojo Mobile. This third-party library integration facilitates code reuse and reduces implementation times. If you have a shell project, several types of compatible native code or libraries can be included.

## Integrated build engine

The build chain of Worklight Studio combines common implementation code, which is used on all target platforms, with platform-unique implementation code, which is used on a specific target platform. At build-time, the integrated build engine combines these implementations into a complete mobile application. You can then use a single, common implementation for as much of the mobile application function as possible, instead of a unique implementation for every supported platform.

## Integrated development tools

You can extend the Eclipse IDE with custom plug-ins, and use Worklight Studio to develop all components of your application from within the same development environment. These components include the mobile application and the integration code, which is called IBM Worklight adapters. With integrated development tools, you can develop and test these adapters within Worklight Studio.

## Mobile browser simulator

Worklight Studio includes a mobile browser simulator that you can use during the development cycle. You can use the mobile browser simulator to test mobile web and hybrid applications that are displayed in a desktop browser. This mobile browser simulator support cross-platform browser testing for mobile devices.

Many desktop browsers and mobile browsers use the WebKit engine as their underlying core technology, which provides a common platform for developing applications that support HTML5, CSS3, and JavaScript. If you use a desktop browser that is based on WebKit, such as Chrome or Safari, to host the mobile browser simulator, you can validate the behavior of the application in the browser before you deploy it on the device. When you test your application on the device or mobile emulator, you can verify that the core WebKit engine provides the same consistent user experience that you verify when you test with the browser.

The mobile browser simulator also provides default implementations of the various Apache Cordova APIs. You can then use these default implementations to test hybrid applications that leverage the device features, without having to run the applications on the actual device.

## Ant tasks

Worklight Studio provides a set of Ant tasks that you can use to run a mobile application build for various platforms. For example, you can distribute build tasks to various build machines that run Apple OS X (for an Apple iOS binary file), or Microsoft Windows (for a Microsoft Windows Phone 8 binary file). If you use this mechanism, you do not need to access multiple build machines to create several builds for specific mobile platforms.

---

## IBM Worklight client-side API overview

You can use IBM Worklight client-side API capabilities to improve application development, and IBM Worklight server-side API to improve client/server integration and communication between mobile applications and back-end systems.

With the IBM Worklight client-side API, your mobile application has access to various IBM Worklight features during run time, by using libraries that are bundled into the application. The libraries integrate your mobile application with Worklight Server by using predefined communication interfaces. The libraries also provide unified access to native device functionality, which simplifies application development.

IBM Worklight client-side API includes native, hybrid, mixed hybrid, and web-based APIs. These APIs provide support for all mobile development approaches with enhanced security and integration features. IBM Worklight client-side API components deliver a uniform bridge between web technologies (HTML5, CSS3, JavaScript) and the native functions that are available on different mobile platforms.

For hybrid and mixed hybrid applications, the Apache Cordova plug-ins that are included add native capabilities and cross-platform user interface controls.

The IBM Worklight client-side API provide access to IBM Worklight functions across multiple device platforms and development approaches. Applications that are built by using web technologies can access Worklight Server through the APIs by using JavaScript, and application using native components can access the APIs directly by using Java and Objective-C. Mobile applications developed with the hybrid and native development approaches, including the applications that run on Android, iOS, or Java ME, benefit from simplified application security and integration features of IBM Worklight.

IBM Worklight client-side API components also provide the following features, which improve application development.

### **Cross-platform compatibility layer**

This cross-platform compatibility layer supports development for all supported platforms. If you develop hybrid mobile applications, you can access common control elements such as tab bars and clipboards, and native device capabilities such as the location service or camera. You can extend these functions for Android and iOS by using a custom shell.

### **Client to server integration**

Client to server integration ensures transparent communication between a mobile application that is built with IBM Worklight technology, and Worklight Server. IBM Worklight mobile applications always use an SSL-enabled connection to the server, including for authentication. With such an integration, you can manage your applications and implement security features such as remotely disabling the ability to connect to Worklight Server for a specific application version, or updating the web resources of a hybrid application.

## Encrypted data store

This encrypted data store is located on the device and can access private data by using an API. This helps prevent malicious users to access private data, because all they can obtain is highly encrypted data. The encryption uses ISO/IEC 18033-3 security standards, such as AES256 or PKCS#5, that complies with the United States National Security Agency regulations for transmitting confidential or secret information. The key that is used to encrypt the information is unique to the current user of the application and the device. Worklight Server issues a special key when a new encrypted data store is created.

## JSONStore

A JSONStore store is included in IBM Worklight to synchronize mobile application data with related data on the back-end. JSONStore provides an offline-capable, key-value database that can be synchronized. JSONStore implements the application local read, write, update, and delete operations and use the IBM Worklight adapter technology to synchronize the related back-end data.

## Run time skinning

Run time skinning is a feature that helps you incorporate an adaptive design that you can adapt to each mobile device. The IBM Worklight run time skin is a user-interface variant that you can apply during application run time, which is based on device properties such as operating system, screen resolution, and form factor. This type of user-interface abstraction helps you develop applications for multiple mobile device models at the same time.

---

## IBM Worklight server-side API overview

Worklight Server provides a set of mobile capabilities with the use of client/server integration and communication between mobile applications and back-end systems.

### Server-side application code

You can develop server-side application code and optimize performance, security, and maintenance. By developing server-side application code, your mobile application has direct access to back-end transactional capabilities and cloud-based services. This improves error handling, and enhances security by including more custom steps for request validation or process authorization.

### Built-in JSON translation capability

A built-in JSON translation capability reduces the footprint of data transferred between the mobile application and Worklight Server. JSON is a lightweight and human-readable data interchange format. Because JSON messages have a smaller footprint than other comparable data-interchange formats, such as XML, they can be more quickly parsed and generated by mobile devices. In addition, Worklight Server can automatically convert hierarchical data to the JSON format to optimize delivery and consumption.

### Built-in security framework

You can use encryption and obfuscation techniques with a built-in security framework to protect both user-specific and application business logic. A built-in



security framework provides easy connectivity or integration into your existing enterprise security mechanisms. This security framework handles connection credentials for back-end connectivity, so the mobile application can use a back-end service, without having to know how to authenticate with it. The authentication credentials stay with Worklight Server, and do not stay on the mobile device. If you are running Worklight Server with IBM WebSphere Application Server, you can use enterprise-class security and enable Single-Sign-On (SSO) by using IBM Lightweight Third Party Authentication (LTPA).

## Adapter library

You can use the adapter library to connect to various back-end systems, such as web services, databases, and messaging applications. For example, IBM Worklight provides adapters for SOAP or XML over HTTP, JDBC, and JMS. Extra adapters simplify integration with IBM WebSphere Cast Iron, which in turn supplies connectors for various cloud-based or on-premise services and systems. With the adapter library, you can define complex lookup procedures and combine data from multiple back-end services. This aggregation helps to reduce overall traffic between a mobile application and Worklight Server.

## Unified push notification

You can use unified push notification, which simplifies the notification process because the application remains platform-neutral. Unified push notification is an abstraction layer for sending notifications to mobile devices by using either the device vendor's infrastructure or Worklight Server SMS capabilities. The user of a mobile application can subscribe to notifications through the mobile application. This request, which contains information about the device and platform, is sent to the Worklight Server. The system administrator can manage subscriptions, push or poll notifications from back-end systems, and use the Application Center to send notifications to mobile devices.

---

## Artifacts produced during development cycle

When you use Worklight Studio within the IBM Worklight framework to develop a mobile application, you produce client and server artifacts.

### 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.

### Application metadata and resources (`.wlapp`)

A `.wlapp` file. Metadata and web resources of an IBM Worklight application deployed on the Worklight Server. Used by the Worklight Server to identify and service mobile applications.

### Adapter files (`.adapter`)

An adapter file (`.adapter`) 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 applications 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 project web archive (WAR) file to be deployed on your application server**

This file contains the default server-specific configurations such as security profiles, server properties, and more. `.wlap` and `.adapter` files use these properties at various stages. Typically, the project WAR file is adapted to the test and production environment, when you deploy the file to your application server. For more information, see “Deploying an IBM Worklight project” on page 665.

## **IBM Worklight projects, 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 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.

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 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 (deprecated in IBM Worklight V6.0.0) 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.
  - For example, since Facebook uses iframes as containers to its apps, you can use the Desktop Browser web page environment to create Facebook apps by setting `https://host:port/apps/services/www/application_name/desktopbrowser/` as the canvas URL in the Facebook dashboard.

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

Option	Description
<b>Hybrid Application</b>	To create a Worklight project with an initial hybrid application.
<b>Inner Application</b>	To create a Worklight project with an initial inner application and point to a built shell component
<b>Shell Component</b>	To create a Worklight project with an initial shell component application
<b>Native Application</b>	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
<b>Add jQuery Mobile</b>	To add jQuery Mobile support to the application. You must identify the directory where the required files for jQuery Mobile are located.
<b>Add Sencha Touch</b>	To add Sencha Touch support to the application. You must identify the directory where the required files for Sencha Touch are located.
<b>Add Dojo Toolkit</b>	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.

## 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**
  - **Worklight Shell Component**

A dialog opens, based on the type of application that you selected.

2. Depending on the selected type of application, set the properties of your application, as described in the following sections.

- 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, Sencha Touch, Dojo Mobile**.

**Note:** If you add jQuery Mobile to your application, and you are using Windows Phone 7.5 (deprecated in IBM Worklight V6.0.0) or Windows Phone 8, you must ensure that the following conditions are met:

- The `$.mobile.allowCrossDomainPages` option is set to true (in jQuery Mobile).
  - An absolute URL is used for file, for example `x-wmapp0:/www/default/app-pages/myPage.html`.
- 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, if a Shell archive exists within your project.
  - 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**.
  - 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, Sencha Touch, Dojo Mobile**.
3. Click **Finish** to save your choices. An application of the type that you selected is now visible in your IBM Worklight project.

## 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 IBM Worklight project, and create your application in it, as described in “Creating an application in an IBM Worklight project” on page 241.
- Create an IBM Worklight project, and your application in it as its first application, as described in “Creating IBM Worklight projects” on page 240

You can build your IBM Worklight application for specific mobile, desktop, and web environments that you can select in Worklight Studio.

- To learn about the available environments, see “IBM Worklight environments” on page 239
- To learn how to set up environments for your IBM Worklight application, see “Setting up a new IBM Worklight environment for your application” on page 262

After you create your IBM 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 ME apps”

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

For more information about how to develop your applications, see “Developing hybrid and web applications” on page 246 and “Developing native applications” on page 331.

### **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.

For more information, see “JavaScript client-side API” on page 489.

### **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 620.

### **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 620.

### **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 620.

## 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 IBM Worklight (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.



```

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
|                 | icudt461.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

```

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 247.

---

## Developing hybrid and web applications

Develop hybrid and web applications as detailed here.

### 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 IBM Worklight, 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
	bin	Artifacts resulting from building adapters, apps, and server-side configuration and libraries
	components	Source code for all shell components belonging to the project
	www	Source code of the Dojo JavaScript framework, if installed as part of Worklight Studio
	server	

		conf	Worklight Server configuration files, such as <code>worklight.properties</code> and <code>authenticationConfig.xml</code>
		java	Java code that must be compiled and packaged into jar files deployable to the Worklight Server
		lib	Pre-compiled libraries that must be deployed to the Worklight Server

## 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 `www` folder

If you installed the Dojo JavaScript framework, the `www` folder contains a minified version of Dojo Mobile libraries. This minified version contains all the Dojo mobile components. If you need to add more Dojo components or Dojo features to your application, see the topic “Creating Dojo-enabled Worklight projects” on page 286.

## Anatomy of an IBM Worklight application

This collection of topics describes the files within an IBM Worklight application

With IBM Worklight, 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. 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
	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 (deprecated in IBM Worklight V6.0.0)
	air		Resources specific to Air
	desktopbrowser		Resources specific to desktop browsers
	mobilewebapp		Web resources specific to mobile web applications
	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.

### **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`
  - 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 Phone 7.5 (deprecated in IBM Worklight V6.0.0), the file name of the splash image is `SplashScreenImage.jpg`, located under `windowsphone\native`. This file must be in .jpg format, with a width of 480 pixels and height of 800 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.

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 Phone 7.5 (deprecated in IBM Worklight V6.0.0):
  1. Replace the default image in windowsphone\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 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 39. 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	
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- <i>ldpi</i> <i>-hdpi -or other options</i>

Table 39. Application icons (continued)

Environment	File name	Description	Location of default icon	Location of overriding icon
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 <a href="https://developer.blackberry.com/devzone/design/application_icons.html">https://developer.blackberry.com/devzone/design/application_icons.html</a> .	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: <ul style="list-style-type: none"> <li>• Non-Retina display: 72 by 72 pixels</li> <li>• Retina display: 144 by 144 pixels</li> </ul> iOS7: <ul style="list-style-type: none"> <li>• Non-Retina display: 76 by 76 pixels</li> <li>• Retina display: 152 by 152 pixels</li> </ul>	ipad\native\nresources	\ipad\nativeResources\nResources

Table 39. 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: <ul style="list-style-type: none"> <li>• Non-retina display: 57 by 57 pixels</li> <li>• Retina display: 114 by 114 pixels</li> </ul> iOS7: <ul style="list-style-type: none"> <li>• 120 by 120 pixels</li> </ul>	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	
Windows Phone 7.5	Background.png ApplicationIcon.png	Both icons are used to identify the application.  Background.png is displayed on the device home screen, and must be 173 by 173 pixels.  ApplicationIcon.png is displayed in the list of applications, and must be 62 by 62 pixels.	windowsphone\native	

Table 39. Application icons (continued)

Environment	File name	Description	Location of default icon	Location of overriding icon
Windows 8	storelogo.png logo.png smalllogo.png	<p>All icons are used to identify the application.</p> <p>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.</p> <p>logo.png represents the square tile image of the app in the Start screen. The image must be 150 by 150 pixels.</p> <p>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.</p>	windows8\native\images	

### The application descriptor

The application descriptor is a metadata file that is used to define various aspects of the application. It is 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 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-inks
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 IBM Worklight 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.

```
<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.

```
<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" />
<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 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 Worklight Server with which the app communicates. To enable it for the 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 259 for details of this element.

```

<android version="1.0" sharedUserId="com.mycompany">
<pushSender key="AIzaSyDcSz70vxQwr7XKg_0Ud0aNJz0pYXuaS_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 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 259 for details of this element.

```
<windowsPhone8 version="1.0">
<uuid>87e096eb-6882-4cef-9f66-e68769de3926</uuid>
<pushSender/>
<allowedDomainsForRemoteImages>
  <domain>http://icons.aniboomb.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 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. You also use the `<pushsender>` subelement to set attributes for authenticated push. For more information, see “Windows Phone 8 push notifications” on page 415.
- 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.

**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 (deprecated in IBM Worklight V6.0.0) on the device. It is automatically generated by the 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 330 for more details.

**<uuid>** Use the <uuid> subelement to uniquely identify a Windows 8 application. It is automatically generated by the 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"/>
<height>410</height>
<width>264</width></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 328 for more details.
- Use the <certificate> element to sign the AIR application before you publish it. See “Signing Adobe AIR applications” on page 329 for more details.
- The <height> element is used to determine the height of the application on desktop environments.
- The <width> element is used to set the width of the application on desktop environments.

```
<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.

```
</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 323

### The `<features>` element

In Worklight V6.0.0, you can control the features included in your application. This ability gives you a finer degree of control over the size of your application, and thus its ability to download and start quickly.

In the `application-descriptor.xml` file, the `<features>` element is added automatically when the application is first created, but with no contents. If you later add JSONStore features and want to include these resources in the application build, you can edit the `<features>` element. You can do this using the Application Descriptor Editor or in XML, as shown in the following example:

```
<application xmlns="http://www.worklight.com/application-descriptor" id="MyProj" platformVersion="6.0.0">
  ...
  <features>
    <JSONStore/>
  </features>
</application>
```

If you do not include JSONStore in the build but use it in your code, an error is raised when you run the app, and you can add it to the `<features>` element with a QuickFix.

If you find during testing that your application does not actually use the JSONStore resources, you can reduce the size of your Android app by removing the JSONStore argument from the `<features>` element. (The JSONStore resources are still included in your iOS application builds.) When a feature is added or removed, the application must be built again before the change takes effect.

For more information about the `<features>` element, see “Including and excluding application features” on page 343.

### The `<cacheManifest>` element

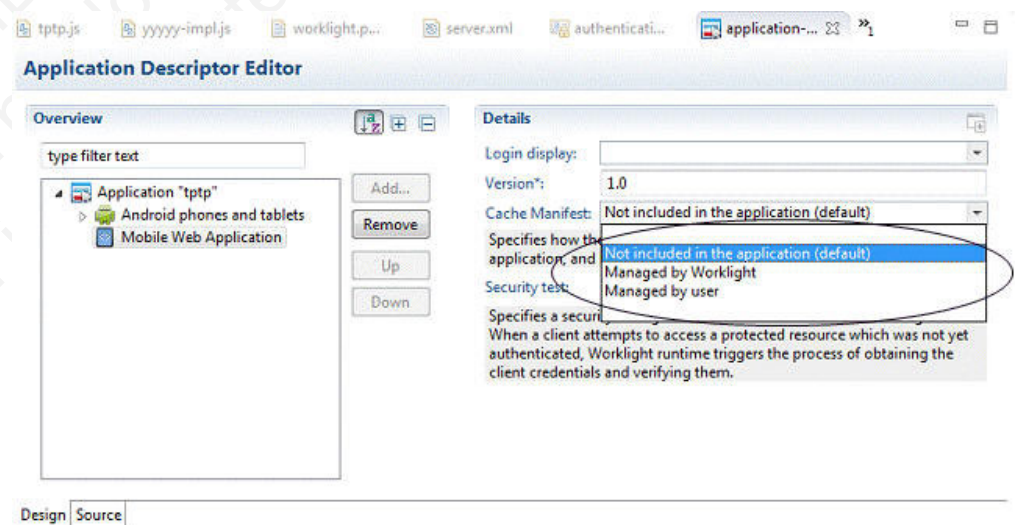
A new element now exists in the Application Descriptor (`application-descriptor.xml`) named `<cacheManifest>`. Using this element, you can manage and edit the contents of the application cache for Desktop Browser and Mobile Web applications, and thus control which resources are fetched when the application starts. Unused resources such as large images or unneeded files included in the Cache Manifest file slow startup time for these applications. By editing this file, you can remove these unnecessary resources.

The cacheManifest element accepts three values, as shown in the following table.

Table 40. cacheManifest properties

Property	Description
<b>generated</b>	<p>In this mode, the Worklight Studio builder generates a default Cache Manifest and includes it in the application's HTML files. The default Cache Manifest is generated depending on the environment:</p> <ul style="list-style-type: none"> <li>• For Desktop Browser environments – all resources are under NETWORK, which means: no cache at all.</li> <li>• For Mobile Web environments – all resources are under CACHE, which means: cache everything.</li> </ul> <p>In <b>generated</b> mode, in addition to creating the Cache Manifest, the builder creates a backup of the previous Cache Manifest, called worklight.manifest.bak. This file is overwritten in every build.</p>
<b>no-use</b>	<p>In this mode (which is the default), the Cache Manifest is not included in the application's HTML files. This setting means that there is no Cache Manifest and that decisions about which resources are cached are up to the browser.</p>
<b>user</b>	<p>In this mode, the Worklight Studio builder does not generate the Cache Manifest, but it does include it in the application's HTML files. This setting means that the user must maintain the Cache Manifest manually.</p>

If you open the Application Descriptor in Design view, you can view and set the current mode of the <cacheManifest> attribute with the DDE editor:



In this view, each of these attribute options is given a description:

- **Not Included in the application (default)** corresponds to **no-use** mode

- **Managed by Worklight** corresponds to **generated** mode
- **Managed by user** corresponds to **user** mode

You can also edit the value in the `<cacheManifest>` element of the `application-descriptor.xml` file itself, as shown in the following code sample:

```
<application xmlns="http://www.worklight.com/application-descriptor" id="MyProj" platformVersion="6.0.0">
  ...
  <mobileWebApp cacheManifest="generated" version="1.0"/>
  <desktopBrowser cacheManifest="generated" version="1.0"/>
</application>
```

For more information about the Cache Manifest, see “Application cache management in Desktop Browser and Mobile Web apps” on page 346.

### Deprecated elements

The following elements are deprecated since IBM Worklight V4.1.3:

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

The following elements are deprecated since IBM Worklight V5.0:

```
<worklightRootURL>
```

The following elements are deprecated since IBM Worklight V5.0.0.3:

```
<usage>
```

The following element (a replacement for `<worklightRootURL>`) is removed in IBM Worklight V6.0.0:

```
<worklightServerRootURL>
```

### 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 Chapter 3, “Tutorials and samples,” on page 25.

## Setting up a new IBM Worklight environment for your application

With Worklight Studio, you can build applications for different mobile, desktop, or web environments within your IBM Worklight project.

### About this task

With Worklight Studio, you can add environments to your IBM Worklight application, and write code that is specific to one or several mobile, desktop, or web environments. If you want to create a version of your IBM Worklight application for a specific platform, you must add the environment that corresponds



to that platform to your application. For example, if you want to create an iPhone version of your IBM Worklight application, you must add an iPhone environment. When you add an environment to your application, a new folder for that environment is created. This folder contains the resources of the new environment:

**images:** This folder contains images that override the images in the common environment that have the same name.

**css:** This folder contains files that extend or override the CSS files in the common environment.

**js:** This folder contains JavaScript files that extend the common application instance JavaScript object. The class that is defined in this environment folder extends the common app class.

**HTML:** This HTML file overrides the HTML file in the common environment that have the same name.

**Note:** The **common** folder in your IBM Worklight application folder contains the code and resources that are common to several environments.

## Procedure

1. Go to your application in Worklight Studio, which is in your IBM Worklight project. To learn how to create a project, see “Creating IBM Worklight projects” on page 240. To learn how to create an application, see “Creating an application in an IBM Worklight project” on page 241

You can see your new application within your IBM Worklight project in the Project Explorer.

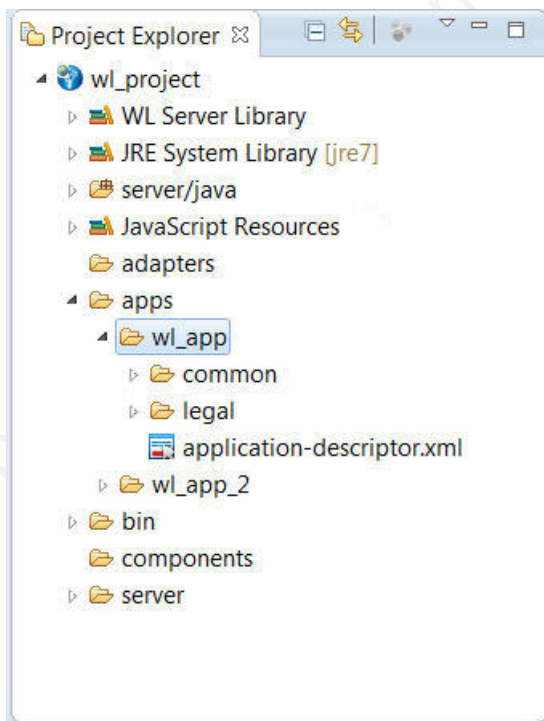


Figure 21. Your IBM Worklight application folder

2. From the menu on top of the screen, click **File > New > Worklight Environment**.

A window opens where you can select the environment that you want to add.

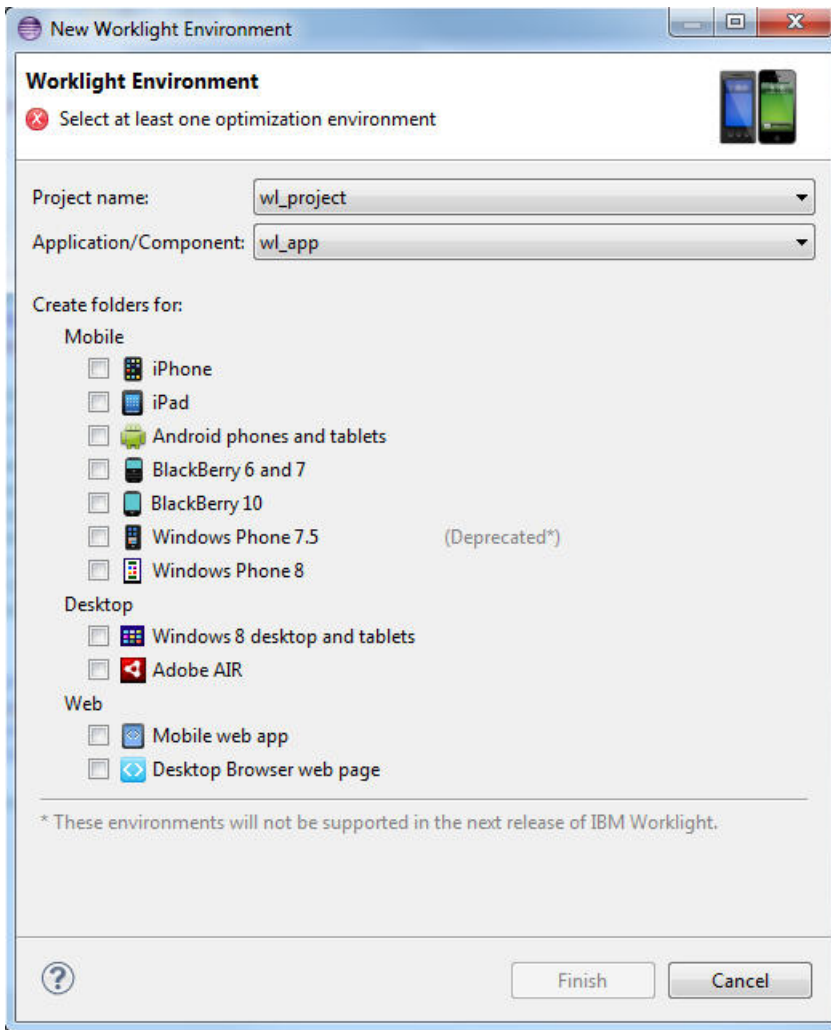


Figure 22. New Worklight Environment window

3. In the **Project name** list, select your IBM Worklight project.
4. In the **Application/Component** list, select your IBM Worklight application.
5. Select the environments that you want to add.

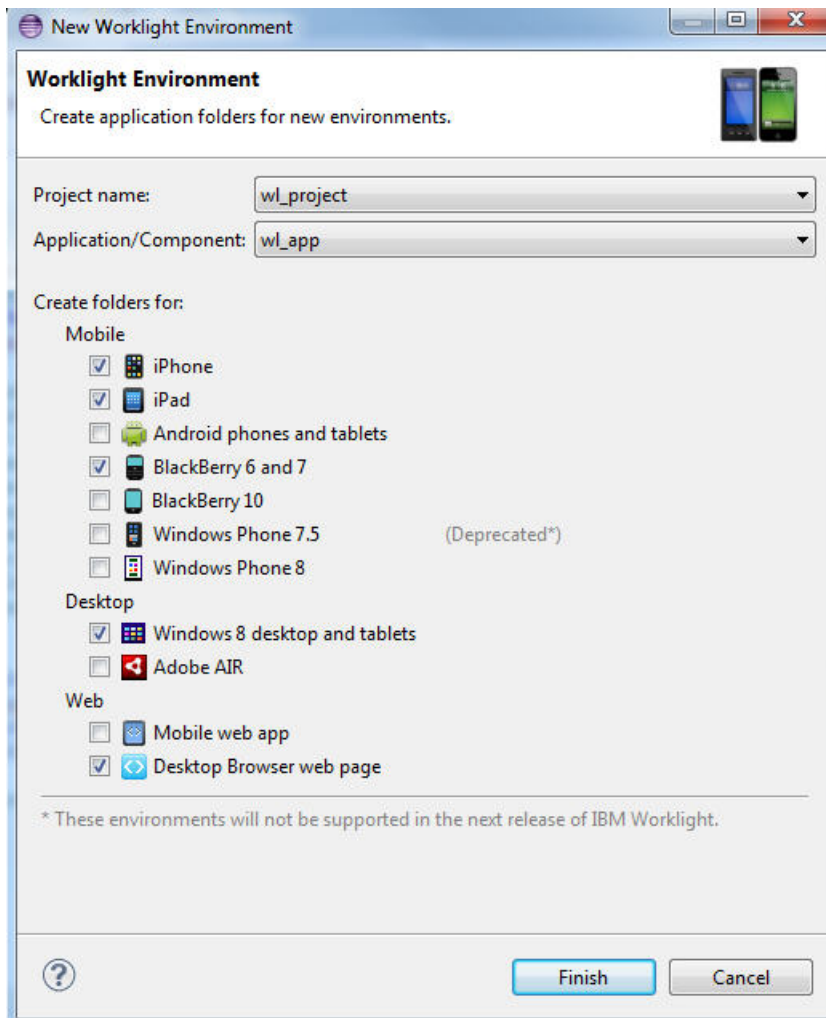


Figure 23. Selecting the mobile, desktop, and web environments that you want to add to your application

You can see the folders corresponding to the environments you added in your application folder.

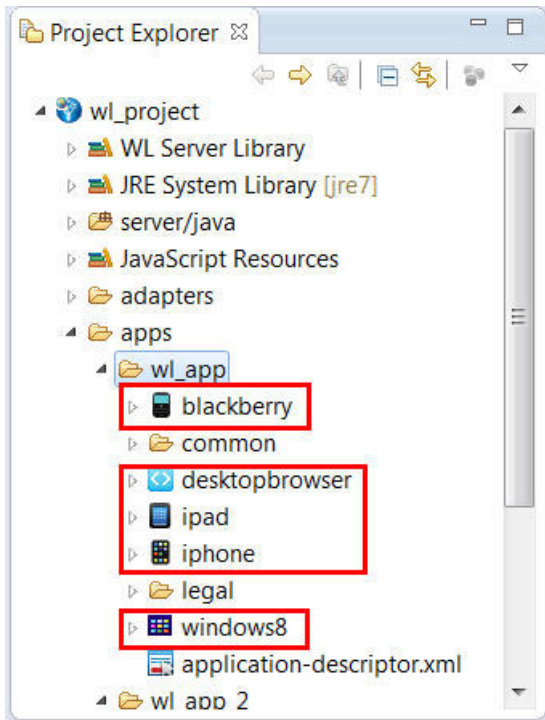


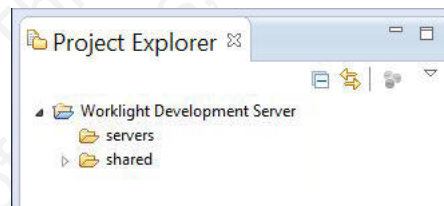
Figure 24. IBM Worklight application folder that contains folders for the environments you selected

## The Worklight Development Server and the Worklight Console

IBM Worklight V6.0.0 introduces changes to the test server used during development and to how it is viewed in the Worklight Console.

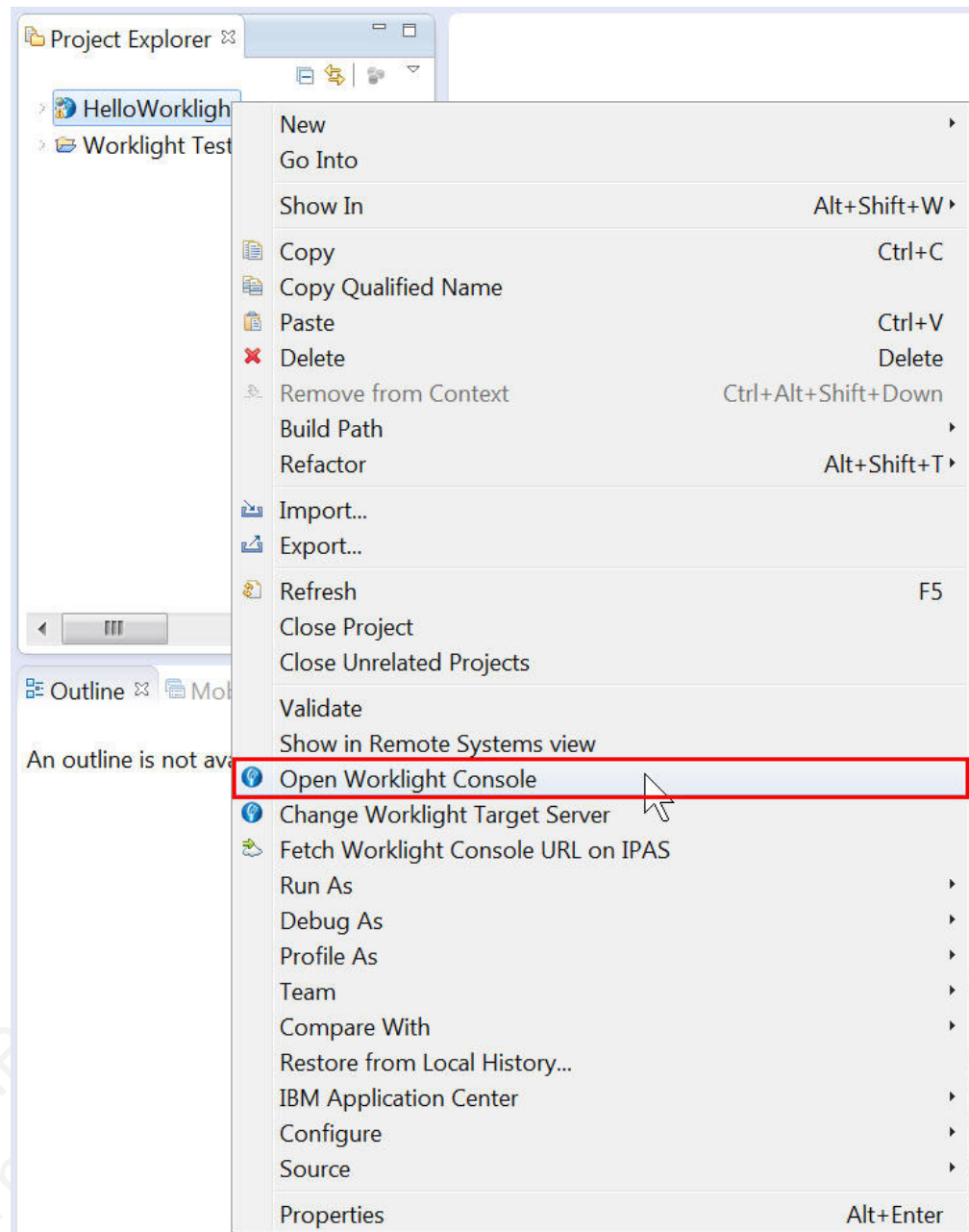
In IBM Worklight V6.0.0, the Jetty server is replaced with an embedded instance of WebSphere Application Server Liberty Profile. This Liberty profile server is installed with Worklight Studio, and becomes the default test server.

As a result, you see a new **Worklight Development Server** element in your Eclipse Project Explorer view, even before you begin creating new projects and working with them.



### Viewing the Worklight Development Server in the Worklight Console

A new menu item in Worklight Studio allows you to open this console even more easily. Right-click the project name, and choose **Open Worklight Console** from the menu, as shown in the following screen capture:



Choosing this menu option opens the Worklight Console for the selected project (context root) in your default browser. You can change the browser in Eclipse by selecting **Window > Preferences > General > Web Browser**.

**Note:** It is possible to work with additional instances of Worklight Server other than the embedded Worklight Development Server. For example, if you have an additional instance of WebSphere Application Server Liberty Profile or Apache Tomcat that is installed in your development environment, you can change the context root to the correct server when building, deploying, or viewing its actions in the console. To do this, you use the **Change Worklight Target Server** menu command described in “Working with multiple Worklight Servers in Worklight Studio” on page 268.

## Creating a URL to access the Worklight Console directly

In previous releases of Worklight Studio, the Worklight Console URL used to view the development test server in a browser had the following syntax:

```
http://localhost:<port>/console
```

The default <port> used was usually 8080, although that was often changed according to developers' needs.

In Worklight Studio V6.0.0, the syntax for the Worklight Console URL uses the following format, which now includes the Worklight project name to provide a *context root*:

```
http://localhost:<port>/<projectName>/console
```

The default <port> after Worklight Studio installation is now 10080. So the Worklight Console URL for a project named **myProject** becomes:

```
http://localhost:10080/myProject/console
```

**Note:** The **Open Worklight Console** menu command in Worklight Studio can only point to one instance of Worklight Server at a time. It will display the console for the server instance for which the context root was set using the **Change Worklight Target Server** command. If you need to work with several different servers for test purposes (for example, one instance of Liberty profile and another of Apache Tomcat), you should save the URLs for these servers' Worklight Consoles as bookmarks in your default browser.

## Working with multiple Worklight Servers in Worklight Studio

Information about how to work in Worklight Studio in a development environment with multiple instances of Worklight Server.

As noted in “The Worklight Development Server and the Worklight Console” on page 266, in IBM Worklight V6.0.0 the embedded Jetty test server was replaced with an instance of WebSphere Application Server Liberty Profile. This server is referred to as the *Worklight Development Server*, and is associated with Worklight projects as the default development server. A new **Open Worklight Console** menu item enables you to view it in the Worklight Console. You can think of this instance of Liberty profile as the *embedded* or *internal* development server.

Worklight Studio can, however, also work with additional *external* Worklight Servers, for example, an instance of Liberty profile or Apache Tomcat that are installed on your development computer. These external servers are defined in Eclipse's **Servers** view. This topic covers the information that you need to know to work with these external servers.

### Starting and stopping Worklight Server

Because Worklight Server V6.0.0 can support multiple Worklight projects, there are no longer **Start Server** and **Stop Server** menu options that are associated directly to the Worklight project. Instead, the server that is associated with a Worklight project is started automatically (if the server is not already running) when you perform an action against that server or adapter. For example, the target server starts when you use the Worklight Studio command **Build All and Deploy**.



## Path to the Worklight Development Server and its console

As noted above, the default server that is associated with Worklight projects is the embedded Worklight Development Server. The default path for this server is: `http://localhost:10080/PROJECT_NAME`. The path to the Worklight Console for this embedded server is: `http://localhost:10080/PROJECT_NAME/console`.

There are two consoles now. The first is the **Worklight Console**, which contains the builder and plugin logs. The second is the **Worklight Development Server Console**, which contains the Worklight Server logs and Liberty profile logs. For more information about setting logging levels for these consoles, see “Configuring logging in the development server” on page 836.

## Working with multiple development servers

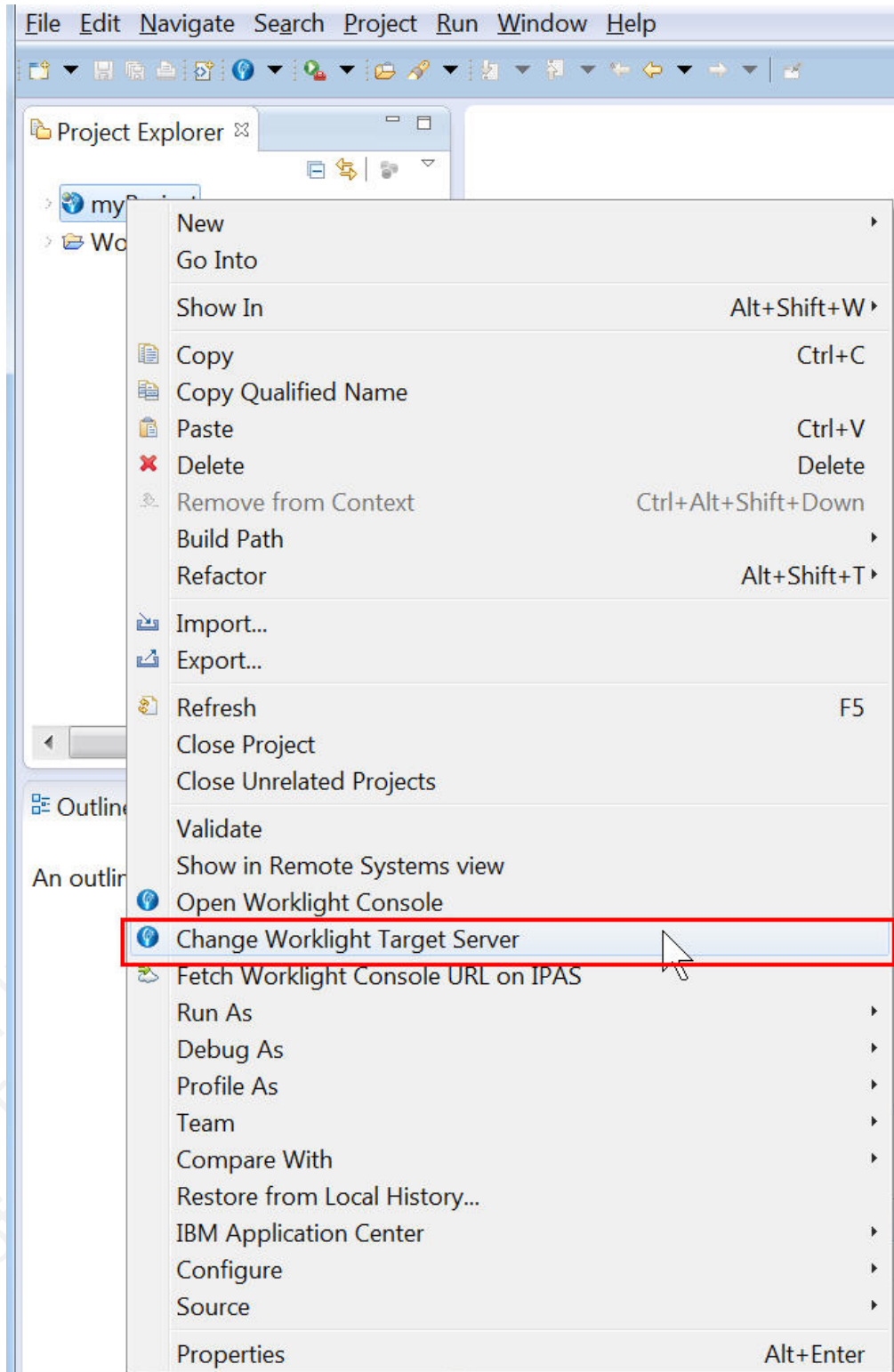
You can create and run multiple Worklight projects against the same Worklight Server. Therefore, if you have an additional instance of Liberty profile or Apache Tomcat that is installed in your development environment, you must ensure that the project you are working with is pointing to the correct server when building, deploying, or viewing it in the console.

Every change that is made to the project source that is related to the project WAR (under the `/server` folder) is automatically built and deployed to the current target server. The database connector JAR files and Worklight JAR file are also automatically deployed to this target server when you deploy the WAR file. That means that the project WAR (not applications or adapters) is always updated on the target server. Every time that the project WAR is built, it also gets deployed to the server associated with that project.

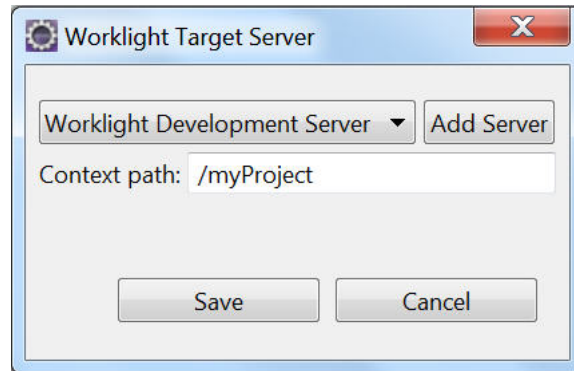
**Note:** The status of the server and its projects as it appears in the Eclipse **Servers** view does not always reflect its current status. This is a known issue.

## Changing the Worklight Server associated with a project

You can change the Worklight project *context root* (which Worklight Server it is associated with) by right-clicking the project and selecting **Change Worklight Target Server**.



This action displays the following window:



If the Worklight Server instance you want to associate with this project is visible in the drop-down list, select it and click **Save**. The outcome of this action is:

1. The project's WAR file is automatically updated with the new context root value when you click **Save**.
2. After rebuilding and deploying the application, the new context root is also saved in the client-side files.

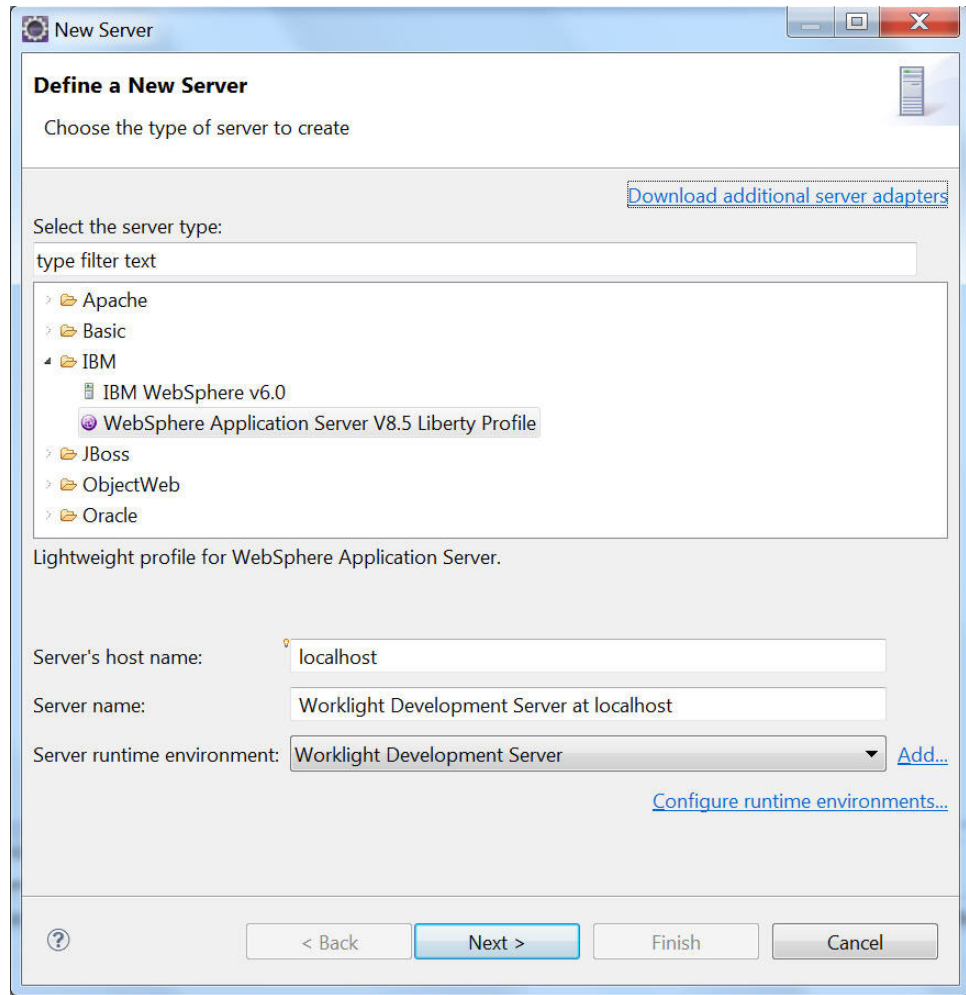
The selected server now becomes your default development Worklight Server. Using the **Change Worklight Target Server** command to choose a different Worklight Server also changes the URL under the **Open Worklight Console** menu command, so that it now points to the new server.

**Note:** If the Worklight Server instance you want is not displayed in the list on this Worklight Target Server window, use the following procedure to add it.

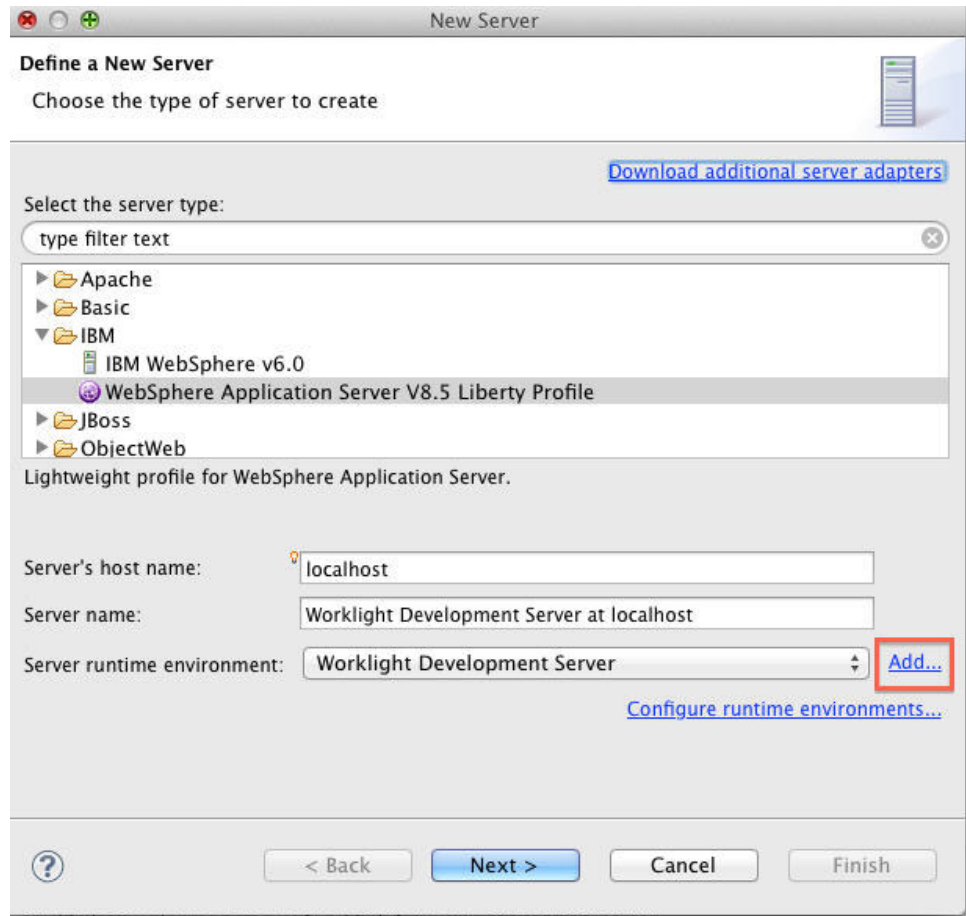
### Adding a new external Worklight Server

If the Worklight Server instance you want to select is not visible in the Worklight Target Server window, you can add a new Worklight Server by using the following procedure. In this example, the user is creating a new server entry for an instance of WebSphere Application Server Liberty Profile that is installed on his development computer.

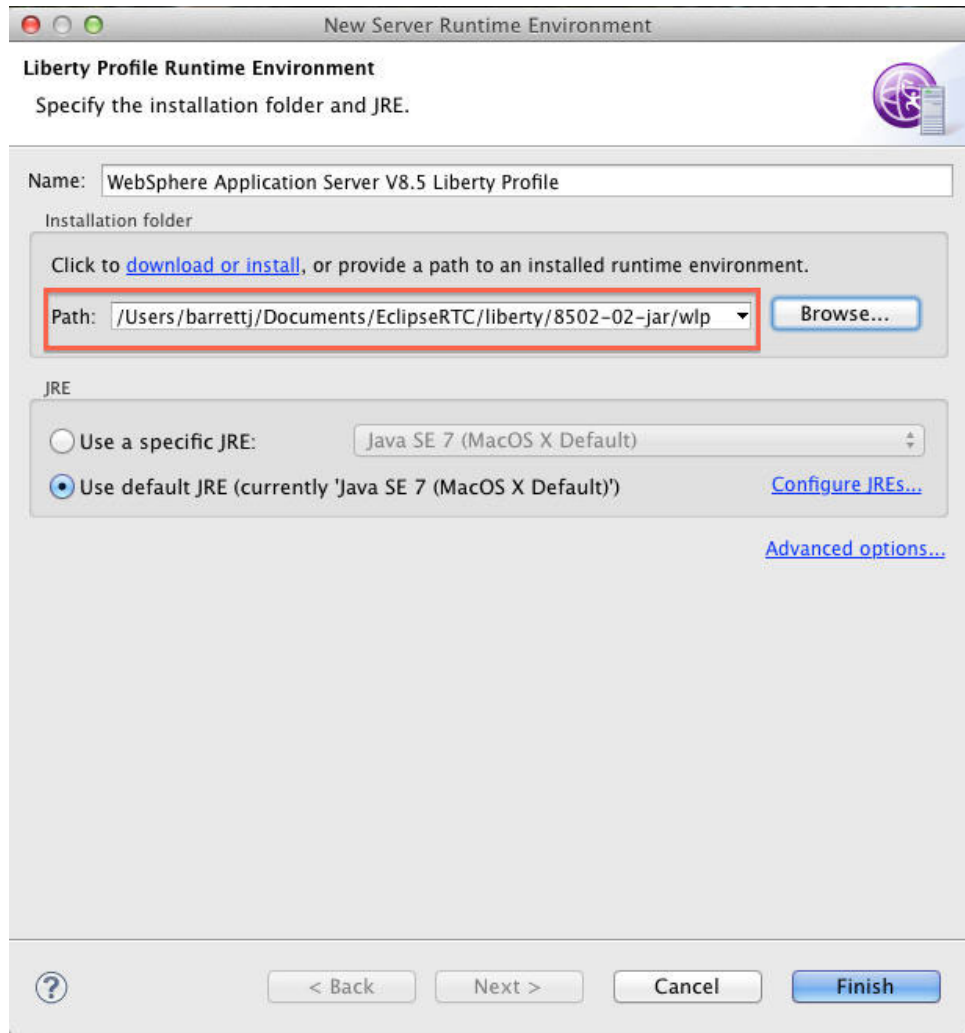
1. First, on the Worklight Target Server window, click **Add Server** to display the following window:



2. Select the server type that you want (in this example, **WebSphere Application Server V8.5 Liberty Profile**), and click **Add** (highlighted in the following screen capture):

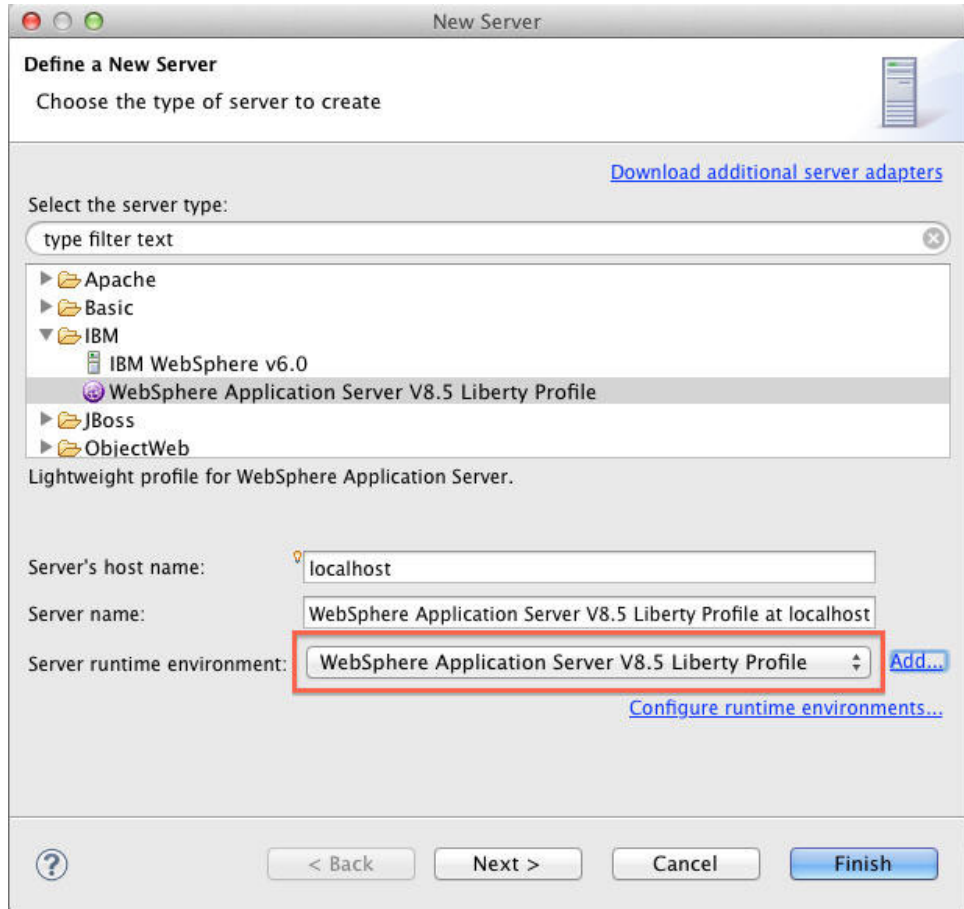


3. On the resulting screen, set **Path** to point to the directory containing the new external Liberty profile server.

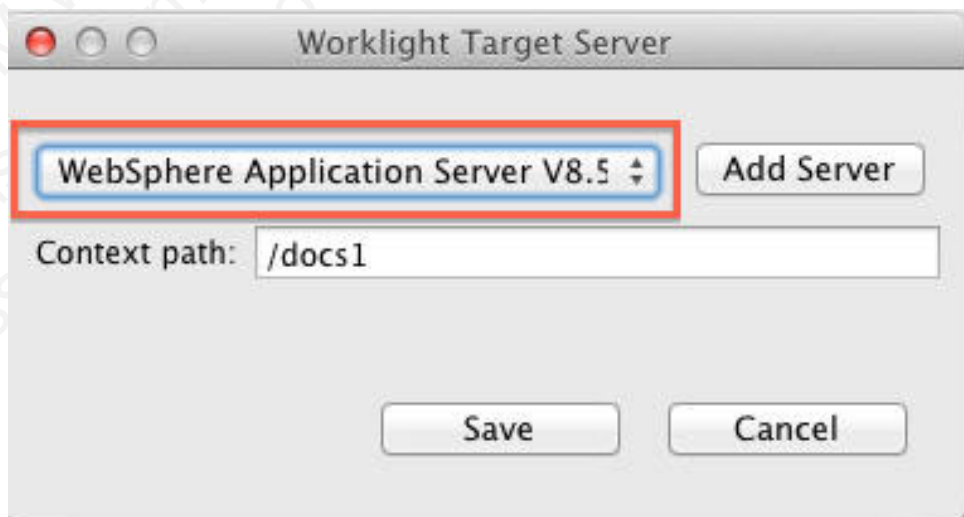


4. After you add the new server, it displays under Server runtime environment.





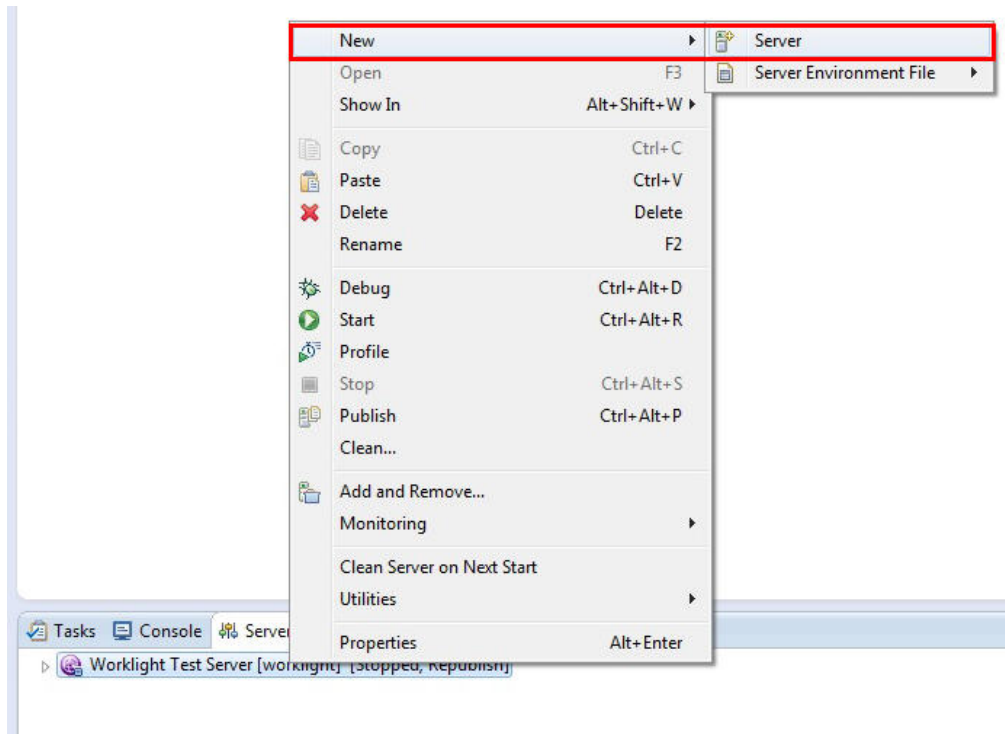
5. The new external server now is displayed as the default in the Worklight Target Server window:



If you select the new server on the Worklight Target Server window, it becomes the default target server, and all builds, deployments, and updates of the project WAR files will go there.

An alternate method of reaching this New Server window is to right-click the entry for an existing server in the Eclipse Servers view and select **New > Server** from the

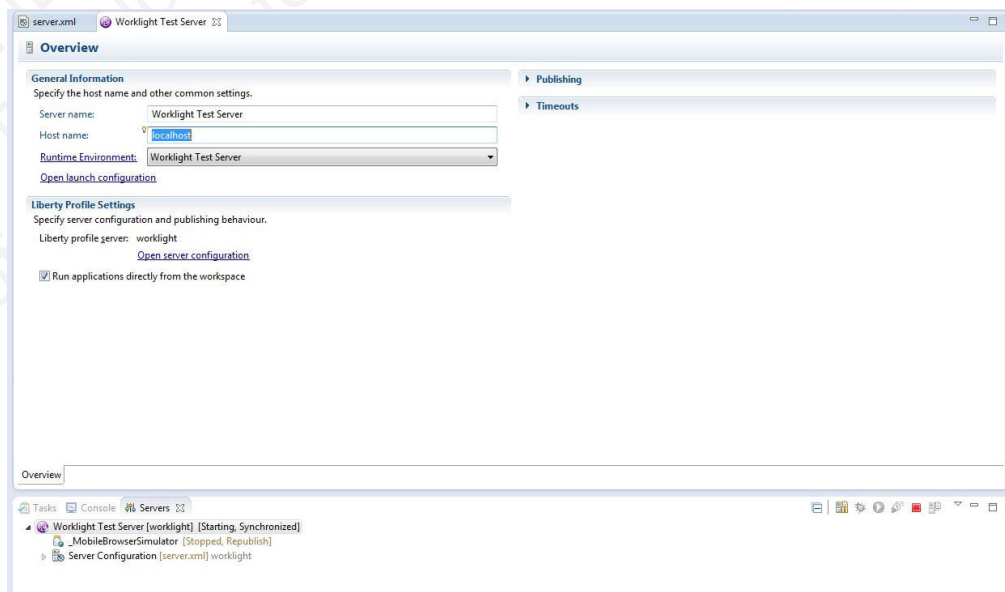
menu, as shown in the following screen capture:



On the New Server window, select the type of server you want to add and click **Next**. Continue with the remaining screens of the New Server wizard to define your new server.

## Setting the port for new Worklight Servers

When your new server is added, you can see it in the Eclipse **Servers** view. When you double-click it, you can view an **Overview** page on which you can change the **Server Name**, change the **Host name**, and other settings.



## For WebSphere Application Server Liberty Profile:

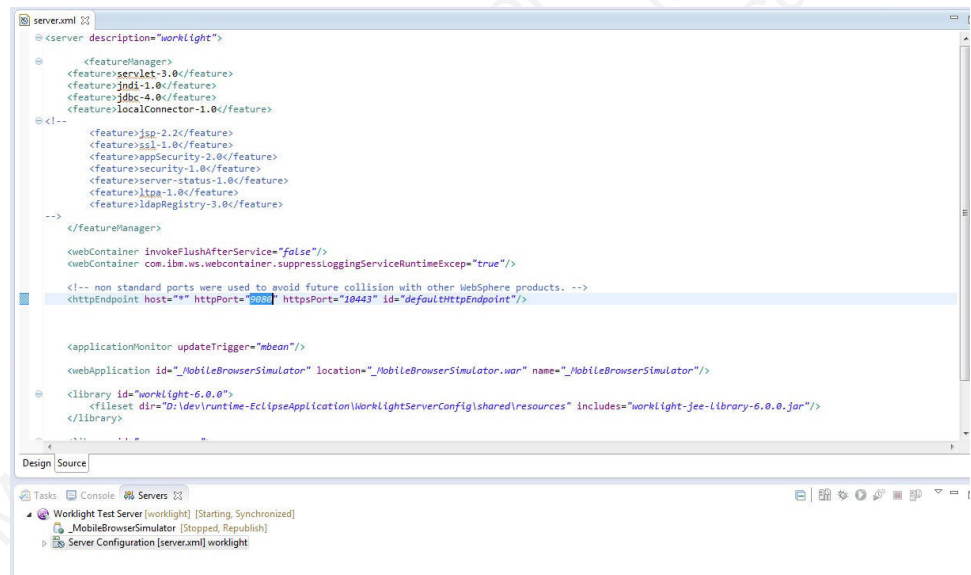
When you connect a project in Worklight Studio to an existing Liberty profile server (not the Worklight Development Server), you must check one thing before you attempt to build and deploy Worklight applications:

- In the target server's `server.xml` file, inside the `httpEndpoint` element, make sure that the Liberty server listens on an external network interface host. Either use a wildcard symbol (for example, `host="*"`) or use a true public listening IP. Do not use `localhost`.

Any change that is done directly on the Liberty `server.xml` and related to the Liberty configuration causes a server restart.

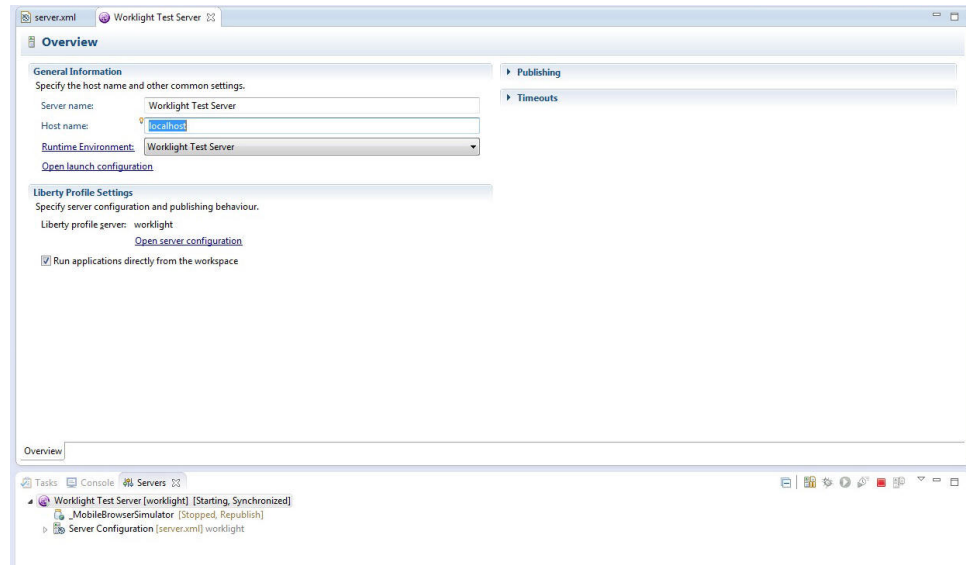
## For Apache Tomcat:

1. To change the port that Tomcat runs on, go to Eclipse **Servers** view, double-click the Tomcat server, and edit the **HTTP port**. Any change that is done directly in this screen requires a restart of Tomcat.



```
server.xml
<server description="worklight">
  <featureManager>
    <feature>genJlet-3.0</feature>
    <feature>jndi-1.0</feature>
    <feature>jdbc-4.0</feature>
    <feature>localConnector-1.0</feature>
  </featureManager>
  <!--
    <feature>jsp-2.2</feature>
    <feature>ssl-1.0</feature>
    <feature>appSecurity-2.0</feature>
    <feature>security-1.0</feature>
    <feature>server-status-1.0</feature>
    <feature>lpa-1.0</feature>
    <feature>ldapRegistry-3.0</feature>
  -->
  </featureManager>
  <webContainer invokeFlushAfterService="false"/>
  <webContainer com.ibm.ws.webcontainer.suppressLoggingServiceRuntimeExcep="true"/>
  <!-- non standard ports were used to avoid future collision with other WebSphere products. -->
  <httpEndpoint host="*" httpPort="9080" httpsPort="10443" id="defaultHttpEndpoint"/>
  <applicationMonitor updateTrigger="mbean"/>
  <webApplication id="_MobileBrowserSimulator" location="_MobileBrowserSimulator.war" name="_MobileBrowserSimulator"/>
  <library id="worklight-6.0.0">
    <fileset dir="D:\dev\runtime-EclipseApplication\WorklightServerConfig\shared\resources" includes="worklight-je-6.0.0.jar"/>
  </library>
</server>
```

2. To change the host name that Tomcat runs on, go to Eclipse **Servers** view, double-click the Tomcat server, and edit the **Host name**.



Both changes require a restart of Tomcat.

## Writing server-side Java code in a Worklight project

Server-side Java code can be added to a Worklight project under the `<project>/server/java` folder. If that code uses specific server runtime classes, be sure to add the Server Runtime Library to the Project Build Path:

1. Right-click the project and select **Build Path > Configure Build Path**.
2. Then, on the **Libraries** tab, click **Add Library**.
3. In the Add Library window, select **Server Runtime** and click **Next**.
4. On the next screen, select a server runtime library to add to the path and click **Finish**.

Otherwise, compilation markers can appear in the Java code.

## Developing user interface of hybrid applications

Develop the user interface of hybrid applications as detailed here.

For more information about the *Rich Page Editor*, *Testing mobile web applications*, and *Previewing your Worklight applications*, expand the entry for this topic in the **Contents** panel and see those topics listed there.

### Using common UI controls

You can use a JavaScript API to invoke common user-interface controls, regardless of the environment.

With IBM Worklight, you can use a JavaScript API to invoke user-interface controls that are common to most environments, such as modal pop-up windows, loading screens, or tab bars.

You can use the following API to automatically renders these controls in a native way for each mobile platform:

## WL.BusyIndicator

WL.BusyIndicator implements a common API to display a modal activity indicator. This method uses native implementation on Android, iPhone, and Windows Phone platforms.

For more information about the functions of this API, see “WL.BusyIndicator (object)” on page 524.

## WL.SimpleDialog

WL.SimpleDialog implements a common API for showing a modal dialog window with buttons. This method uses native implementation on mobile platforms. The dialog closes when the user presses any of the buttons.

For more information about the functions of this API, see “WL.SimpleDialog” on page 607.

## WL.TabBar

WL.TabBar implements a common API to support tabbed application navigation with a tab bar component for Android and iOS environments.

For more information about the functions of this API, see “Tab Bar API” on page 613.

## WL.OptionsMenu

WL.OptionsMenu implements a common API to display a menu of options for Android and Windows Phone.

For more information about the functions of this API, see “Options Menu and Application Bar API” on page 600.

**Note:** For more information about common UI controls, see the module *Common UI controls*, under category 5, *Advanced client side development*, in Chapter 3, “Tutorials and samples,” on page 25.

## Using JavaScript toolkits

Learn how to use javascript toolkits such as JQuery, Dojo Mobile, Sencha Touch.

During the development process, you must design and implement the user interface of your application. You can achieve a high level of customization by writing entirely your own CSS style for each component. However, doing so requires a large amount of resources. You can also use existing JavaScript UI frameworks such as jQuery Mobile, Sencha Touch, or Dojo Mobile to optimize your development process.

## Dojo Mobile

IBM Worklight supports Dojo Mobile for building the user interface of your hybrid mobile application. Dojo Mobile is a world class HTML5 open Source mobile JavaScript framework that you can use to develop mobile web and hybrid applications. Dojo Mobile is part of the Dojo Toolkit, which is developed and maintained by the Dojo Foundation. You can find information about Dojo Mobile, including its documentation, at <http://dojotoolkit.org/>.

You can use Dojo Mobile to develop mobile web applications that have the appearance of the native device on iPhone, iPod Touch, iPad, Android, and BlackBerry touch devices.

The version that is supported by IBM Worklight is the Dojo 1.9 version, which is embedded in Worklight Studio. When you create an IBM Worklight hybrid application, you can select Dojo Mobile among several JavaScript toolkit choices. If you select this option, a copy of Dojo Mobile is added in your project, and a Dojo library project is created in your workspace to support advanced usages of Dojo Mobile.

With Worklight Studio you can do the following tasks:

- Create a hybrid application that uses Dojo Mobile. For more information, see “Creating Dojo-enabled Worklight projects” on page 286.
- Create the user interface of your Dojo Mobile application with the Rich Page Editor, which is a WYSIWYG editor that Worklight Studio provides. The Rich Page Editor supports HTML, Dojo Mobile, and JQuery Mobile. For more information, see “Rich Page Editor” on page 290.
- Use predefined application templates to speed up the development of your application. For more information, see “Mobile patterns” on page 301.
- Use all the power of Dojo Mobile through the Dojo library project. For more information, see “Worklight Dojo library project setup” on page 281.
- For information about how to use Dojo to create a globalized IBM Worklight application, and how to achieve this process by using Dojo Mobile, see “Developing globalized hybrid applications” on page 445.
- For information about how to change Dojo versions that are used by your Worklight projects, see “Changing the Dojo version for Worklight projects” on page 287.

## Sencha Touch

With Sencha Touch, developers can build mobile web applications that have the appearance of the native device on iPhone, Android, and BlackBerry touch devices. Sencha Touch is developed and maintained by Sencha Inc. To download the Sencha Touch package, see <http://www.sencha.com/products/touch/>. To begin the development of your application, you need the `sencha-touch.js`, and `sencha-touch.css` files.

## jQuery Mobile

jQuery Mobile is a touch-optimized web framework for smartphones and tablets. You need jQuery to run jQuery Mobile.

You can download the required jQuery Mobile components, which are in the `.js` and `.css` files, at <http://jquerymobile.com/download/>.

**Note:** Worklight Studio also provides a WYSIWYG editor that supports HTML, Dojo Mobile, and JQuery Mobile. You can use this editor to create the JQuery Mobile user interface of your application. For more information, see “Rich Page Editor” on page 290.



### Worklight Dojo library project setup:

IBM Worklight projects that use Dojo contain a small subset of Dojo resources. This subset of Dojo resources is supplemented with resources (that might not be typical within mobile applications) from a separate Dojo library project.

A Dojo library project contains an entire distribution of Dojo and provides a full view of the capabilities that you can use in a Dojo application. This version of Dojo includes both mobile and desktop Dojo resources. You can test your application by using any of the widgets from the Dojo library project. Dojo resources are served to your application from an internal Dojo library project server.

**Important:** Dojo library project resources are not included in Worklight applications. If a Worklight application requires Dojo library project resources, you must place the resources into the Worklight project before production deployment. If your application needs Dojo resources that are in the Dojo library project but not in your Worklight project, then the IDE must be running if you want to test on a device or in the Mobile Browser Simulator. This is so that the Dojo library project server can serve Dojo resources to your application. Worklight Studio must be running and have IP connectivity for you to test externally to Worklight Studio. If the Dojo library project's IP changes, then the app can't access resources from the Dojo library project until it is rebuilt.

The Dojo library project prints the resources that are requested from it by the IBM Worklight application to provide an accurate account of the parts of Dojo that are required for the application to work. This request list is complete only if the user tests every path through the application: the user must run every path that might request resources from the Dojo Library server to ensure that the list is complete.

You must place all of the resources that are directly referenced in the application HTML page (such as css and images) in the www directory within the Worklight project. The Dojo library project provides only the resources that are requested directly by the Dojo loader, such as the JavaScript modules, their template HTML fragments, and associated images.

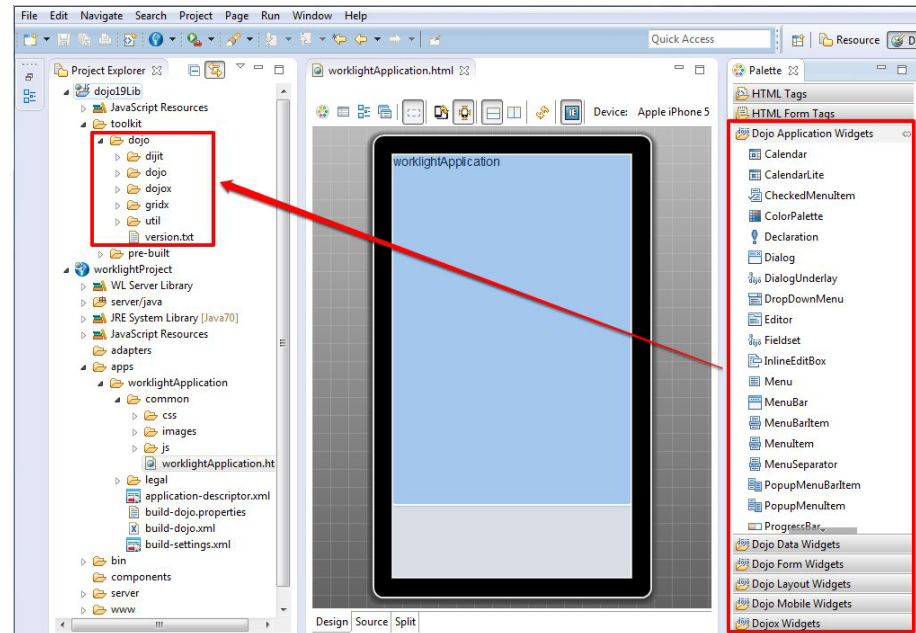
A Worklight project contains only a minimal set of mobile layers and themes. The Worklight project contains the Dojo resources that are deployed as part of the Worklight application. The Dojo that is contained in the Worklight project is optimized for size and includes only the features that are required for a basic mobile application.

Dojo Mobile applications that are built by using Worklight are created by using an optimized set of layers to help the quick development and deployment of your application. The pre-built layers are configured to support numerous locales; however, the application includes the locale for US English only, upon project creation. On some mobile devices, if the locale is configured to anything other than US English, the application can crash due to a defect in Dojo. To correct this, supported locales must be added to the application. The NLS files can be found in your Dojo library project's toolkit/dojo/dojo/nls folder, and they must be copied into your mobile application project's www/dojo/nls folder. After this change, the application can then be deployed to the device with various locale support.

The IBM Worklight Studio tools use the Dojo that is contained in the Worklight project and the associated Dojo library project. The following Worklight Studio tools use the Dojo library project content:

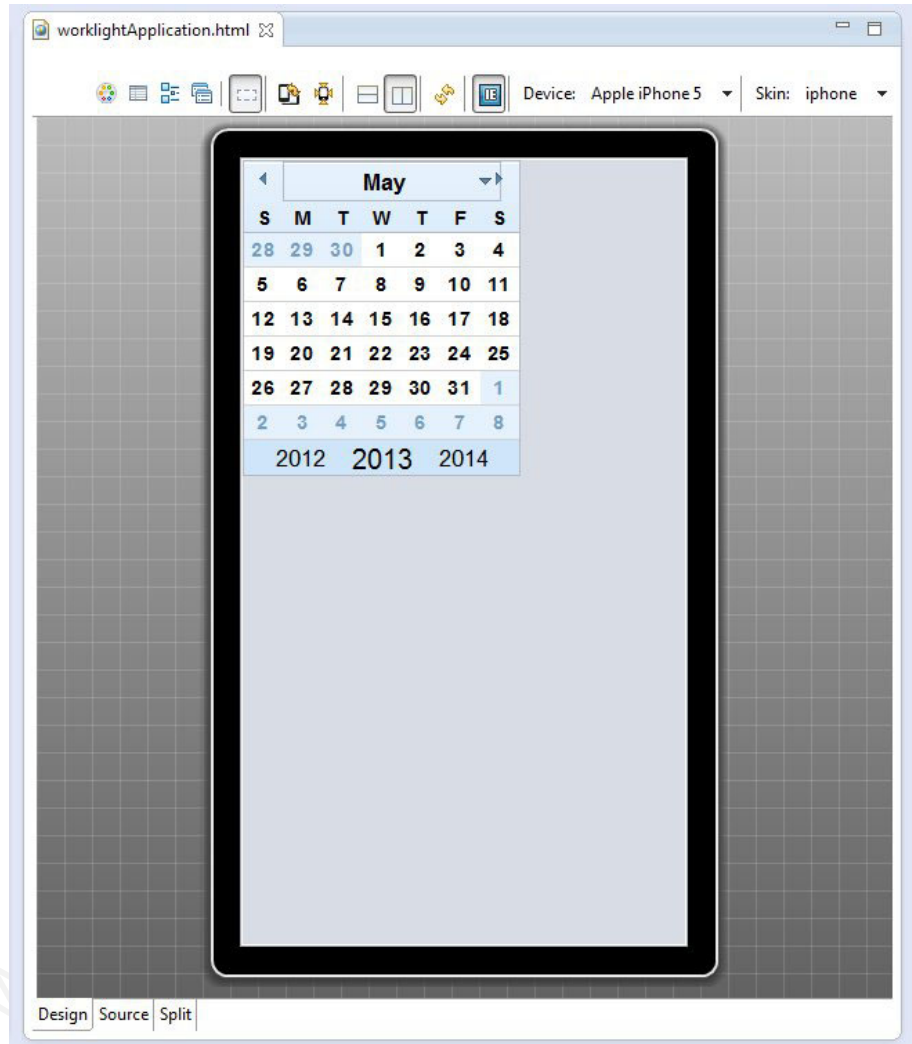
## Rich Page Editor

The Rich Page Editor displays all of the widgets that are available in the Dojo library project.



You can explore and test various Dojo artifacts. It might be necessary for you to run and test your application outside of the Dojo library project. Worklight Studio provides a Console view that shows which Dojo resources are located only in the Dojo library project. For example, if you add the `dijit.Calendar` Dojo widget (that is not part of the mobile layers) to the Worklight application HTML page, Rich Page Editor uses the Dojo library project to display this widget.

**Note:** If you run and test your application on a mobile device or use a device emulator, Worklight Studio must be running to provide Dojo library project resources. To shut down Eclipse and test your application in an environment that is similar to a production environment, you must remove Dojo library project instrumentation. See “Removing Dojo library instrumentation” on page 285.

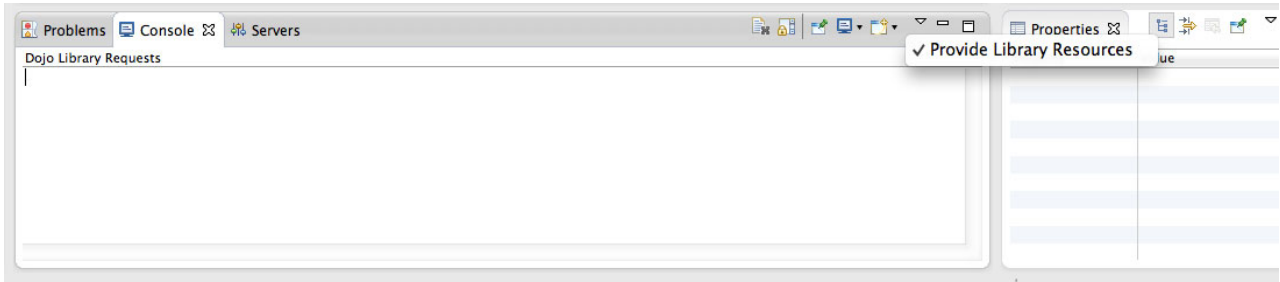


### JavaScript source validation and content assist

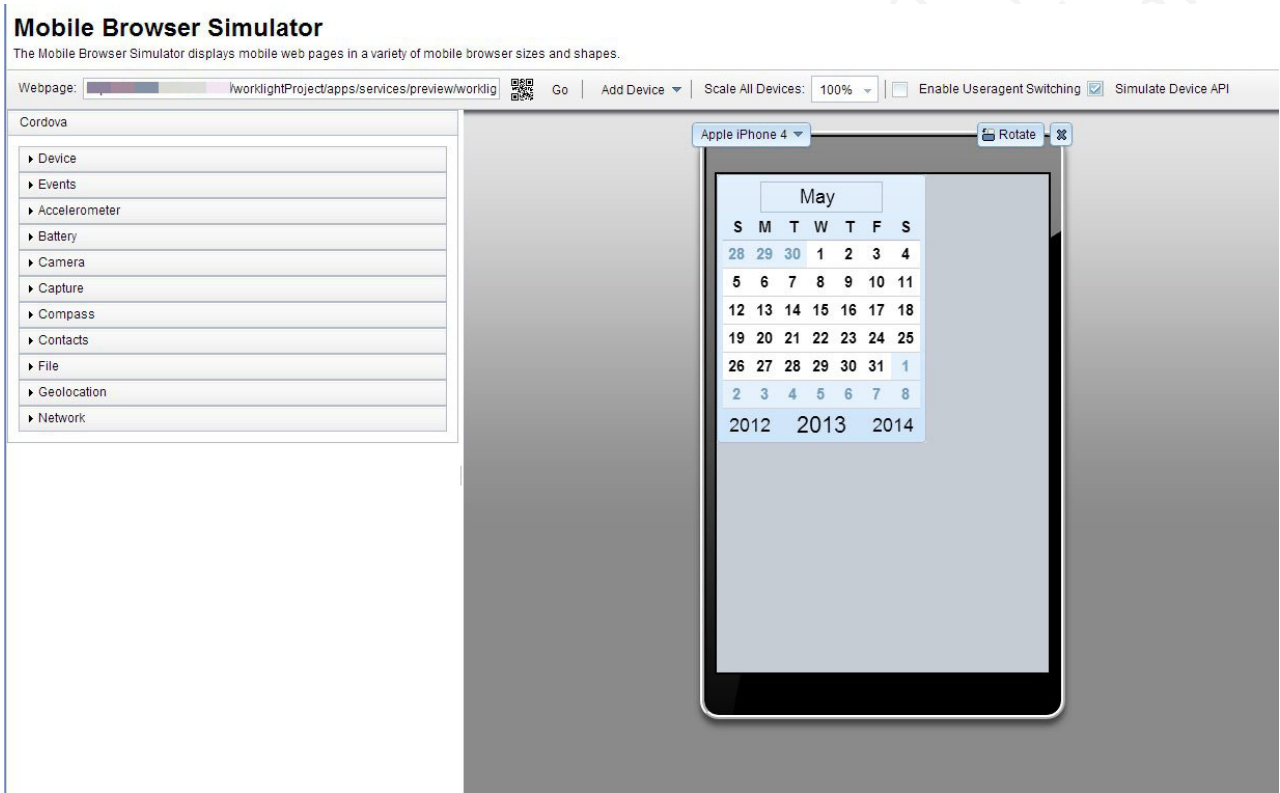
Content assist suggests all of the Dojo widgets that are contained in the Dojo library project, not just the widgets that are contained within the Worklight project. For example, if you are developing in the Source view and use content assist to view all possible values for the data-dojo-type attribute, the `dijit.Calendar` widget is suggested as part of the Dojo library project, not the Worklight application.

### Mobile Browser Simulator


The Mobile Browser Simulator can run with or without the Dojo library project resources. You can use the Console view to turn on and off the Dojo library resources.



Select the **Provide Library Resources** option to specify that you want the Mobile Browser Simulator to use the Dojo library project when it runs. For example, when this option is selected the `dijit.Calendar` widget is displayed correctly.



While the Mobile Browser Simulator is running, the **Dojo Library Requests** Console view shows messages to indicate which resources are served from the Dojo library project.



```
Dojo Library Requests
[2013-05-29 15:31:20] Application 'worklightApplication' requested a missing resource. Providing library resource:
/dojo19Lib/toolkit/dojo/dijit/Calendar.js
[2013-05-29 15:31:20] Application 'worklightApplication' requested a missing resource. Providing library resource:
/dojo19Lib/toolkit/dojo/dijit/CalendarLite.js
[2013-05-29 15:31:20] Application 'worklightApplication' requested a missing resource. Providing library resource:
/dojo19Lib/toolkit/dojo/dijit/templates/Calendar.html
[2013-05-29 15:31:20] Application 'worklightApplication' requested a missing resource. Providing library resource:
/dojo19Lib/toolkit/dojo/dijit/form/DropDownButton.js
[2013-05-29 15:31:20] Application 'worklightApplication' requested a missing resource. Providing library resource:
/dojo19Lib/toolkit/dojo/dijit/form/templates/DropDownButton.html
[2013-05-29 15:31:20] Application 'worklightApplication' requested a missing resource. Providing library resource:
/dojo19Lib/toolkit/dojo/dojo/resources/blank.gif
[2013-05-29 15:31:20] Application 'worklightApplication' requested a missing resource. Providing library resource:
/dojo19Lib/toolkit/dojo/dijit/_HasDropDown.js
```

Examine the logging in the Console view. If the missing resources are required by the final Worklight application, you must add all of the missing resources to the Worklight project. The resources that are logged in the Console view are not available outside of the Worklight Studio development environment.

If you do not select the **Provide Library Resources** option, the Mobile Browser Simulator does not use the Dojo library project when it runs. The Mobile Browser Simulator uses only the resources that are contained in the Worklight project. For example, when this option is selected, the `dijit.Calendar` widget is not displayed. When the Mobile Browser Simulator runs in this mode, the preview emulates the mobile device. The preview provides only the resources that are available to the application when it is deployed to a mobile device. No entries are logged in the **Dojo Library Requests** Console view.

Dojo library projects are only intended for use during development in Worklight Studio. In production, after a Worklight application is deployed to a device, it can access only the Dojo resources that were contained within the application's Worklight project. You must find and move the required resources from the Dojo library project to the Worklight project.

#### *Removing Dojo library instrumentation:*

If you run and test your application outside of the Dojo library project, you must remove the Dojo library instrumentation.

#### **Procedure**

1. Copy all resources that are provided by the Dojo library project and are required by the application into the `www` folder of the IBM Worklight project. The Dojo Console view helps you determine which resources were provided by the Dojo library project. Only the resources in the `www` folder are available when an application is running on a native platform.
2. In the Dojo Library Requests console view, ensure that **Provide Library Resources** is cleared. When **Provide Library Resources** is cleared, the `dojoConfig` mapping that points to the Dojo library project is removed.
3. Run **Preview**. You can complete debugging actions in the Preview window.
4. Build and deploy the application. All required Dojo resources are in the Worklight project `www` folder.

#### **Related concepts:**

“Worklight Dojo library project setup” on page 281

IBM Worklight projects that use Dojo contain a small subset of Dojo resources. This subset of Dojo resources is supplemented with resources (that might not be typical within mobile applications) from a separate Dojo library project.

### Creating Dojo-enabled Worklight projects:

You can create Dojo-enabled Worklight projects that hold all of the resources that are created and used when you develop a Dojo mobile application.

#### Procedure

1. In the main menu, click **File** > **New** > **Worklight Project** to open the New Worklight Project wizard.
2. In the **Name** field, enter a name for your new project.
3. From the list of project templates, click one of the following templates to generate an application for your Worklight project, and then click **Next**.

Template	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.
Native API	To create a Worklight project with a Native API.
Shell Component	To create a Worklight project with an initial shell component application.

4. In the **Application name** field, enter a name for your new application.
5. In the **Dojo installation** section, select the **Add Dojo Toolkit** check box 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.

6. Specify the Dojo library project that you want to use in your new Worklight project:

Option	Description
Select an existing Dojo library project	From the list of available Dojo library projects, select the library that you want to use in your Worklight project. For example, dojoLib.



Option	Description
<p><b>Create a Dojo library project</b></p>	<ol style="list-style-type: none"> <li>1. Click <b>New Dojo Library</b> to open the Dojo Library Setup wizard.</li> <li>2. In the <b>Name</b> field, enter a name for your new Dojo library project.</li> <li>3. Specify the version of Dojo that you want to install.</li> <li>4. Configure how your Dojo library project accesses the Dojo Toolkit and which version of the toolkit to use: <ul style="list-style-type: none"> <li>• Click <b>Provided</b> to select a Dojo Toolkit that is provided with the product.</li> <li>• Click <b>On Disk</b> and then choose one of the following options: <ul style="list-style-type: none"> <li>– Click <b>Archive File</b> to select an archive file of a compressed Dojo distribution. When you click <b>Finish</b>, the contents of the archive file are automatically extracted into your project.</li> <li>– Click <b>Folder</b> to browse to the root Dojo folder in another project in your workspace.</li> </ul> </li> </ul> </li> <li>5. Expand the <b>Select the Dojo components to be included in the project</b> section and select the Dojo components that you want to include in your project.</li> <li>6. Click <b>Finish</b>. The new project is now displayed as an option in the list of available Dojo library projects.</li> </ol>

7. Click **Finish**. Both the Worklight project and the Dojo library project are created.

### Changing the Dojo version for Worklight projects:

You can change the version of Dojo that is used by an existing IBM Worklight project.

#### Before you begin

**Note:** A “pre-built” folder for versions of the Dojo toolkit is provided by IBM Worklight and is officially supported. If you download Dojo from the Dojo website <http://dojotoolkit.org/> and use that for the Dojo library, Step 5 in the following procedure does not happen.

The procedure explains how to upgrade from Dojo that is included with IBM Worklight to a new version of Dojo that is included with IBM Worklight. If you want to take advantage of a version of Dojo in open source that is not yet in IBM Worklight, extra steps are required, see **Alternate Procedure**.

#### Procedure

1. In the Project Explorer view, locate the Worklight project that you want to change the Dojo version for.

2. Right-click the Worklight project and select **Properties** to open the Properties dialog.
3. In the left pane, click **Dojo Toolkit** to open the properties page for the Dojo Toolkit that is used by the selected Worklight project.
4. Choose one of the following options to change the Dojo version that is used by the Worklight project:
  - From the Dojo Library Project list, select an existing Dojo library project that you want to use in your Worklight project.
  - Click **New Dojo Library** to create a Dojo library project for use in your Worklight project.
5. Click **OK**. A dialog box opens prompting you to confirm whether you want to overwrite the existing Dojo layer files with the new Dojo layer files.

**Note:** To avoid unpredictable behavior, use Dojo layer files that match the Dojo library. For example, using Dojo 1.8 layer files with a Dojo 1.9 library might cause unpredictable behavior. If you choose not to overwrite the Dojo layer files now, you can manually overwrite them later using the pre-built files that are contained within the Dojo library project (In the Project Explorer view, expand *Dojo library project* > **toolkit** > **pre-built**).

#### Alternate Procedure

6. Follow the **Procedure** steps 1-4. Then, continue with the following steps.
  - a. Ensure the resources that are being used by the application are copied into the application. Follow the documentation that is outlined for the **Dojo Library Requests** view or Console (depending on Studio version).
  - b. One suitable method involves building new layers from the new version of Dojo so that the same core and mobile UI layers are created from the updates. Manually copy them into the project's `www` folder.
  - c. The alternative way is to remove the references to the core and mobile UI layers ("layers/core-web-layer" and "layers/mobile-ui-layer") from the application's JavaScript file, and use the **Dojo Library Requests** view or Console to find out what's used and then start copying them into the project.

#### 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.

```
<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 `environment/js` folder of the application.

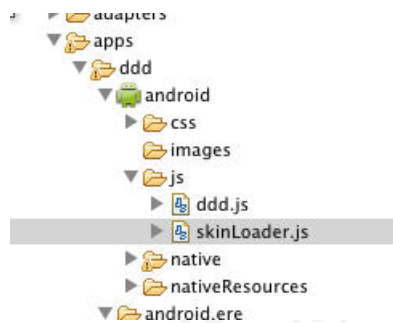


Figure 25. The `skinLoader.js` file

### Deleting a skin

To delete a skin, remove the element that defines the skin from the app descriptor, delete the skin directory, and delete or modify the `skinLoader.js` file.

### Settings page to change the server URL

With IBM Worklight, you can create a settings page to change the URL of the Worklight Server.

With IBM Worklight, you can create a settings page that allows the following:

1. Directs the application to connect to a different Worklight Server by changing the `<protocolca>://<hostname>:<port>/<contextRoot>` values.
2. Loads web resources belonging to a different application or version of the application.

**Note:** This technique works only if the different Worklight Server already exists and these resources or applications are already deployed. This feature is meant only for use in the development environment and not in production.

The settings page is available for the following environments: Android, iPhone, and iPad.

To create the settings page for the supported environments, `<worklightSettings>` is set to `true` in the relevant environment element in the `application-descriptor.xml` file. For example:

```
<iphone version="1.0" bundleId="com.mycompany.myapp">
  <worklightSettings include="true"/>
  <security>
    ...
  </security>
</iphone>
```

## Rich Page Editor

Use Rich Page Editor to easily edit HTML files, add Dojo widgets to HTML pages, and create and edit web pages for mobile devices. Rich Page Editor is a multi-tabbed editor that provides multiple views to show different representations of your page.

## Views

You can use the Source, Design, and Split views in Rich Page Editor to view and work with your files or pages. Each view in Rich Page Editor works with several other views and tools that are included in the Web perspective, including these interface elements:

- Mobile Navigation, Outline, and Properties views
- Toolbar buttons
- Menu bar options
- Pop-up (right-click) menus
- Palette components

Table 41. Rich Page Editor views


Editor view	Description
Source	The Source view helps you to view and work directly with the source code of a file. The Mobile Navigation, Palette, Outline, Page Data, and Properties views have features that supplement the Source view.
Split	The Split view combines the Source and Design views in a split screen view. Changes that you make in one part of the split screen are automatically updated in the other part. You can split the view horizontally or vertically. 

Table 41. Rich Page Editor views (continued)

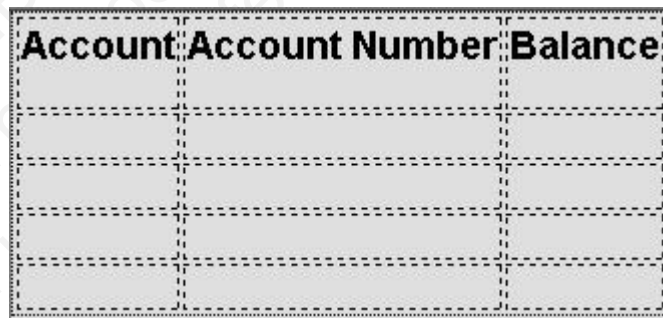
Editor view	Description
Design	<p>The Design view is a WYSIWYG environment. This view helps you to create and work with a file while viewing how your web page and dynamic content might look on a mobile device. You can use this view to visually edit files. For example, the Design view includes features that you can use to complete the following tasks:</p> <ul style="list-style-type: none"> <li>• Drag items from the Palette and Enterprise Explorer views.</li> <li>• Rotate the screen orientation when you use a mobile device profile to view your mobile web page in either portrait or landscape mode.</li> <li>• Scale the mobile device to fit the size of the current Design view. Using this feature, you can see the entire visual canvas without the need to scroll.</li> <li>• View how your page is displayed on different devices by selecting a device from the device list. The selected device specifies the size of the mobile device that you want to view and affects the size of the Design view area.</li> <li>• View how your mobile web page is displayed in different styles. For example, Android, iPhone, or Blackberry. By choosing a particular skin, you can switch in other device-specific styling to view the layout and appearance of your page.</li> </ul> <p><b>Note:</b> The <b>Skin</b> list is available only for Worklight application pages.</p>

### Design Mode editing

You can use the Design Mode editing features of Rich Page Editor to add and edit widgets in the Design view. To enable the Design Mode editing features, click the Design Mode icon.



The following screen capture shows what a table looks like in the Design view of Rich Page Editor when Design Mode is enabled.

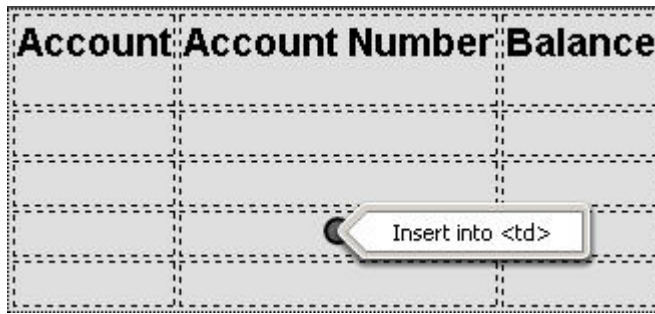


The following screen capture shows what the same table looks like in the Design view of Rich Page Editor when Design Mode is not enabled.

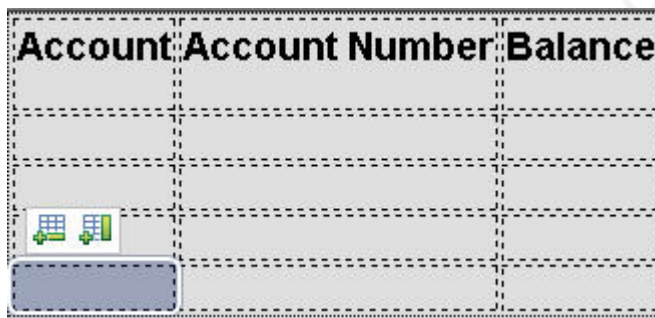


The Design Mode editing features guide the placement of code when you drop a widget on a container widget. Visual cues highlight the possible drop locations and pop-up cues indicate the editing function that is available for the selected widget.

Design Mode also adds dashed borders to empty table cells. For example, dragging a tag from the Palette to a table provides a visual cue for placement:



Selecting a cell in a table opens a pop-up cue that you can use to add a column or row:



#### Browser requirements for Rich Page Editor:

Rich Page Editor uses embedded browsers to produce a visual representation of a web page in the Design view. The browsers that are available in Rich Page Editor and their installation requirements vary according to the platform.

#### Procedure

The following table lists and describes the supported browsers in Rich Page Editor, by platform:

Platform	Supported browsers
Windows	<p><b>Internet Explorer</b> Available for all installations; uses the native browser code in Windows.</p> <p><b>Firefox</b> Firefox support for Windows is embedded in the product and is functionally equivalent to a Firefox version 3.6 installation. Firefox is available only on 32-bit installations of the product.</p>



Platform	Supported browsers
Linux	<p><b>Firefox or WebKit</b></p> <p>The product attempts to locate and use browser code:</p> <ul style="list-style-type: none"> <li>• WebKitGTK+libraries</li> <li>• XULRunner installation</li> </ul> <p>The editor operates with a compatible XULRunner installation that is in the range of Firefox version 3.0 to version 3.6. You can also use WebKitGTK+ libraries with some additional setup. The Firefox indicators are still used in the editor even if you create a webkit-based browser. For more information about setting up the Linux browser, see “Embedded browsers for Linux.”</p>
Mac	<p><b>Safari</b> The native Safari browser is automatically used for products that are available on the Mac platform.</p>

The supported browsers are available from the editor toolbar in both the design and split views.

On the toolbar, click the icon for the browser you want to use. For example in the following screen capture, Firefox, Internet Explorer, and Safari are supported.



### Embedded browsers for Linux:

On Linux systems, to ensure that product features, such as the Rich Page Editor use an appropriate embedded web browser, additional steps to configure the browser are necessary.

Product features that use an embedded web browser might not work correctly if an inappropriate browser is used. Using an inappropriate browser can cause problems such as: scenarios that fail, error messages, or an unexpected output. Product features that use an embedded browser include:

- Rich Page Editor
- Web Browser component
- Welcome page
- Page Designer Preview pane

The Eclipse Standard Widget Toolkit (SWT) supports the following browser types for Linux systems:

- Mozilla (Firefox) through the XULRunner package
- WebKit through WebKitGTK+ shared libraries

The version of Eclipse included in the product determines the default browser type used by SWT. However, you can explicitly configure the default browser type. Only one browser type is available at a time within the product.

- For Eclipse version 3.6.x, Mozilla is the default browser type used by SWT on Linux.
- For Eclipse versions 3.7 and later, the WebKit browser is the default browser on Linux. If suitable WebKit libraries are not found, the XULRunner browser is used.

*Configuring for the WebKit embedded browser:*

A WebKit embedded browser is supplied as a separate installation of the WebKitGTK+ shared libraries, however; these libraries are included in many of the supported Linux distributions.

### **Procedure**

If necessary, install the WebKitGTK+ package onto the system and ensure that it is included on the default library path.

*Configuring for the XULRunner embedded browser:*

The XULRunner package enables Mozilla as the embedded browser. If several XULRunner packages are installed on the same system, a version mismatch can occur even if a specific XULRunner installation is registered as the default version. To clearly define the XULRunner browser and level to be used in your configuration, you must set up an explicit pointer to a XULRunner version.

### **About this task**

The supported XULRunner versions are:

- 1.8.x
- 1.9.2
- 3.6.x

**Note:** The XULRunner package must match the architecture (32-bit or 64-bit) of the product installation.

To download the XULRunner 1.9.2, click one of the following links:

- [XULRunner 32-bit download](#)
- [XULRunner 64-bit download](#)

### **Procedure**

To set up an explicit pointer to a XULRunner version, complete the following steps.

1. In the `eclipse.ini` file included in the product installation, locate the `-vmargs` section.

#### **Note:**

- Some IBM Worklight Studio installations use JRE arguments from the `Worklight.sh` script instead of from the `eclipse.ini` file.
  - If a `Worklight.sh` file is present in the same product directory as the `eclipse.ini` file, add your updates to the `-vmargs` sections of both files.
2. In the `-vmargs` section, add the following JVM system variable where `/home/myuser/xulrunner` is the path to the root of an uncompressed XULRunner package.

```
-Dorg.eclipse.swt.browser.XULRunnerPath=/home/myuser/xulrunner
```

Complete the following step to use the XULRunner browser instead of the WebKit browser.

3. Add the following JVM parameter to the `-vmargs` section at the end of the `eclipse.ini` file. If the `Worklight.sh` file is present, add the same code to the end of this file.

```
-Dorg.eclipse.swt.browser.DefaultType=mozilla
```

### Setting the Rich Page Editor preferences:

You can customize the display of Rich Page Editor by setting the preferences for view shortcuts, pane visibility and layout, design mode, and web browser.

#### Procedure

1. In the main menu, click **Window > Preferences**.
2. Expand **Web > Rich Page Editor**.
3. Specify the default preference settings for Rich Page Editor.

Editor preference	Description
View shortcuts	Specify whether to show or hide the shortcut toolbar buttons in Rich Page Editor for these views: Palette, Properties, Outline, and Mobile Views.
Visible pane	Select which view to show when you open a file with Rich Page Editor. You can choose from these views: Design, Source, and Split.
Pane layout	Set the Split view layout, which is a combination of the Source view and Design view, to split the editor view either horizontally or vertically.
Design mode	<p>Specify whether to enable or disable Design Mode.</p> <p>When Design Mode is available, the editing features help you to add and edit widgets in the Design view of the editor. For example, the editing features guide the placement of code when you drop a widget on a container widget. Visual cues highlight the possible drop locations and pop-up cues indicate the editing function that is available for the selected widget. Design Mode also adds dashed borders to empty table cells.</p> <p>When Design Mode is unavailable, elements in the Design view are displayed exactly as they are shown in the web browser, without any visual aids for editing.</p>
Web browser	Select the web browser in which to show the page that is being edited. <b>Note:</b> The list of available web browsers is dependent on the platform and web browsers that are installed on your computer.

**Tip:** When you are working with Rich Page Editor, you can change these settings from the editor window. To change the view shortcuts, pane layout, design mode, and web browser settings, use the toolbar in the upper-right corner of the editor window. To change the visible pane, use the tabs in the lower-left corner.

4. Optional: To specify that you want to remember these preference settings for each resource, select the **Remember settings for each individual resource** check box.
5. Specify the Smart Highlight settings for Rich Page Editor.

Smart Highlight preference	Description
jQuery	Specify whether to highlight nodes in the Design and Outline views that are matched by jQuery expression selectors in the Source view or Javascript editors. <b>Tip:</b> By default, matched nodes are highlighted in yellow. To change the highlight color, click <b>Change Highlight Color</b> .

6. Click **Apply** and then save your changes by clicking **OK**.

#### Opening web pages in Rich Page Editor:

You can open web pages in Rich Page Editor to edit HTML files, add Dojo widgets to HTML pages, and edit web pages for mobile devices.

#### Before you begin

You must complete the following tasks before you can open a web page in Rich Page Editor:

1. Create a web project.
2. Create a web page.

#### Procedure

In the Enterprise Explorer view, use one of the following methods to open a web page in Rich Page Editor:

- Double-click your web page.
- Right-click your web page and select **Open**.

*Working in the Design and Split views:*

You can use the Design and Split views in Rich Page Editor to edit HTML files in WYSIWYG mode.

When you edit in the Design view, your work reflects the layout and style of the web pages that you build. The Design view removes the added complexity of source tag syntax, navigation, and debugging.

Use the Split view to show both the Design view and the Source view in a split screen view. Changes that you make in one part of the split screen are automatically updated in the other part. You can split the view horizontally or vertically.

## About this task

The design and split views provide full access to the following features:

- Editor menu options
- Pop-up menu actions
- User interface options, such as those in the Styles view
- Drag-and-drop behavior

The Design and Split views also provide support for absolute positioning. You can see the immediate impact of design decisions more quickly than in a text editor. Using these views, you can efficiently and precisely change the composition and attributes of pages, tags, images, and effects.

Many actions available through the editor menus are also available from design element pop-up menus. To access the design element pop-up menus, select a page object, and then right-click the object.

*Working in the Source view:*

You can use the Source view in Rich Page Editor to edit HTML and other markup text, such as embedded JavaScript. Any changes you make in the source view are also reflected in the Design and Split views.

## About this task

You can also show the Source view by opening the Split view. The Split view shows both the Design and Source views, split vertically or horizontally. If you add or update an attribute value in the Source view while the Properties view is visible, the properties are also refreshed.

Table 42. Source view features

Feature	Description
Syntax highlighting	Each tag type uses different highlighting to make it easy to find a specific type of tag for editing. For example, you cannot edit read-only regions of the page which are highlighted in gray.
Unlimited undo and redo	You can incrementally undo and redo every change made to a file for the entire editing session. For text, changes are incremented one character or set of selected characters at a time.
Content assist	Content assist helps you to finish tags or lines of code, and insert macros. The available options in the content assist list are based on the tags that are defined by the tagging standard specified for the file being edited. If content assist does not automatically open, press Ctrl + Space. The content assist text is displayed in a yellow box as you type.
User-defined macros	You can access user-defined templates, which are chunks of predefined code, with content assist to help you add the tagging combinations that are used often.

Table 42. Source view features (continued)

Feature	Description
Element selection	The element selection indicator is located within the vertical border in the left area of the Source view. Based on the location of your cursor, the element selection indicator highlights the line numbers that contain the elements being edited.
Pop-up menu options	You can right-click at a specific position in the editor to open the editor pop-up menu. This menu contains many of the same editing options that are available in the workbench <b>Edit</b> menu.
Drag-and-drop	You can drag objects from the Palette view to the position of the cursor in the Source view.
Copy and paste	You can press <b>Ctrl + C</b> and <b>Ctrl + V</b> to copy and paste a selected tag in the Source view.
Validation	You can configure an option on the preferences page to validate your code as you type: <ol style="list-style-type: none"> <li>From the main menu, select <b>Window &gt; Preferences &gt; General &gt; Editors &gt; Structured Text Editor</b>.</li> <li>On the Structured Text Editor preferences page, select the <b>Report problems as you type</b> check box.</li> </ol>
Customization	You can customize the appearance of the editor on either of the following preferences pages: <ul style="list-style-type: none"> <li><b>Window &gt; Preferences &gt; General &gt; Editors &gt; Editors (or Structured Text Editors)</b></li> <li><b>Window &gt; Preferences &gt; Web &gt; HTML Files &gt; Editor</b></li> </ul>

The HTML 5 specification is supported only in the Source view. For example, you can use content assist to insert the `<canvas>` tag.

You can use any of the following methods to enter, insert, or delete tags and text in the Source view:

- Type the tags directly.
- Use content assist to receive prompts for valid tags.
- Select the menu items.
- Select the toolbar buttons.
- Use the Properties view to change tags.

### Procedure

To edit an HTML file in the Source view:

1. Open the HTML file that you want to work with in the editor.
2. In the **Source** tab, use the available features to edit the code, as required.



**Tip:** You can select attribute values, attribute-value pairs, and entire tag sets by using the double-click feature available in the editor. Use this feature to quickly update, copy, or remove content.

3. At intervals, to see the nesting hierarchies more clearly in the file, format individual elements or the entire document to restore element and attribute indentation.
4. Save the file.

### Creating web pages in Rich Page Editor:

You can create interactive web pages in Rich Page Editor.

#### Before you begin

Before you can create a web page in Rich Page Editor, you must create a web project.

#### Procedure

1. Click **File > New > Web Page** to open the New Web Page wizard.
2. Specify a file name and template for the new web page, and then click **Finish**. Your new web page opens in Rich Page Editor.

### Creating web pages for mobile devices:

You can create interactive web pages that are optimized for mobile devices.

#### Before you begin

Ensure that you complete the following tasks before you create a web page for a mobile device in Rich Page Editor:

1. Create a web project.
2. Set the target device for your web project.
3. Set Rich Page Editor as the default web page editor.

#### Procedure

1. Click **File > New > Web Page** to open the New Web Page wizard.
2. Specify a file name and choose one of the following mobile templates for the new web page:

##### Dojo Mobile HTML template

Sets up the web page for Dojo. Generates content into the web page to prepare the web page for use with its libraries. This content can include:

- JavaScript and CSS includes.
- Basic widgets that are typically required for Dojo Mobile web pages, such as a mobile View widget.

##### jQuery Mobile HTML template

Sets up the web page for jQuery. Generates content into the web page to prepare the web page for use with its libraries. This content can include:

- JavaScript and CSS includes.
- Basic widgets that are typically required for jQuery Mobile web pages, such as a Page widget.

- Optional: To open the New Web Page Options page and add more options to your mobile web page, click **Options**.

Option	Description
<p><b>Set the document type declaration to HTML 5 and cache the page</b></p>	<ol style="list-style-type: none"> <li>From the list of options, click <b>Document Markup</b>.</li> <li>From the <b>Document Type</b> list, select <b>HTML 5</b> to show more options.</li> <li>Specify the icon that is used by mobile devices when users add bookmarks. To select an icon from your workspace, click <b>Browse</b> next to the <b>File href</b> field.</li> <li>Enable browser application caching. In the <b>Manifest Section</b> field, select <b>CACHE</b> and then specify a manifest file. For example, WebContent/META-INF/cache.mf.  HTML 5 application caching ensures performance and availability when the mobile device is offline. For more information about cache manifest files, see the latest HTML5 specification at: <a href="http://dev.w3.org/html5/spec">http://dev.w3.org/html5/spec</a>, and search for "cache manifest".</li> </ol>
<p><b>Set the device detection and stylesheet options</b></p>	<ol style="list-style-type: none"> <li>From the list of options, click <b>Mobile Web Page</b>.</li> <li>Select one of the following options: <ul style="list-style-type: none"> <li><b>Detect device</b> The web page detects the device that shows the content and loads the appropriate CSS by including the script <code>dojox/mobile/deviceTheme.js</code>.</li> <li><b>Select dojox.mobile stylesheet</b> The selected style sheet is loaded by using the <code>&lt;link&gt;</code> tag. You can select one of the following style sheets: <ul style="list-style-type: none"> <li>• blackberry.css</li> <li>• android.css</li> <li>• ipad.css</li> <li>• iphone.css</li> </ul> </li> <li><b>No CSS</b> Use a style sheet other than the ones that are available when you select the dojox.mobile style sheet option. When you specify the No CSS option, you can select <b>Stylesheets</b> from the list of options and add the style sheets that you want to use.</li> </ul> </li> </ol>

- Click **Finish**. Your web page opens in Rich Page Editor.

## Mobile patterns:

Mobile patterns provides templates you can use to develop pages associated with a jQuery or Dojo mobile application. Using mobile patterns accelerates development of your mobile application by providing views common to many mobile applications.

There are many mobile patterns that you can use. They are grouped into four categories. Selecting a category on the Add Mobile Page window displays a list of available patterns associated with the category.

**Lists** Choose from a number of different list formats from simple to complex. You can choose unordered lists patterns or ordered list patterns.

### Authentications

Choose the type of login page for your application that contains only an User ID and password fields. Or, select a template that contains additional input areas or buttons, such as forgot password and register.

### Navigation and search

Choose from a variety of navigation patterns including toolbars, navigation lists, or lists with searchable content.

### Configuration

Choose from blank configuration pages to pages that have predefined configuration items, such as language.

Some mobile patterns are sets where mobile views within the set are appropriately linked. For example, selecting a login page with **Reset password** button, the Reset password template page is also created. When you select a mobile pattern that is a set, you will see all pages in the preview.

Choosing a mobile pattern adds the appropriate code into your application, after which you may alter it as needed.

### Related tasks:

“Adding a mobile pattern to an application”

Use mobile patterns to accelerate development of mobile applications. Select from a predefined list of mobile patterns to quickly add code to your application.

*Adding a mobile pattern to an application:*

Use mobile patterns to accelerate development of mobile applications. Select from a predefined list of mobile patterns to quickly add code to your application.

### Procedure

1. In the Mobile Navigation View, click the plus sign icon.
2. In the add window, select a category and click **Create view from UI pattern**. The available patterns associated with the category are loaded in the view.
3. Select the desired mobile pattern and click **Finish** to insert into your application.

### Related concepts:

“Mobile patterns”

Mobile patterns provides templates you can use to develop pages associated with a jQuery or Dojo mobile application. Using mobile patterns accelerates development of your mobile application by providing views common to many mobile applications.

## Adding elements to web pages from the palette:

You can populate a web page with content by dragging elements from the Palette view to the web page in Rich Page Editor.

### Before you begin

You must complete the following tasks before you can add elements to a web page in Rich Page Editor:

1. Create a web project.
2. Create a web page.

### Procedure

1. In the Enterprise Explorer view, double-click your web page to open it in Rich Page Editor.
2. Add various elements to your web page by dragging objects from the different drawers in the Palette view, such as radio buttons, check boxes, and submit buttons.

**Tip:** In the Web perspective, the Palette view is located by default on the right side of the workbench, underneath the Outline and Snippets views.

3. You can select multiple elements by pressing Ctrl and then performing actions on the selected elements from the menu, such as copy, paste, or delete.
4. When you finish adding elements to your web page, save your changes by pressing Ctrl + S.

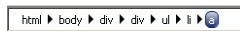
### Properties view associated with Rich Page Editor:

The Properties view that is associated with Rich Page Editor displays specific information for the currently selected tag in a web page. You can use the Properties view to edit properties that are related to the appearance of tags in a web browser. For example, you can change CSS style information, default attribute values, Dojo properties, and jQuery properties, as required.

You can use the Properties view to edit JavaScript, HTML, or JSP tags when the Design, Source, or Split view is open in Rich Page Editor. Changes in the Properties view are displayed in Rich Page Editor when you change the cursor focus or press Enter. If you update tags in the Source view of Rich Page Editor, your changes take effect immediately in the Properties view.

### Breadcrumb navigation

When you select a node in Rich Page Editor, the Properties view uses a breadcrumb trail to provide context for the selected node:



You can scroll through the breadcrumb trail without losing the position of your cursor in Rich Page Editor. Using this feature, you can quickly update the properties of ancestor elements.

### Categorized property pages

The Properties view organizes properties into various categories, including:

**Styles** Use to manipulate basic CSS style information (such as an attribute or the class that is associated with it) or various font, color, and alignment properties.

**Layout**

Use to configure properties that control the layout of the element within the presentation of the page.

**All** Use to view all of the attributes for an element, in a tabbed list.

**Dojo** Use to configure Dojo-specific properties on certain widgets.

**Note:** This category applies only to Dojo-enabled web projects.

**jQuery**

Use to configure jQuery-specific properties on certain widgets.

**Note:** This category applies only to jQuery-enabled projects.

**Mobile Navigation view:**

You can use the Mobile Navigation view to manage Dojo mobile view widgets and jQuery mobile web page widgets.

For example, by using the Mobile Navigation view, you can:

- Add or remove mobile views and pages.
- Switch visibility from one mobile view or page to another.
- Rename mobile views and pages.
- Set the default mobile view or page that is shown the first time that a web page opens.
- Link mobile views or pages.

The Mobile Navigation view is available from both the Web perspective and Rich Page Editor:

- To open the view from the Web perspective, select **Window > Show view > Other > Web > Mobile Navigation**.
- To open the view from Rich Page Editor, on the toolbar click **Show/Hide Mobile Navigation**:



A mobile web page can contain multiple views or pages. You can create these views and pages inline or in external files.

**Inline mobile view or page**

A mobile view or page that is written within the source code of the mobile web page.

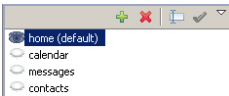
**External mobile view or page**

A mobile view or page that is written in a separate file or fragment. Creating mobile views or pages in separate files or fragments makes source code shorter and easier to manage.

When you open a mobile web page in Rich Page Editor, the mobile views or pages that are contained within that web page are displayed in the Mobile Navigation view. The icon to the left of each of the mobile views and pages indicates which one is visible in Rich Page Editor. If the mobile web page references external


mobile views or pages, they are displayed in the Mobile Navigation view with a decorated icon. To open a new instance of Rich Page Editor for an external mobile view or page, double-click the mobile view or page.

The following table lists and describes the features available for mobile web pages in the Mobile Navigation view.

What you can do in the Mobile Navigation view	Description
Create mobile views or pages	<p>You can create the following types of Dojo widgets:</p> <p><b>View</b> A container that represents the entire mobile device screen.</p> <p><b>ScrollableView</b> A view widget with touch scrolling capability that you can use to provide fixed position header and footer bars.</p> <p><b>SwapView</b> A view widget that you can swipe horizontally to show adjacent SwapView widgets.</p> <p>You can create the following types of jQuery widgets:</p> <p><b>Page</b> A container that represents the entire mobile device screen.</p> <p><b>Dialog page</b> A container that is shown in the form of a dialog box.</p>
Switch between mobile views or pages	<p>You can switch visibility between views or pages to specify which view or page is available in Rich Page Editor. In the following screen capture, the <b>home</b> view is visible in Rich Page Editor; the calendar, messages, and contacts views are not visible.</p>  <p>To switch to the calendar view, click the icon to the left of <b>calendar</b>.</p>
Rename mobile views or pages	<p>Right-click the view or page that you want to rename and then click <b>Rename</b>. For example, to rename the calendar view to internet:</p> <ol style="list-style-type: none"> <li>1. Right-click <b>calendar</b> and then click <b>Rename</b>.</li> <li>2. In the <b>Mobile View id</b> field, specify internet.</li> </ol>
Set the default mobile view or page	<p>Right-click the view or page that you want to set as the default and click <b>Set as default</b>.</p>
Remove mobile views or pages	<p>Right-click the view or page that you want to remove and click <b>Remove</b>.</p>



The following table lists and describes the features available for mobile web pages in the Mobile Navigation view.

What you can do in the Mobile Navigation view	Description
Link mobile views or pages	<p>You can link widgets, such as buttons or list view items, to mobile views or pages. You can drag a widget from the Design view in Rich Page Editor to a mobile view or page in the Mobile Navigation view. You can also drag mobile views or pages from the Mobile Navigation view to widgets in the Design view within Rich Page Editor.</p> <p><b>Tip:</b> You can link Dojo mobile widgets to mobile views by using the <b>Link to Mobile View</b> action.</p> <ol style="list-style-type: none"> <li>1. In the Design view within Rich Page Editor, click the Dojo mobile widget that you want to link to a mobile view to open the toolbar.</li> <li>2. To open the Link to Mobile View dialog, on the toolbar click <b>Link to Mobile View</b>:  </li> <li>3. Select one of the following options. <ul style="list-style-type: none"> <li>• Click <b>Inline Mobile View</b>; from the list, select the mobile view that you want to link to the widget.</li> <li>• Click <b>Page Fragment</b>, and then click <b>Browse</b> to browse to the mobile page file that you want to link to the mobile view.</li> </ul> </li> <li>4. Click <b>Finish</b>.</li> </ol>

## Testing mobile web applications

You can use the mobile browser simulator to emulate various mobile devices and test your mobile web applications without the need to install device vendor native SDK.

### Before you begin

1. Create a mobile web project.
2. Create web pages for mobile devices.
3. Add Dojo mobile widgets to your mobile web pages.

### About this task

**Important:** The Mobile Browser Simulator supports the following web browsers:

- Firefox version 3.6 and later.
- Chrome 17 and later.
- Safari 5 and later.

## Procedure

1. In the Enterprise Explorer view, right-click your mobile web page and select **Run As > Run on Mobile Browser Simulator**. The Run on Server dialog opens.
2. In the Run on Server wizard, verify that the **Choose an existing server** radio button is enabled.
3. Select a server from the list of servers and then click **Finish**.

## What to do next

After your web page is running in the mobile browser simulator, you can view how your page renders in different devices.

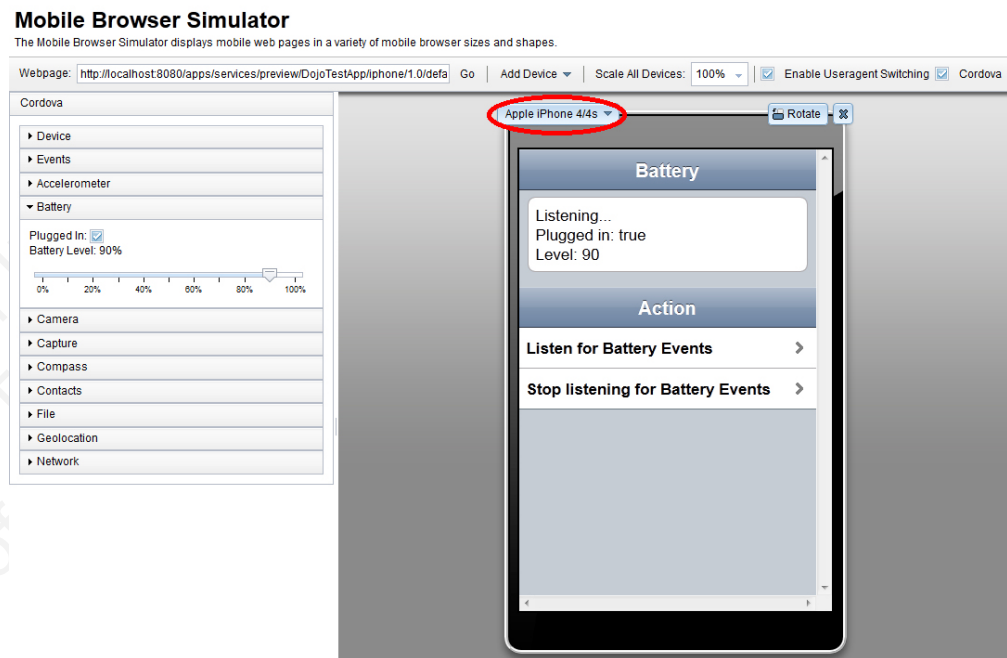
### Switching devices: Before you begin

To view your web application in the simulated devices using the appropriate style sheets, ensure that you completed the following tasks:

1. "Creating web pages for mobile devices" on page 299
2. Enable user agent switching.

## Procedure

In the simulator, click the device list and then select the device that you want to simulate.



### Adding devices: Before you begin

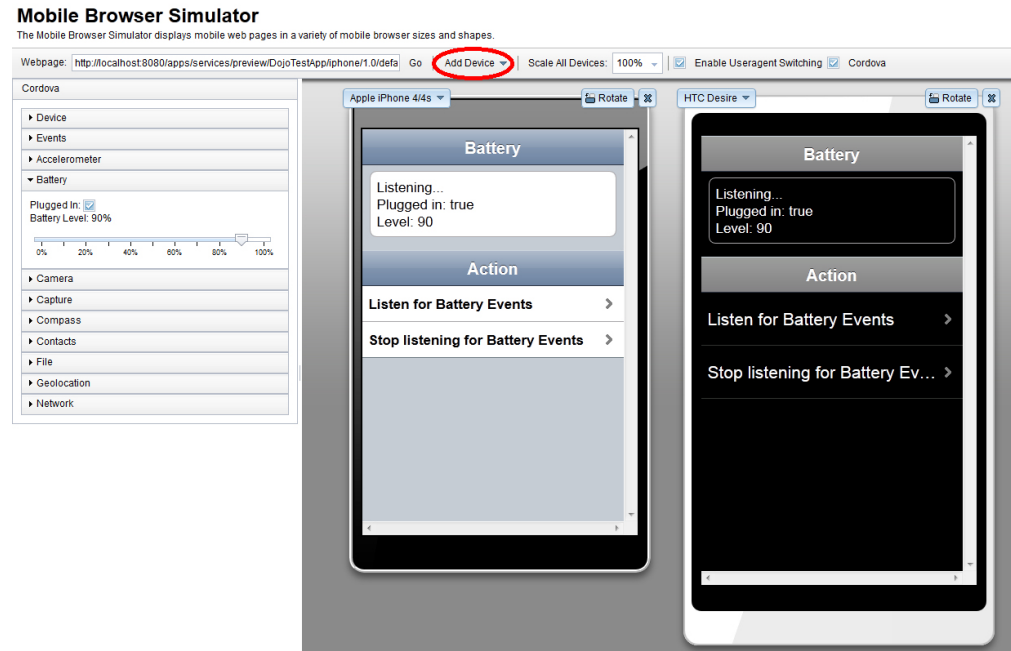
To view your web application in the simulated devices using the appropriate style sheets, ensure that you completed the following tasks:

1. "Creating web pages for mobile devices" on page 299

2. Enable user agent switching.

## Procedure

In the simulator, click **Add Device** and then select the device that you want to simulate.



**Tip:** You can customize the list of device options that are available in the mobile browser simulator.

1. In Worklight Studio, select **Window > Preferences > Web > Target Devices**.
2. Add your custom device to the current list of target devices, and then launch the simulator again.

The custom device that you added is now available as an option from the **Add Device** list in the simulator.

### Mobile browser simulator:

The mobile browser simulator is a web application that helps you test mobile web applications without having to install device vendor native SDK.

**Important:** The Mobile Browser Simulator supports the following web browsers:

- Firefox version 3.6 and later.
- Chrome 17 and later.
- Safari 5 and later.

You can use the Mobile Browser Simulator to preview Worklight applications on Android, iPhone, iPad, BlackBerry 6 and 7, Windows Phone 7, Windows Phone 8, and mobile web application environments.

**Tip:** When you preview a Worklight application on an Android, iPhone, iPad, BlackBerry 6 and 7, Windows Phone 7, or Windows Phone 8 environment, only the devices for the selected environment are available. For example, if you preview a Worklight application on an Android environment, you can select only from the

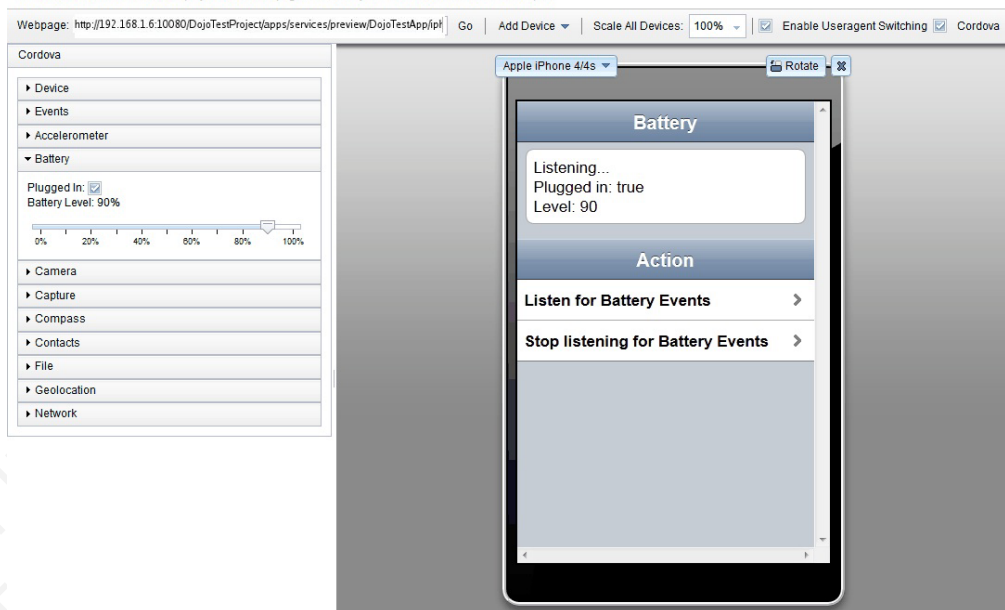
list of available Android devices. When you preview a Worklight application on a mobile web application environment, you can select from the list of available Android, iPhone, iPad, BlackBerry 6 and 7, Windows Phone 7, and Windows Phone 8 devices.

You can also use the Ripple emulator to simulate the WebWorks API in your BlackBerry application. Using Chrome as your web browser, click **Open Simple Preview** in the simulator. A new tab opens in Chrome with your application loaded; you can open the Ripple emulator from this tab.

The mobile browser simulator contains a frame that emulates a target device. It shows you what your page looks like inside the mobile device browser. You can switch the frame to emulate different screen resolutions and form factors, including BlackBerry 6 and 7, Android, iPad, iPhone, Windows Phone 7, and Windows Phone 8 mobile devices. You can also rotate the frame to mimic orientation change (portrait or landscape). You can add multiple devices to the frame to view the various displays simultaneously. If a device detection servlet is configured for your web project, the simulator emulates requests from different device-specific agents.

### Mobile Browser Simulator

The Mobile Browser Simulator displays mobile web pages in a variety of mobile browser sizes and shapes.



### Calibrating the mobile browser simulator:

Since browsers cannot accurately paint physical dimensions, you must calibrate the mobile browser simulator.

#### Before you begin

Test your application using the mobile browser simulator.

#### Procedure

1. From the **Scale All Devices** list, select **Physical device size**.
2. Click **Calibrate Physical Size** to open the Physical Size Calibration dialog.
3. Follow the instructions in the dialog to calibrate your mobile browser simulator. After you complete all of the steps in the dialog, close the dialog.

## Enabling user agent switching:

You can use the mobile browser simulator to render your web applications on different mobile devices. To render your web applications with the appropriate style sheets and theme, you must enable user agent switching.

### Before you begin

- Enable the detect device option when you create your web page.
- Test your application by using the mobile browser simulator.

### About this task

The Useragent Switcher Extension is a browser extension that provides the user agent switching feature. The mobile browser simulator supports implementations of this browser extension for the following web browsers:

- Mozilla Firefox.
- Chrome 17 and later, with limitations.

### Useragent Switcher Extension for Chrome

The Useragent Switcher Extension emulates requests from different device-specific agents. When a web application checks the user agent on the server to create content, it is correctly simulated.

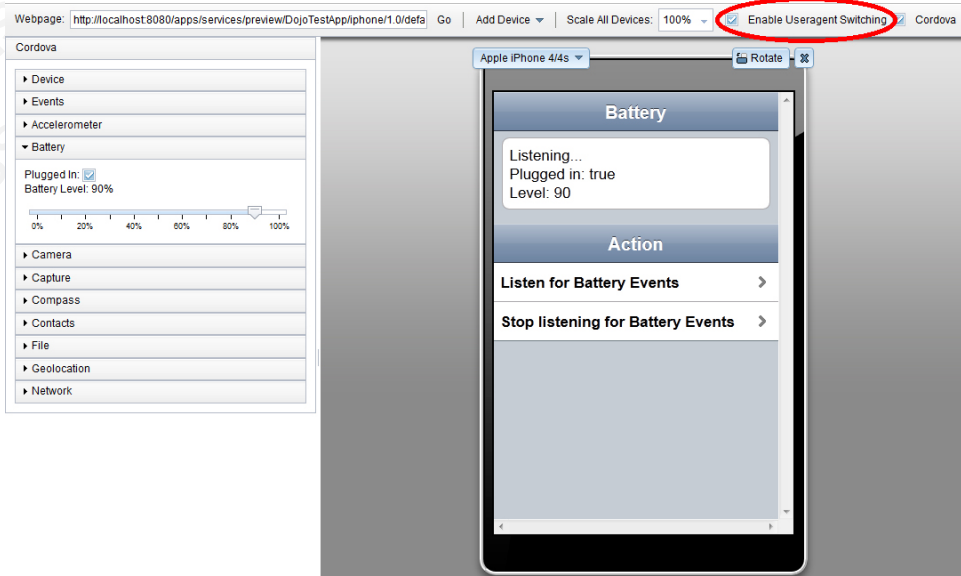
The Useragent Switcher Extension includes support for Dojo Mobile 1.7 and later. If you enabled the detect device option when you created your Dojo Mobile page, the Useragent Switcher Extension uses the automatic device detection and theme loading for Dojo Mobile to select the appropriate theme.

### Procedure

1. Click **Enable Useragent Switching**.
2. If the latest version of the Useragent Switcher Extension is not installed, the Install Useragent Switcher Extension dialog opens. Click **Install Browser Extension**.

#### Mobile Browser Simulator

The Mobile Browser Simulator displays mobile web pages in a variety of mobile browser sizes and shapes.



## Results

You can now view your web application with the appropriate style sheets and theme in the simulated mobile devices.

## Previewing your Worklight applications

You can use the Mobile Browser Simulator to preview Worklight applications on iPhone, iPad, Android phones and tablets, BlackBerry 6 and 7, BlackBerry 10, Windows Phone 7.5, Windows Phone 8, Windows 8 desktop and tablets, and Mobile web app environments. You can simulate several mobile devices simultaneously.

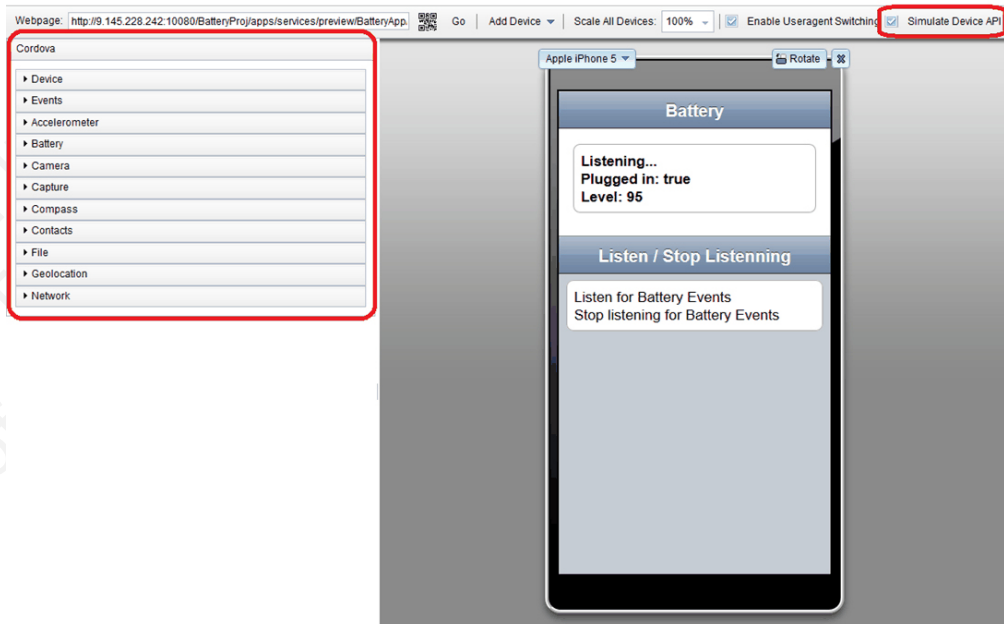
**Tip:** This preview is only available when the `com.ibm.imp.worklight.simulation.ui` plug-in is enabled.

The Apache Cordova API simulation user interface is packaged with the Mobile Browser Simulator. When the Mobile Browser Simulator opens, the various data types and values that are used by Cordova are displayed in the left side. The Cordova simulation is available on the following environments:

- Android
- BlackBerry 10
- iPhone
- iPad
- Windows Phone 7.5
- Windows Phone 8
- Windows 8 desktop and tablets environments

### Mobile Browser Simulator

The Mobile Browser Simulator displays mobile web pages in a variety of mobile browser sizes and shapes.



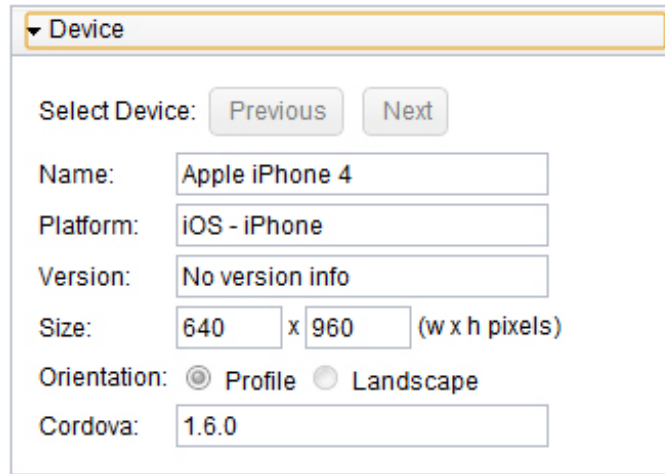
**Tip:** If you do not want to use the Cordova simulation to preview your Worklight applications, clear the **Cordova** check box to disable the Cordova simulation.

### Device

Shows the property values for the `window.device` object of each simulated



device. This data is read-only. To show the values for other devices, click **Previous** or **Next**.



▼ Device

Select Device: Previous Next

Name: Apple iPhone 4

Platform: iOS - iPhone

Version: No version info

Size: 640 x 960 (w x h pixels)

Orientation:  Profile  Landscape

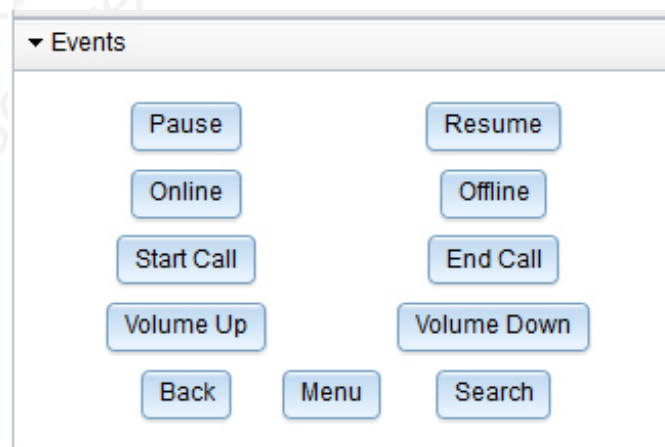
Cordova: 1.6.0

### Events

Triggers any of the following Cordova events:

- pause
- resume
- online
- offline
- backbutton
- menubutton
- searchbutton
- startcallbutton
- endcallbutton
- volumedownbutton
- volumeupbutton

To trigger a Cordova event, click the corresponding button:



▼ Events

Pause	Resume	
Online	Offline	
Start Call	End Call	
Volume Up	Volume Down	
Back	Menu	Search

### Accelerometer

Defines the Accelerometer values returned by the Cordova API when querying Accelerometer data. To generate a new set of values, click **Next**. To generate the values periodically, click **Start**.

▼ Accelerometer

X:

Y:

Z:

### Battery

Defines battery-related data, such as the battery level. You can use the slider to change the battery level and trigger a batterystatus event. The following battery levels trigger events:

- Twenty percent triggers the batterylow event
- Five percent triggers the batterycritical event

To define the plugged in status of your mobile device, select or clear the **Plugged In** check box.

▼ Battery

Plugged In:

Battery Level: 90%

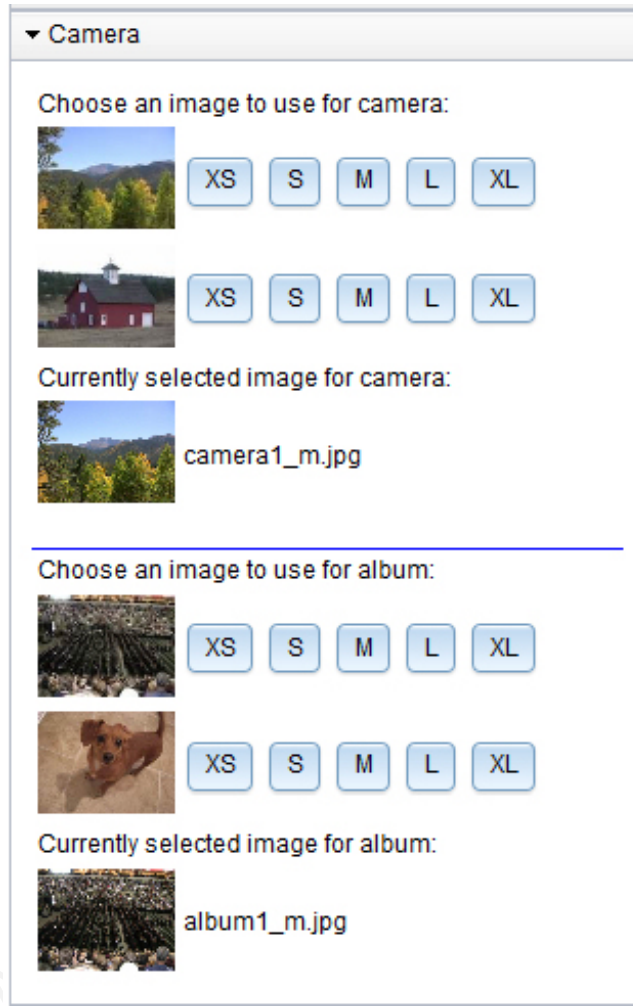
0% 20% 40% 60% 80% 100%

### Camera

Specifies which image to use for the camera and for the album:

- Simulate a photo taken with the camera (Camera.sourceType == Camera.PictureSourceType.CAMERA)
- Photo from the device photo album or library (Camera.sourceType == Camera.PictureSourceType.PHOTOLIBRARY or Camera.sourceType == Camera.PictureSourceType.SAVEDPHOTOALBUM)

To change the size of the selected photos, click **XS**, **S**, **M**, **L**, or **XL**.

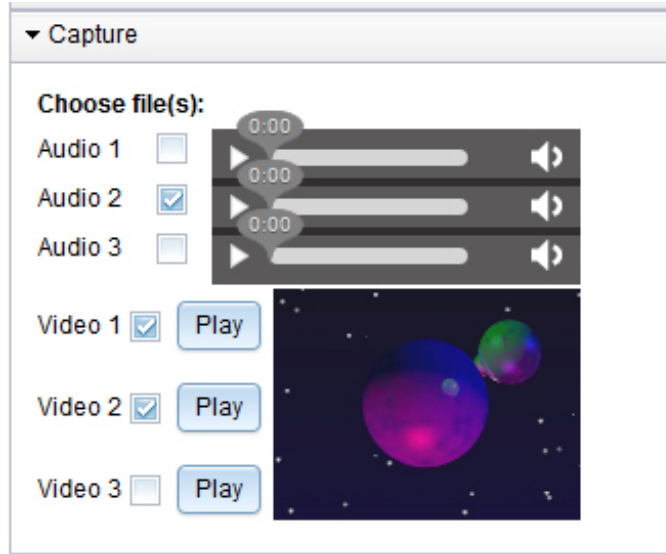


### Capture

Simulates the Cordova capture API by using the following methods:

- capture.captureAudio
- capture.captureVideo

You can select the audio and video recordings that you want to use, and play these recordings by using the HTML5 players.



**Note:** The Capture section is available on both Mozilla Firefox and Google Chrome. For improved support of the HTML 5 players, upgrade these browsers to the latest version.

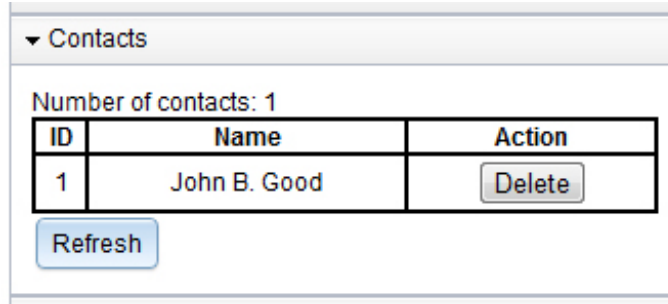
### Compass

Defines the values returned by the Cordova API when querying Compass data. To generate a new set of values, click **Next**. To generate the values periodically, click **Start**. You can also set the compass values by directly interacting with the compass widget.



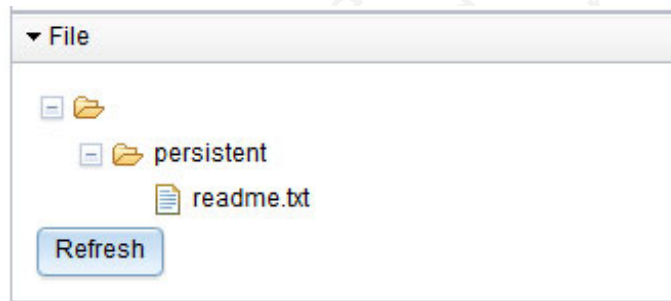
### Contacts

Shows the available contacts for the mobile device. You can delete contacts and refresh the list of available contacts.



To create new contacts for the mobile device, use the Cordova Contacts API from your simulated mobile web page. The contacts are stored in the Web SQL Database which is supported by default by Google Chrome and Safari. To simulate the Contacts API with Firefox, you must install an Add-on in your browser that adds basic WebSQL support to Firefox.

**File** Simulates the Cordova File API by running an applet. To update the display of the file system that you can access through the Cordova API, click **Refresh**. Use the Cordova API to access this file system to read and write.

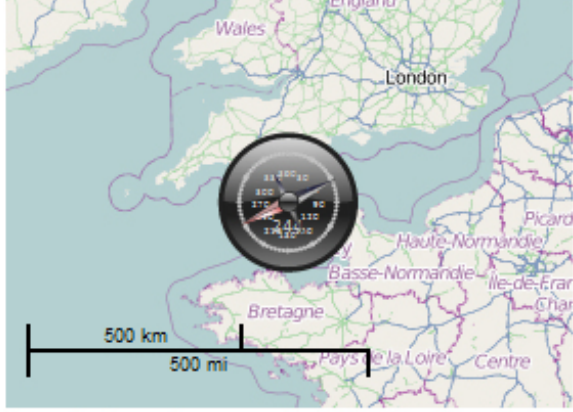


**Geolocation**

Generates the Geolocation values returned by the Cordova API when querying Geolocation data. To generate a new set of values, click **Next**. To generate the values periodically, click **Start**.

▼ Geolocation

Latitude	49.77
Longitude	-2.82
Accuracy	71.7
Altitude	3672.10
Altitude Accuracy	1
Heading	244.10077603068203
Velocity	57.2



Next Start

### Network

Defines the active connection of the device.

▼ Network

- None
- Ethernet Network
- Wifi Network
- 2G Network
- 3G Network
- 4G Network
- Unknown Network

The Cordova API also contains media simulation. Media simulation is available only for audio playback; audio recording is not supported. The media simulation uses an HTML audio player and audio playback is supported on Mozilla Firefox and Google Chrome. Since some browsers might not support all audio file formats, use OGG audio files.

The Cordova Notification API is simulated but does not require any user interface in the Mobile Browser Simulator.

## Using IBM Worklight Client API

Using IBM Worklight client API

IBM Worklight defines a series of client API that you can use in your apps. With these API, you can perform various actions, including the following ones:

- 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, and Windows Phone 8 devices
- Work offline
- Detect jailbreak
- Use encrypted cache

For more information about these client API, see “IBM Worklight client-side API” on page 489.

## 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 503

This method establishes a connection to the Worklight Server.

## Web and native code in iPhone, iPad, and Android

Using IBM Worklight, 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.

#### 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](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](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.

### 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 view]
}
-(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"
```

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.

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

## Developing hybrid applications for iOS

Develop hybrid applications for iOS as detailed here.

## 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.

## Developing hybrid applications for Android

Develop hybrid applications for Android as detailed here.

### 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.

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.

### 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:

```
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 26. 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 27. Custom code in a migrated application

## 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.

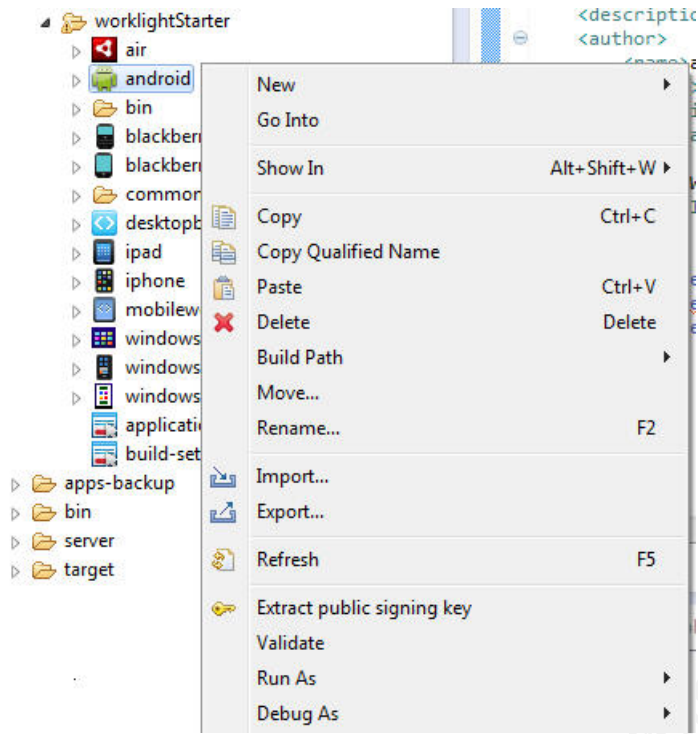


Figure 28. Extracting the public signing key

A wizard window opens.



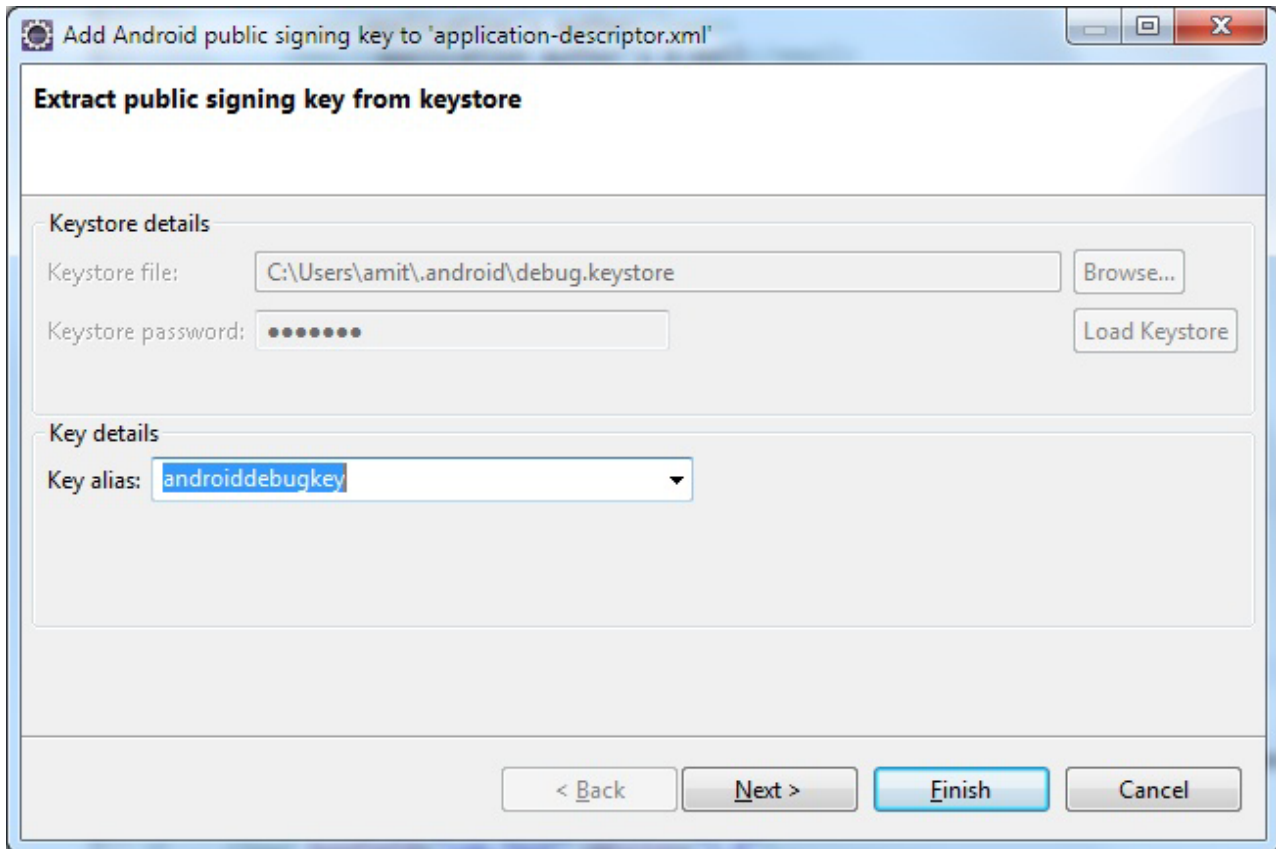


Figure 29. Adding the Android public signing key

- 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 XP	C:\Documents and Settings\user_name\ .android\
OS X and Linux	~/.android/

- Enter the password to your keystore and click **Load Keystore**.
- 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>.
- In the window, click **Finish** to copy the public signing key directly into the application descriptor.

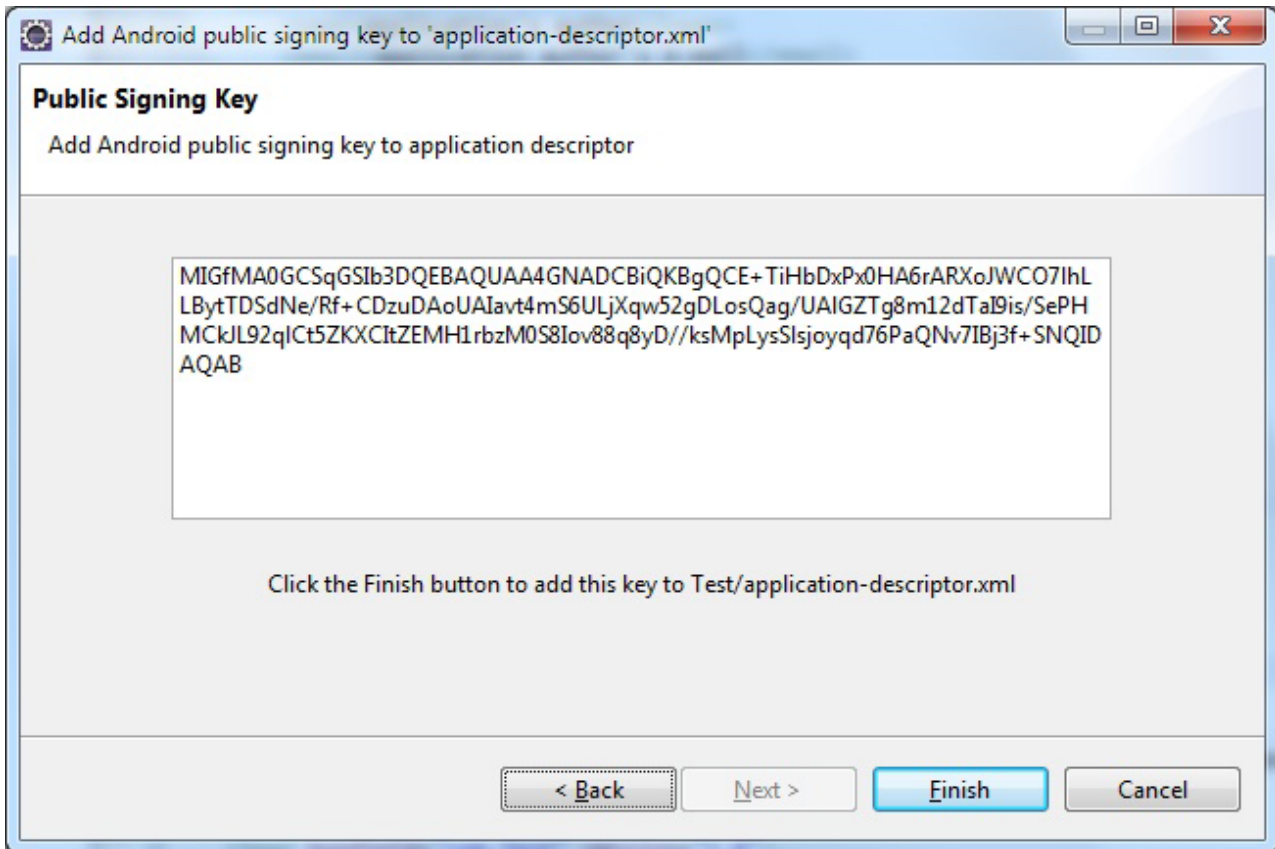


Figure 30. Android public signing key

## 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+TiHbDxPx0HA6rARXoJWC07lhL
    LBytTDSdNe/Rf+CDzuDAoUAIavt4mS6ULjXqw52gDLosQag/UAIGZTg8m12dTal9is/SePH
    MCKlJL92qIct5ZKXCItZEMH1rbzM0S8Iov88q8yD//ksMpLysSljoyqd76PaQNv7IBj3f+SNQID
    AQAB
  </security>
</android>
```

Figure 31. Android public signing key as shown in the application descriptor

## Managing device orientation

When you develop Android applications that target an API level equal or higher than 13, you must include the `screenSize` value to the `android:configChanges` attribute in the `AndroidManifest.xml` file. Otherwise, the application fails to run properly when the device orientation changes.

Assuming that IBM Worklight is the first main activity in the `AndroidManifest.xml` file of your application:

- If your target API is equal or higher than 13, you must add the `screenSize` value to the `android:configChanges` attribute of the `<activity>` element, as shown in the following example:

```
<activity android:name=".worklightStarter" android:label="@string/app_name" android:configChange
```

- If your target API is smaller than 13, your activity always handles this configuration change itself, and you do not need to add the screenSize value to the <activity> element.

## Developing hybrid applications for BlackBerry

Develop hybrid applications for BlackBerry as detailed here.

### 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 2.6. 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.

**Note:** BlackBerry OS 10 is not supported by the current version of the Application Center.

#### 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
#
qnx.sim.password=
```

```
for device:
```

```
The initial device ip is 169.254.0.1, that is, the one that is usually given when connected v
# 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 debugging 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.

## 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

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.
- 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 “IBM Worklight security framework” on page 417.

### 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.

### 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.

### 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.

### 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 10080, make sure that you also set this port as the value of the configuration `publicWorkLightPort`. Otherwise, the action **Embed in Web Page** does not provide you with the correct URL. For descriptions of `publicWorkLightPort` and other IBM Worklight configuration properties, see “Configuring an IBM Worklight project in production by using JNDI environment entries” on page 723. For information about how to specify IBM Worklight configuration properties, see “Configuration of IBM Worklight applications on the server” on page 714.



## 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:

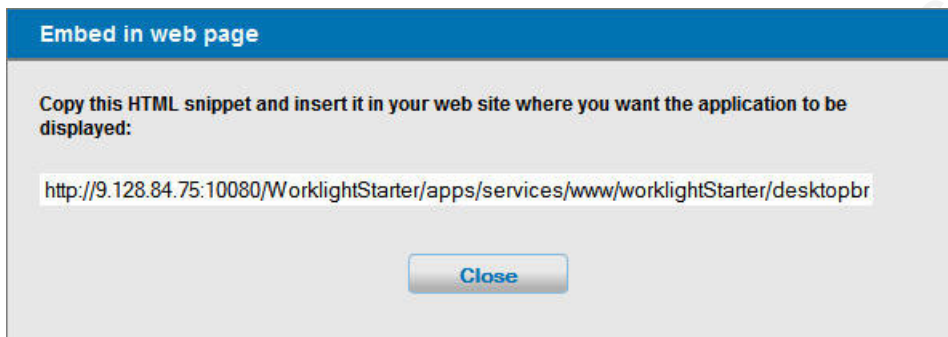


Figure 32. Embedding a widget in a predefined web page

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;">
```

---

## Developing native applications

This collection of topics gives instructions for developing native applications.

### 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.

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 335.
    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 335.
  - For Android:
    1. To update the application descriptor file, see “Application Descriptor of Native API application for Android” on page 336.
    2. To update the client property file, see “Client property file for Android” on page 338.
    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 339.
  - 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 340.
    2. To update the client property file, see “Client property file for Java Platform, Micro Edition (Java ME)” on page 341.
    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 342.

You build and deploy Native API applications by following the same procedure as for hybrid applications, by creating the `.wla` file and uploading it to the Worklight Console. For more information about deployment, see “Deploying applications and adapters to Worklight Server” on page 733.

## Developing native applications for iOS

This collection of topics gives instructions for developing native applications for iOS.

### 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="6.0.0"
```

```

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="6.0.0"
  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 IBM Worklight 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>
```

**<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 `WLCient`.

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 43. Properties of the `worklight.plist` file

Property	Description	Example values
<b>protocol</b>	The communication protocol with the Worklight Server.	http or https
<b>host</b>	The host name of the Worklight Server.	localhost
<b>port</b>	The port of the Worklight Server. If this value is left blank, the default port is used. If the <b>protocol</b> property value is https, you must leave this value blank.	10080
<b>wlServerContext</b>	The server URL context.	/ <b>Note:</b> If you use IBM Worklight Developer Edition, you must set the value of this property to the name of your Worklight project.
<b>application id</b>	The application ID, as defined in the <code>application-descriptor.xml</code> file.	myApp
<b>application version</b>	The application version, as defined in the <code>application-descriptor.xml</code> file.	1.0
<b>environment</b>	This property defines the IBM Worklight environment. The value of this property must be <code>iOSnative</code> . You must not modify the value of this property value.	iOSnative

## 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

## 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, the `worklight.plist` file, and the `Localizable.strings` 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, the `worklight.plist` file, and the `Localizable.strings` 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`

## Developing native applications for Android

This collection of topics gives instructions for developing native applications for Android

### 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:

```
<?xml version="1.0" encoding="UTF-8"?>
<nativeAndroidApp
  id="android"
  platformVersion="6.0.0"
  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="6.0.0"
  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 IBM Worklight 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"/>
```

#### <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 323.

```
</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 44. Properties and values of the `wlclient.properties` file

Property	Description	Example values
<b>wlServerProtocol</b>	The communication protocol with the Worklight Server.	http or https
<b>wlServerHost</b>	The host name of the Worklight Server.	localhost
<b>wlServerPort</b>	The port of the Worklight Server. If you leave this value blank, the default port is used. If the <b>wlServerProtocol</b> property value is https, you must leave this value blank.	10080
<b>wlServerContext</b>	The server context.	/ <p><b>Note:</b> If you use IBM Worklight Developer Edition, you must set the value of this property to the name of your Worklight project.</p>
<b>wlAppId</b>	The application id, as defined in the <code>application-descriptor.xml</code> file.	myApp
<b>wlAppVersion</b>	The application version, as defined in the <code>application-descriptor.xml</code> file.	1.0
<b>wlEnvironment</b>	This property defines the IBM Worklight environment. The value of this property must be <code>Androidnative</code> . You must not modify the value of this property value.	Androidnative



Table 44. Properties and values of the `wlclient.properties` file (continued)

Property	Description	Example values
<b>GCMsenderID</b>	This property defines the GCM Sender ID that you must use for push notifications. By default, this property is commented.	

## 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 and the `uicandroid.jar` file from the Native API application, and paste them 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.
  - c. `<uses-permission android:name="android.permission.GET_TASKS"/>` This line adds the permission to get a list of running tasks that are required for the heartbeat functionality.
  - d. `<uses-permission android:name="android.permission.ACCESS_WIKI_STATE"/>`
5. If push notification support is required, add the following permissions to the `AndroidManifest.xml` file of your native app for Android:
  - a. `<uses-permission android:name="com.worklight.androidnativepush.permission.C2D_MESSAGE"/>`
  - b. `<uses-permission android:name="com.google.android.c2dm.permission.RECEIVE"/>`
  - c. `<uses-permission android:name="android.permission.WAKE_LOCK"/>`

- d. `<uses-permission android:name="android.permission.GET_ACCOUNTS"/>`
  - e. `<uses-permission android:name="android.permission.USE_CREDENTIALS"/>`
  - f. `<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>`
6. Manage your splash screens: 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:
- a. If you want to use a splash screen in your app, ensure that the required `splash.png` file or `splash.9.png` file is present in each of these folders.
  - b. If you do not want to use a splash screen in your app, ensure that no `splash.png` file or `splash.9.png` file is present in these folders.

## Developing native applications for Java Platform, Micro Edition

This collection of topics gives instructions for developing native applications for Java Platform, Micro Edition

### 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="6.0.0"
  version="1.0"
  securityTest="security test name"
  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="6.0.0"
  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 IBM Worklight 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.

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 45. 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 Worklight Server.	10080

Table 45. Properties and values of the `wlclient.properties` file (continued)

Property	Description	Example values
<code>wlServerContext</code>	The server context.	/ <b>Note:</b> If you use IBM Worklight Developer Edition, you must set the value of this property to the name of your Worklight project.
<code>wlAppId</code>	The application ID, as defined in the <code>application-descriptor.xml</code> file.	myApp
<code>wlAppVersion</code>	The application version, as defined in the <code>application-descriptor.xml</code> file.	1.0
<code>wlEnvironment</code>	This property defines the IBM Worklight environment. The value of this property must be <code>JavaMEnative</code> . 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.

## Optimizing IBM Worklight applications

Worklight Studio has several features that you can use to reduce the size of your application or otherwise improve its performance or reduce its load time.

During development, the applications you develop can perform well. But when these apps are used by mobile devices, performance can be impacted by a number of factors.

The large size of applications can make initial download times from the Application Center too long for users. Inclusion of multiple JavaScript files in Desktop Browser and Mobile Web applications can require multiple requests to retrieve them when the app is started, increasing start time. Unused resources such as large images or unneeded files included in the generated Cache Manifest file can further slow start time for these types of applications.

Worklight Studio V6.0.0 includes a number of features that can reduce the size of your Worklight web applications, such as minification or removing unused features such as JSONStore. It also includes features that can improve performance and user satisfaction by enabling them to start faster, such as concatenation and editing the Cache Manifest. These features are described in the following topics.

## Including and excluding application features

If features such as JSONStore are not used in your application or in certain environments, you can reduce the application size by excluding them.

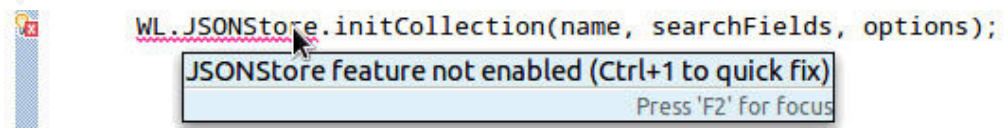
In IBM Worklight V6.0.0, you can include or exclude features from the application build if those features are not required. For example, JSONStore offers many benefits, if code that references it is actually used in the application. If it is not used, the JSONStore resources greatly increase the application size, and thus slow both initial app download time and app start time.

There is a new <features> element in the Application Descriptor that controls the inclusion or exclusion of resources. In the application-descriptor.xml file itself, this element appears similar to the following example, which shows JSONStore resources being included in the build:

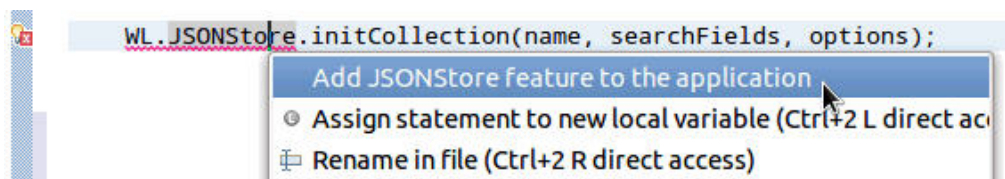
```
<application xmlns="http://www.worklight.com/application-descriptor" id="myApp" platformVersion="6.0.0">
  ...
  <features>
    <JSONStore/>
  </features>
  ...
</application>
```

For more information about Application Descriptor attributes, see “The application descriptor” on page 255.

When you first create an IBM Worklight application, the <features> tag is automatically created in the application-descriptor.xml file, with no contents. What this means is that if you use JSONStore in your code, it is not automatically added to the builds. When you run the application, you receive an error, as shown in the following screen capture:



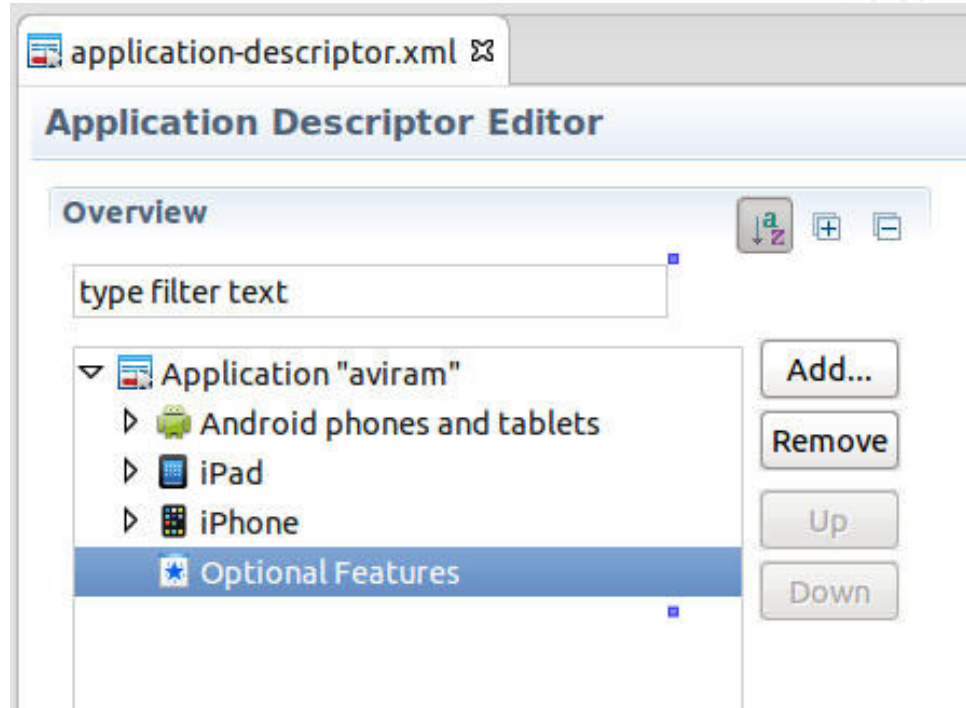
You can resolve this situation by using an Eclipse QuickFix:



But you can also choose which features to include in the build with the Worklight Studio editor, as shown in the following procedure.

### To include or exclude features in Worklight Studio

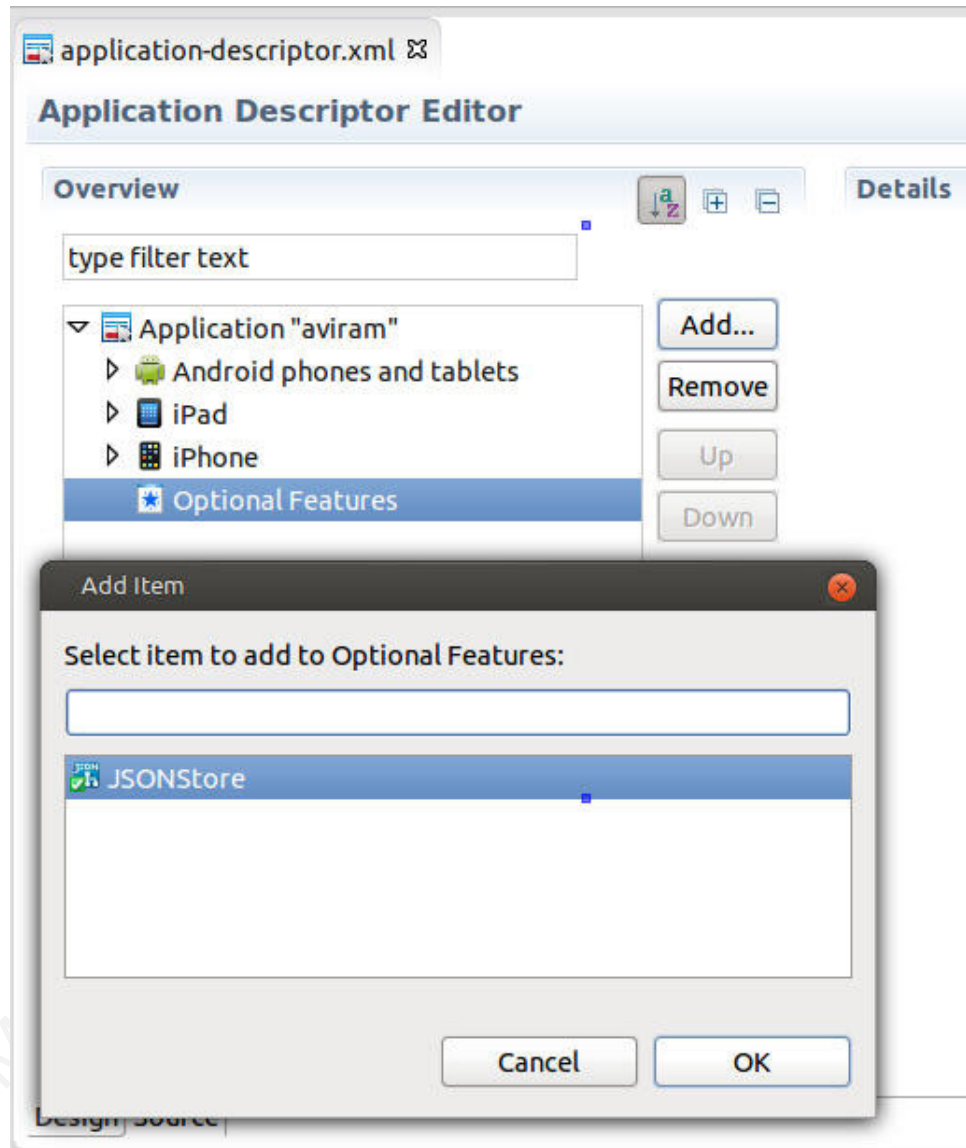
1. In Worklight Studio, open the `application-descriptor.xml` file for your application with the Application Descriptor Editor:



If the **Optional Features** element is empty (as in the screen capture), no features such as JSONStore are included in the build.

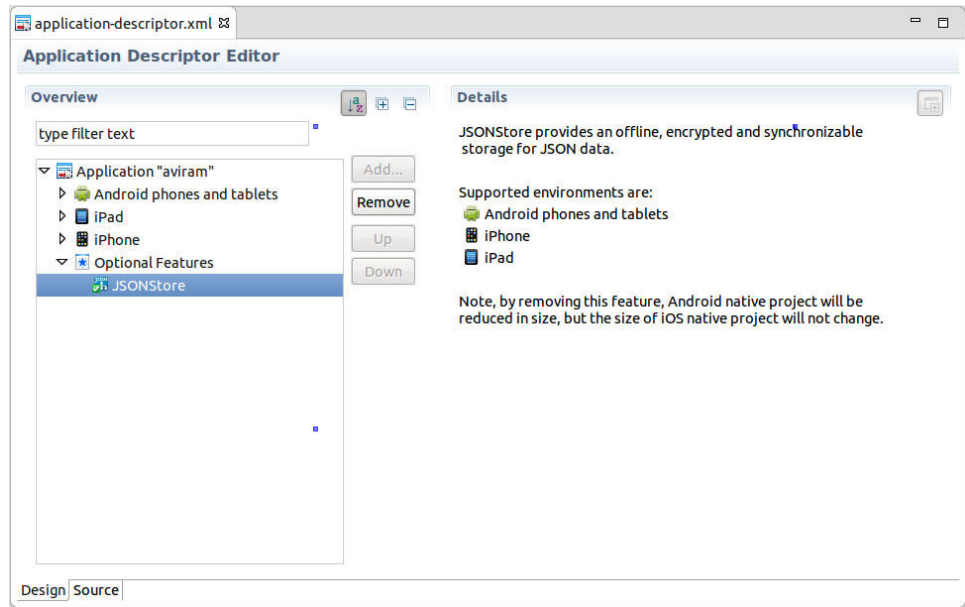
2. To add features, click **Add** to display the Add Item window:





3. Choose the feature that you want to add to the build (in this example, **JSONStore**), and click **OK**.
4. The Application Description Editor now displays **JSONStore** as an attribute of **Optional Features**, along with **Details** about the feature in the right panel:





- To remove a feature, select it in the left panel and click **Remove**.

## Application cache management in Desktop Browser and Mobile Web apps

Worklight Studio provides mechanisms by which you can control the contents of the application cache for Desktop Browser and Mobile Web environments.

### The application cache

Ideally, you want mobile and desktop web applications to be able to work when the user is offline. Older browsers had their own caching mechanisms, but they were not always reliable. The release of HTML5 addressed this need with the introduction of the *application cache*, which provides users three advantages:

- Offline browsing – users can work with the application when they are offline.
- Speed – cached resources are local, and thus load faster.
- Reduced server load – the browser only downloads resources that are updated or changed from the server.

For more information, see HTML5 Application Cache.

### The application cache manifest

The *Cache Manifest* is a simple text file that lists the resources that the browser is to cache for offline access. It contains a list of resources that are explicitly cached after the first time they are downloaded. The Cache Manifest can contain three sections:

- **CACHE** – Files and resources that are listed under this heading (or immediately after the **CACHE MANIFEST** heading if no sections are present) will be explicitly cached after the first time they are downloaded.
- **NETWORK** – Files listed under this heading are white-listed resources that require a connection to the server. All requests to these resources bypass the cache, even if the user is offline.

- **FALLBACK** – An optional section that specifies fallback pages if a resource is inaccessible. The first URI listed is the primary resource, and the second URI is the fallback. Both URIs must be relative and from the same origin as the manifest file.

When the browser opens a document that includes the manifest attribute, the browser loads the document and then fetches all the entries that are listed in the Cache Manifest file. If no application cache exists, the browser creates the first version of the application cache.

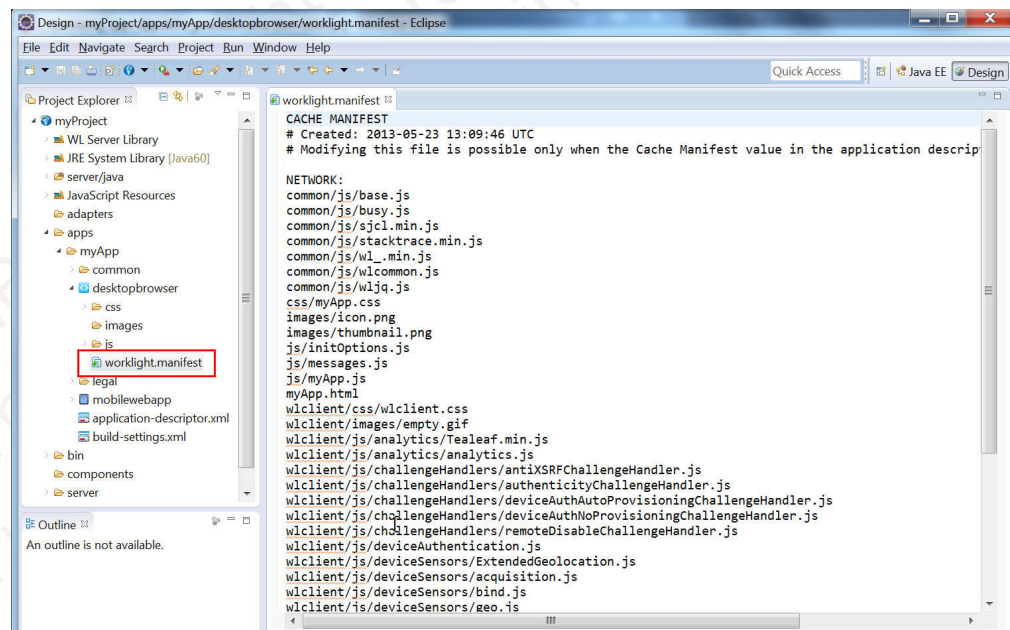
If unnecessary or redundant files are included, they must all be fetched before the application can start, which can create a poor user experience. The procedure that follows documents ways in which you can edit the Cache Manifest to reduce the start time for your Desktop Browser and Mobile Web applications.

## Managing the application Cache Manifest in Worklight Studio

The procedures for managing and editing the contents of the application cache for Desktop Browser and Mobile Web applications are listed in this section.

In Worklight V6.0.0, you can control the Cache Manifest in web environments (Desktop Browser and Mobile Web). The name of the Cache Manifest file is `worklight.manifest`. This file is located in the folder for each of these types of environments.

You can now view (and edit) the contents of this file in Worklight Studio, as shown in the following screen capture:



A new attribute now exists in the Application Descriptor (`application-descriptor.xml`) for Desktop Browser and Mobile Web elements. The current setting of this attribute, called `<cacheManifest>`, can be easily viewed, as shown in the following screen capture:

```

<!-- Attribute id must be identical to application folder name -->
<application xmlns="http://www.worklight.com/application-descriptor" id="lala" platformVersion=
<displayName>222</displayName>
<description>lala22</description>
<author>
  <name>application's author</name>
  <email>application author's e-mail</email>
  <homepage>http://mycompany.com</homepage>
  <copyright>Copyright My Company</copyright>
</author>
<mainFile>lala.html</mainFile>
<thumbnailImage>common/images/thumbnail.png</thumbnailImage>
<iphone bundleId="com.lala" version="1.0">
  <worklightSettings include="true"/>
  <security>
    <encryptWebResources enabled="false"/>
    <testWebResourcesChecksum enabled="false" ignoreFileExtensions="png, jpg, jpeg, gif"/>
  </security>
</iphone>
<mobileWebApp version="1.0" cacheManifest=""/>
<worklightServerRootURL>http://{local.IPAd
</application>

```

For more information about Application Descriptor attributes, see “The application descriptor” on page 255.

The `<cacheManifest>` attribute accepts three values, as shown in the following table. No matter which value or mode is selected, if the Cache Manifest does not exist, the Worklight Studio builder generates the default Cache Manifest to give you something to start with. But after creating this file, the builder leaves the resulting Cache Manifest file in its default **no-use** mode unless you explicitly change the setting.

Table 46. `<cacheManifest>` properties

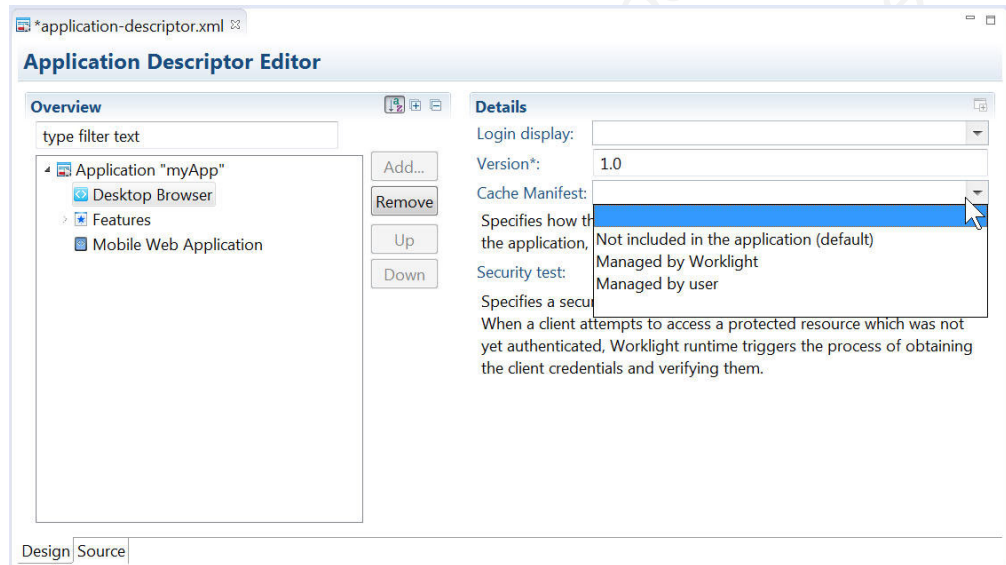
Property	Description
<b>generated</b>	<p>In this mode, the Worklight Studio builder generates a default Cache Manifest and includes it in the application's HTML files. The default Cache Manifest is generated depending on the environment:</p> <ul style="list-style-type: none"> <li>• For Desktop Browser environments – all resources are under NETWORK, which means: no cache at all.</li> <li>• For Mobile Web environments – all resources are under CACHE, which means: cache everything.</li> </ul> <p>In <b>generated</b> mode, in addition to creating the Cache Manifest, the builder creates a backup of the previous Cache Manifest, called <code>worklight.manifest.bak</code>. This file is overwritten in every build.</p>
<b>no-use</b>	<p>In this mode (which is the default), the Cache Manifest is not included in the application's HTML files. This setting means that there is no Cache Manifest and that decisions about which resources are cached are up to the browser.</p>

Table 46. <cacheManifest> properties (continued)

Property	Description
<b>user</b>	In this mode, the Worklight Studio builder does not generate the Cache Manifest, but it does include it in the application's HTML files. This setting means that the user must maintain the Cache Manifest manually.

## Editing the Cache Manifest

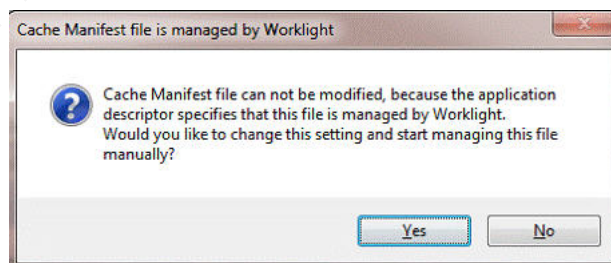
If you select the Application Descriptor (application-descriptor.xml file) in Design view, you can view and set the current mode of the <cacheManifest> attribute:



In this view, each of these attribute options is given a description:

- **Not Included in the application (default)** corresponds to **no-use** mode
- **Managed by Worklight** corresponds to **generated** mode
- **Managed by user** corresponds to **user** mode

The only <cacheManifest> mode that enables the user to edit the Cache Manifest is **user**. If you attempt to edit the file in any other mode, Worklight Studio displays the following message:



If you click **Yes** on this window, you can change the <cacheManifest> mode interactively and then continue to edit the file. You can also change the <cacheManifest> mode at any time with Worklight Studio's DDE editor.

The default <cacheManifest> setting for new Worklight projects is **Not Included in the application (no-use mode)**.

If your testing reveals that certain resources can be removed from the generated Cache Manifest, you can change the setting to **Managed by user (user mode)**. Then, you can edit the Cache Manifest and conduct more performance tests before you deploy to the production environment.

For example, you might notice that the Cache Manifest contains large images or other resources that are not used by the web application but need to remain in the development environment for other platforms. If you edit the Cache Manifest, you can remove them so that the web versions of this app load more quickly.

An example of a generated Cache Manifest file for a Desktop Browser environment is shown in the following sample:

```
CACHE MANIFEST
# Created: 2013-05-13 16:55:34 UTC
# Modifying this file is possible only when the Cache Manifest value in
# the application descriptor is set to "Managed by user"
common/js/base.js
common/js/busy.js
common/js/sjcl.min.js
common/js/wl_min.js
common/js/wlcommon.js
common/js/wljq.js
css/tptp.css
images/icon.png
images/icon114x114.png
images/icon57x57.png
images/icon72x72.png
images/thumbnail.png
js/initOptions.js
js/messages.js
js/tptp.js
tptp.html
wlclient/css/wlclient.css
wlclient/images/empty.gif
wlclient/js/analytics/Tealeaf.js
wlclient/js/analytics/analytics.js
wlclient/js/challengeHandlers/antiXSRFChallengeHandler.js
wlclient/js/challengeHandlers/authenticityChallengeHandler.js
wlclient/js/challengeHandlers/deviceAuthAutoProvisioningChallengeHandler.js
wlclient/js/challengeHandlers/deviceAuthNoProvisioningChallengeHandler.js
wlclient/js/challengeHandlers/remoteDisableChallengeHandler.js
wlclient/js/deviceAuthentication.js
wlclient/js/deviceSensors/acquisition.js
wlclient/js/deviceSensors/bind.js
wlclient/js/deviceSensors/geo.js
wlclient/js/deviceSensors/geoUtilities.js
wlclient/js/deviceSensors/triggers.js
wlclient/js/deviceSensors/wifi.js
wlclient/js/diagnosticDialog.js
wlclient/js/encryptedcache/encryptedcache.js
wlclient/js/encryptedcache/externs.js
wlclient/js/events/eventTransmitter.js
wlclient/js/features_stubs/jsonstore_stub.js
wlclient/js/messages.js
wlclient/js/window.js
wlclient/js/wlclient.js
wlclient/js/wlfragments.js
wlclient/js/worklight.js
```

NETWORK:

\*



## IBM Worklight application build settings

You can use minification to reduce the size of JavaScript and CSS files in your Mobile Web or Desktop Browser application. You can also use concatenation to improve the start time of the application. To do this, you use Worklight build settings.

In IBM Worklight V6.0.0, a new file was added to Worklight applications: `build-settings.xml`.

This file is created when a new Worklight application is created, and is on the same level as `application-descriptor.xml`. The purpose of the file is to prepare minification and concatenation configurations for each environment. These configurations are then used by the minify and concatenation engines during the build process.

The structure of the `build-settings.xml` file is as shown in the following example:

```
<buildSettings xmlns="http://www.ibm.worklight.com/build-settings">
  <common>
    <minification level="simple" includes="*" excludes="**/css/**"/>
    <concatenation includes="*" excludes="**/*.js"/>
  </common>
  <desktopBrowser>
    <minification level="simple" includes="*" excludes="**/css/**"/>
    <concatenation includes="*" excludes="**/*.txt"/>
  </desktopBrowser>
  <mobileWebApp>
    <minification level="simple" includes="*" excludes="**/css/**"/>
    <concatenation includes="*" excludes="**/*.js"/>
  </mobileWebApp>
</buildSettings>
```

The names of elements are aligned with names of environments. Only Mobile Web and Desktop Browser environments can be minified or concatenated, so only those individual environment elements can be configured. The `<common>` element contains configurations that are common to all environments.

All three elements – `<common>`, `<desktopBrowser>`, and `<mobileWebApp>` – are optional.

If any of these three elements are used, the `<minification>` attribute is mandatory within each one. Its `level` attribute specifies the compilation level of minification process and resources that can or cannot be used. Minification level options are listed in the following table.

Table 47. Options for the `<minification>` level attribute

Value	Description
<b>none</b>	No minification is done on your code by the Worklight Studio builder.
<b>whitespaces</b>	Removes comments from your code and also removes line breaks, unnecessary spaces, and other white space. The output JavaScript is functionally identical to the source JavaScript. (In the Worklight Studio Build Settings Editor, this attribute is called <b>Remove whitespaces and comments</b> .)

<b>simple</b>	Removes the same white space and comments as <b>whitespaces</b> , but also optimizes expressions and functions, including renaming local variables and function parameters to shorter names. Renaming variables to shorter names makes the code smaller. Because the <b>simple</b> setting renames only symbols that are local to functions, it does not interfere with the interaction between the compiled JavaScript and other JavaScript. Compilation with this setting always preserves the functionality of syntactically valid JavaScript, if the code does not access local variables with string names, for example, by using <code>eval()</code> statements. (In the Worklight Studio Build Settings Editor, this attribute is called <b>Google Closure Compiler Simple Optimization</b> .)
---------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

The `includes` and `excludes` attributes must be followed by a list of file names or regular expressions as used by Ant, separated by semicolons. Only JavaScript (.js) and Cascading Style Sheet (.css) files can be listed. Wildcard characters are allowed, with the following rules:

- `**` – includes or excludes all files and folders
- `**/foldername/**` – includes or excludes all files and folders under `foldername`
- `**/*.css` – includes or excludes all files in all folders that have an extension of `.css`

The `includes` and `excludes` attributes can be used in combination, such as in the following examples:

- `includes="**" and excludes="**/*.css"` contains all files except `.css` files
- `includes="**" and excludes="**/css/**"` contains all files except files under the `css` folder
- `includes="**/js/**"` contains only files that are found under the `js` folder
- `includes="**/*.js"` contains only files that have an extension of `.js`
- `includes="**/*.js" and excludes="**/*.css"` contains no files at all

For more information about minification, see “Minification of JS and CSS files” on page 354.

The `<concatenation>` element is optional. It contains no `level` attribute, and its `includes` and `excludes` attributes use the same syntax that is listed for the `<minification>` element.

For more information about concatenation, see “Concatenation of JS and CSS files” on page 356.

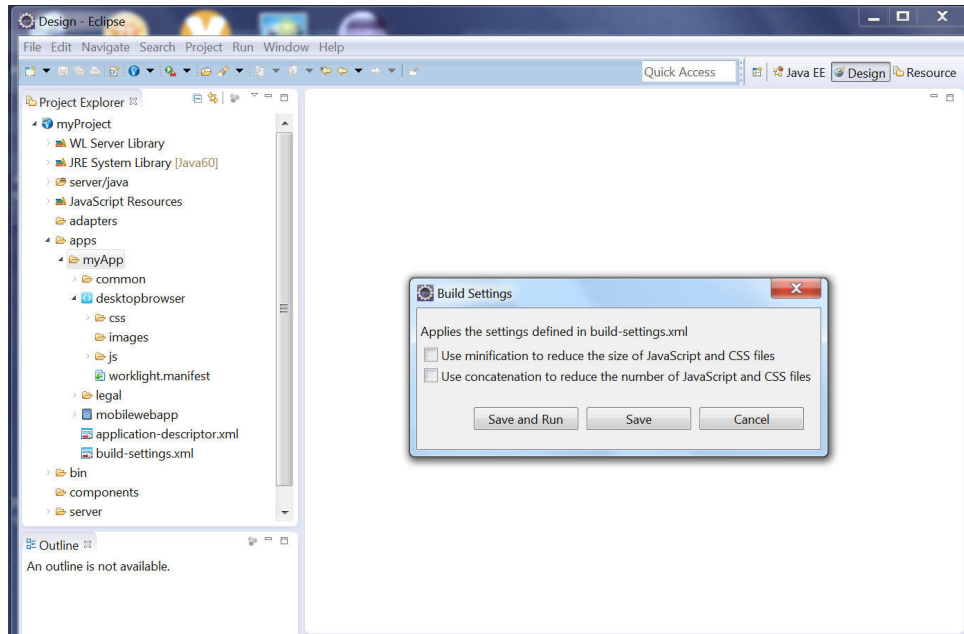
### To turn on minification or concatenation for an environment

To instruct Worklight Studio to use minification, concatenation, or both when it builds the application:

1. In Worklight Studio, right-click the **desktopbrowser** or **mobilewebapp** element of your application and choose **Run As > Change Build And deploy Settings...** from the menu.

The Build and Deploy Settings window is displayed:



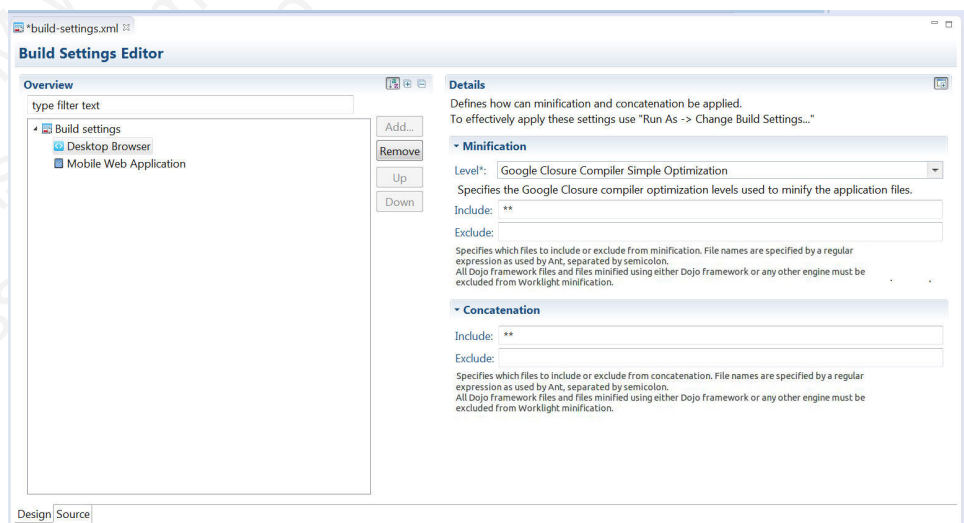


2. Select the check box of the feature or features you want to use when you build this environment.
3. Click either **Save** or **Save and Run**.

### To edit the build-settings.xml file

Similar to the application-descriptor.xml file, the build-settings.xml can be edited with the Eclipse DDE editor:

1. In Worklight Studio, double-click the **build-settings.xml** element of your application to display the Build Settings Editor:



2. To create a configuration for **Concatenation**:
  - a. Enter the list of files to be concatenated or not concatenated in the **includes** and **excludes** fields. Use the Ant syntax that is described earlier.
3. To create a configuration for **Minification**:
  - a. Select the wanted minification level from the **Level** field:
    - **None (Default)** specifies the **none** attribute in the above table.

- **Remove whitespaces and comments** specifies the **whitespaces** attribute in the above table.
  - **Google Closure Compiler Simple Optimization** specifies the **simple** attribute in the above table.
- b. Enter the list of files to be minified or not minified in the **includes** and **excludes** fields. Use the Ant syntax that is described earlier.

The `build-settings.xml` can also be edited with a standard XML editor. If it is not already present, the `<common>` element can be added only with an XML editor. See “IBM Worklight application build settings” on page 351 for examples of the XML syntax.

## Building with the `build-settings.xml` file

At build time, the Worklight Studio builder minifies or concatenates all the files that are included and not excluded, as defined in the `build-settings.xml` file.

During the build process, when either minification or concatenation are specified for an environment, the builder reads the `build-settings.xml` file and configures the compilation level and included and excluded files for that environment. Each environment is minified or concatenated according to its own configuration, and according to the following rules:

- The `compilation level` value of the environment overrides the `compilation level` specified in the `<common>` element.
- The `includes` attribute of each environment overrides an `includes` attribute of `<common>`.
- The `excludes` attribute of each environment is concatenated to the `excludes` attribute of `<common>`.

By editing the `build-settings.xml` file, you can essentially create different configurations for minification and concatenation, depending on the stage of the development cycle. For example, you might have one setting that is commonly used during development, in which the minification level is set to **none** and the concatenation feature is disabled. But when you move the application to production, you can edit the build settings to use a minification level of **simple** and to enable concatenation.

## Minification of JS and CSS files

Settings within Worklight Studio enable you to minimize the size of JavaScript and CSS files deployed with your Desktop Browser and Mobile Web applications.

*Minification* is the process that minifies web resources to make them smaller. The smaller size of the resources means less traffic between the Worklight application and Worklight Server. This is true both when the app is being initially downloaded by users, and at application start time. The feature is a counterpart to another build optimization, *concatenation*, and is almost always used in conjunction with it. Use of these features can either improve the applications' start time (concatenation), or reduce the size of the application (minification).

Minification is done at build time by the Google Closure Compiler. There are three levels of minification that can be used in an IBM Worklight application, as listed in the following table:

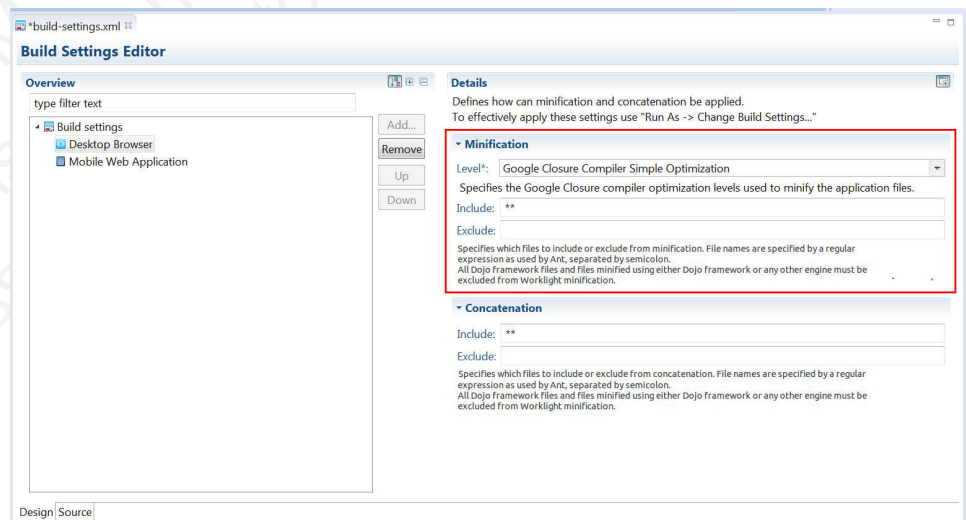
Table 48. Options for the <minification> level attribute

Value	Description
<b>none</b>	No minification is done on your code by the Worklight Studio builder.
<b>whitespaces</b>	Removes comments from your code and also removes line breaks, unnecessary spaces, and other white space. The output JavaScript is functionally identical to the source JavaScript. (In the Worklight Studio Build Settings Editor, this attribute is called <b>Remove whitespaces and comments</b> .)
<b>simple</b>	Removes the same white space and comments as <b>whitespaces</b> , but also optimizes expressions and functions, including renaming local variables and function parameters to shorter names. Renaming variables to shorter names makes the code smaller. Because the <b>simple</b> setting renames only symbols that are local to functions, it does not interfere with the interaction between the compiled JavaScript and other JavaScript. Compilation with this setting always preserves the functionality of syntactically valid JavaScript, if the code does not access local variables with string names, for example, by using <code>eval()</code> statements. (In the Worklight Studio Build Settings Editor, this attribute is called <b>Google Closure Compiler Simple Optimization</b> .)

## To configure minification in Worklight Studio

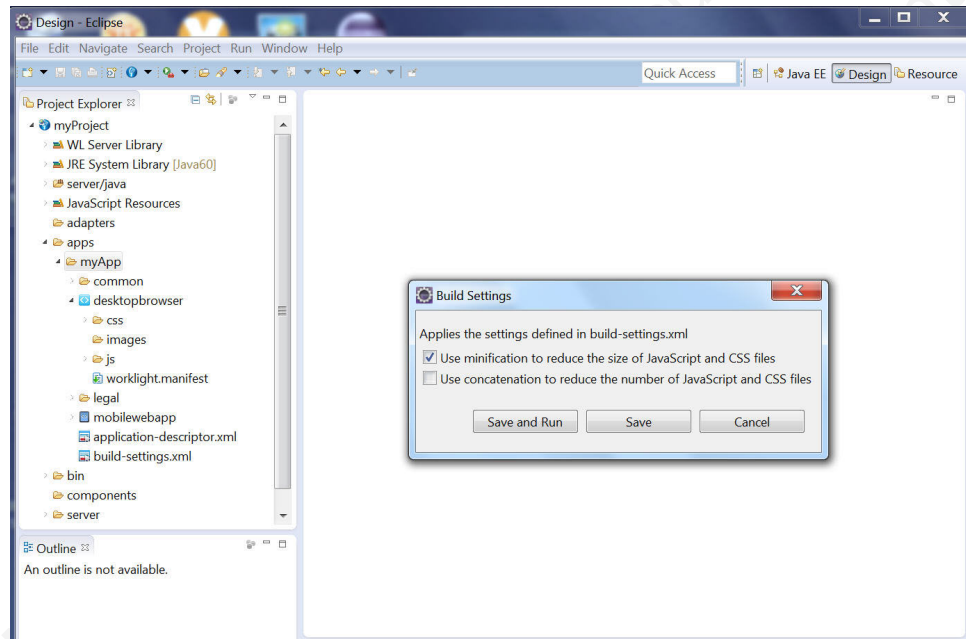
You create a minification configuration for your Mobile Web or Desktop Browser application in two steps. First, you edit the Build Settings for the application, and then you turn on minification for the individual environments.

1. To configure minification, in Worklight Studio, double-click the **build-settings.xml** element of your application to display the Build Settings Editor and **Add** or select the environment:



2. Select the wanted minification level from the **Level** field:
  - **None (Default)** specifies the **none** attribute in the above table.
  - **Remove whitespaces and comments** specifies the **whitespaces** attribute in the above table.

- **Google Closure Compiler Simple Optimization** specifies the **simple** attribute in the above table.
3. Enter the list of files to be minified or excluded from minification in the **includes** and **excludes** fields. When you save, these settings become part of the application code.  
Use the Ant syntax that is described in “IBM Worklight application build settings” on page 351.
  4. To instruct Worklight Studio to use concatenation during the build, in Worklight Studio, right-click the **desktopbrowser** or **mobilewebapp** element of your application and choose **Run As > Apply Build Settings...** from the menu.
  5. Select the check box labeled **Use minification to reduce the size of JavaScript and CSS files**, as shown in the following screen capture:



6. Click **Save**.
7. Rebuild your application. No changes take place after an edit of the minification parameters until after the next build.

The `build-settings.xml` can also be edited with a standard XML editor, and can be invoked using Ant scripts. See “IBM Worklight application build settings” on page 351 for examples of the XML syntax.

## Concatenation of JS and CSS files

Worklight Studio allows concatenation of multiple JavaScript and CSS files that are deployed with your Desktop Browser and Mobile Web applications.

As of IBM Worklight V6.0.0, the *concatenation* feature allows concatenation of the multiple web resources that are used by the application (JavaScript and CSS files) into a smaller number of files. Reducing the total number of files that are referenced by the application HTML results in fewer browser requests when the application starts up, which allows the application to start more quickly.

Concatenation is available for Desktop Browser and Mobile Web environments only. The feature is a counterpart to another build optimization, *minification*, and is

almost always used in conjunction with it. Use of these features can either reduce the size of Worklight applications (minification) or improve their start time (concatenation).

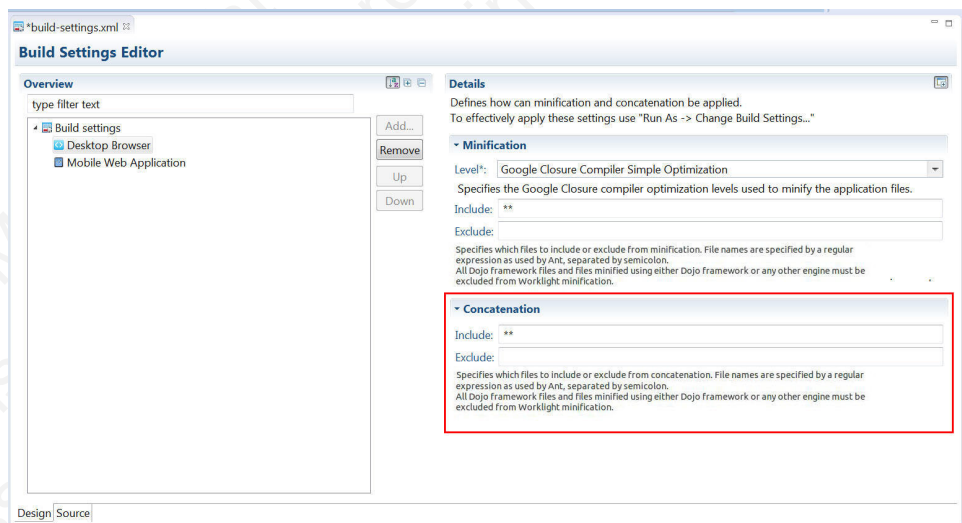
During concatenation, several resources (for example, JavaScript files and inline scripts) are copied into a new file, which is then referenced by the application HTML. References to the original resources are removed from the HTML. This means that less communication between the device and web server is required to retrieve the application code.

At build time, the concatenation algorithm determines which resources to concatenate into which files. Concatenation is controlled by a number of different parameters, such as the structure of the HTML, the type of the resources to be concatenated, and the attributes of these resources. The order of the resources in the HTML is preserved. As a result, the concatenation process does not have any negative effects in terms of code dependencies or functionality.

## To configure concatenation in Worklight Studio

You create a concatenation configuration for your Mobile Web or Desktop Browser application in two steps. First, you edit the Build Settings for the individual environments, and then you turn on concatenation for the application.

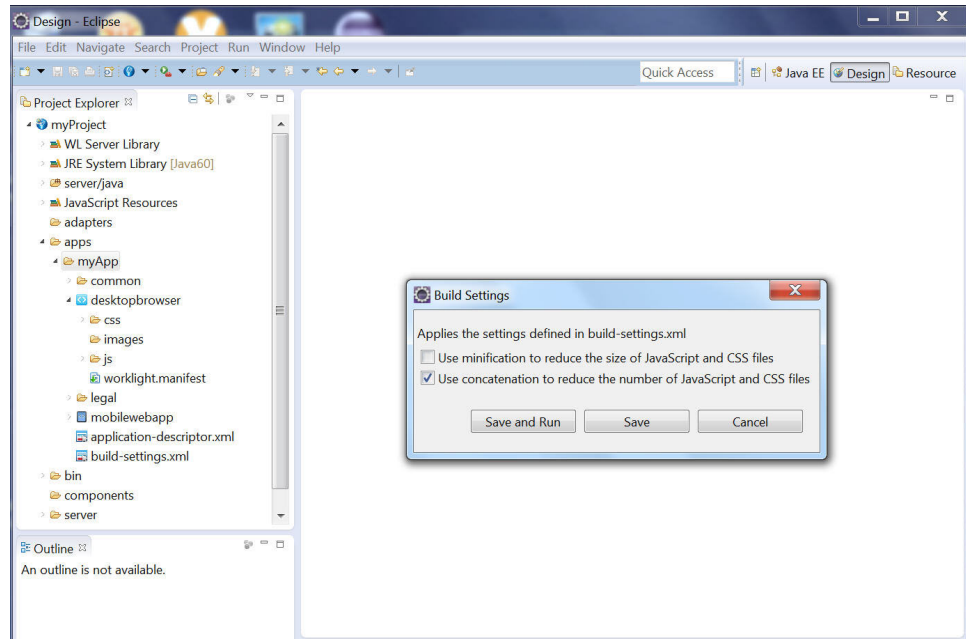
1. To enter the list of files to be concatenated, in Worklight Studio, double-click the **build-settings.xml** element of your application to display the Build Settings Editor and **Add** or select the environment:



2. Enter the list of files to be concatenated or not concatenated in the **includes** and **excludes** fields. When you save, these settings become part of the application code.

Use the Ant syntax that is described in the following “Syntax and examples” on page 358 section and in “IBM Worklight application build settings” on page 351.

3. To instruct Worklight Studio to use concatenation during the build, in Worklight Studio, right-click the appropriate **desktopbrowser** or **mobilewebapp** element of your application and choose **Run As > Apply Build Settings...** from the menu.
4. Select the check box labeled **Use concatenation to reduce the number of JavaScript and CSS files**, as shown in the following screen capture:



5. Click **Save**.
6. Rebuild your application. No changes take place after an edit of the concatenation parameters until after the next build.

The `build-settings.xml` can also be edited with a standard XML editor, and can be invoked using Ant scripts.

## Syntax and examples

The `includes` and `excludes` attributes must be followed by a list of file names or regular expressions as used by Ant. Only JavaScript (`.js`) and Cascading Style Sheet (`.css`) files can be listed. Wildcard characters are allowed, with the following rules:

- `**` – includes or excludes all files and folders
- `**/foldername/**` – includes or excludes all files and folders under `foldername`
- `**/*.css` – includes or excludes all files in all folders that have an extension of `.css`
- Multiple file names or regular expressions are separated by semicolons.
- Included files are concatenated (all files are included by default).
- Excluded files are not concatenated (no files are excluded by default).
- Files that are excluded or not included are not part of the concatenation process.
- In most cases, setting the included list to `**` (the default value – all files) and modifying only the excluded list is sufficient to achieve the wanted results

In practice, users often create more specific `excludes` definitions, relying on wildcards to include the remaining files. For example, JavaScript files with the `async` attribute might be good candidates for exclusion, as it might not make sense to concatenate their content with other files.

The following example shows an IBM Worklight HTML file that contains the standard resources that are provided by the IBM Worklight, along with other resources defined by the user:



```

<html>
<head>
...
<link rel="stylesheet" href="css/myApp.css">
<link rel="stylesheet" href="css/myStyle.css">
<link rel="stylesheet" href="css/myStyle2.css">
<link rel="stylesheet" href="css/myStyle3.css">

<script>window.$ = window.jQuery = WLJQ;</script>
<script src="js/myJSFile.js"></script>
<script src="js/myJSFile2.js"></script>
<script src="js/myJSFile3.js" async></script>
<script src="js/myJSFile4.js"></script>
<script src="js/myJSFile5.js"></script>

</head>
<body id="content" style="display: none;">
...
<script src="js/initOptions.js"></script>
<script src="js/myApp.js"></script>
<script src="js/messages.js"></script>
</body>
</html>

```

After the concatenation process (as part of the build), the resulting HTML file has the following structure:

```

<html>
  <head>
    ...
    <link href="wlclient/css/wlclient.css" rel="stylesheet">
    ...
    <link href="css/wlconcatenated0.css" rel="stylesheet">

    <script>
      ... WL framework initialization code ...
    </script>

    <script src="common/js/wljq.js"></script>
    <script src="wlconcatenatedhead0.js"></script>
    <script src="wlconcatenatedhead1.js"></script>
    <script async="" src="js/myJSFile3.js"></script>
    <script src="wlconcatenatedhead2.js"></script>

  </head>
  <body>
    ...
    <script src="wlconcatenatedbody0.js"></script>
  </body>
</html>

```

The following changes were made to the HTML in the concatenation process:

- All of the CSS files under the `css` folder were concatenated into a single file, `wlconcatenated0.css`. Note the file `wlclient.css`, which is not concatenated, because it is located under a separate folder.
- All of the Worklight framework files were concatenated into two files – `wljq.js` and `wlconcatenatedhead0.js`.
- The inline script and the files `myJSFile.js` and `myJSFile2.js` were concatenated into the file `wlconcatenatedhead1.js`.



- The file `myJSFile3.js` contains the `async` attribute, and so it was not concatenated into another file.
- The files `myJSFile4.js` and `myJSFile5.js` were concatenated into the file `wlconcatenatedhead2.js`.
- In the body, the files `initOptions.js`, `myApp.js` and `messages.js` were concatenated into the file `wlconcatenatedbody0.js`

In this example, the number of resources that are referenced by the HTML is greatly reduced. The number of application resources and user-defined resources is reduced from 12 to 5, and only three files are used for all of the Worklight framework resources. This reduction results in fewer requests by the browser, leading to a faster application startup time.

---

## 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 IBM Worklight. 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 IBM Worklight.

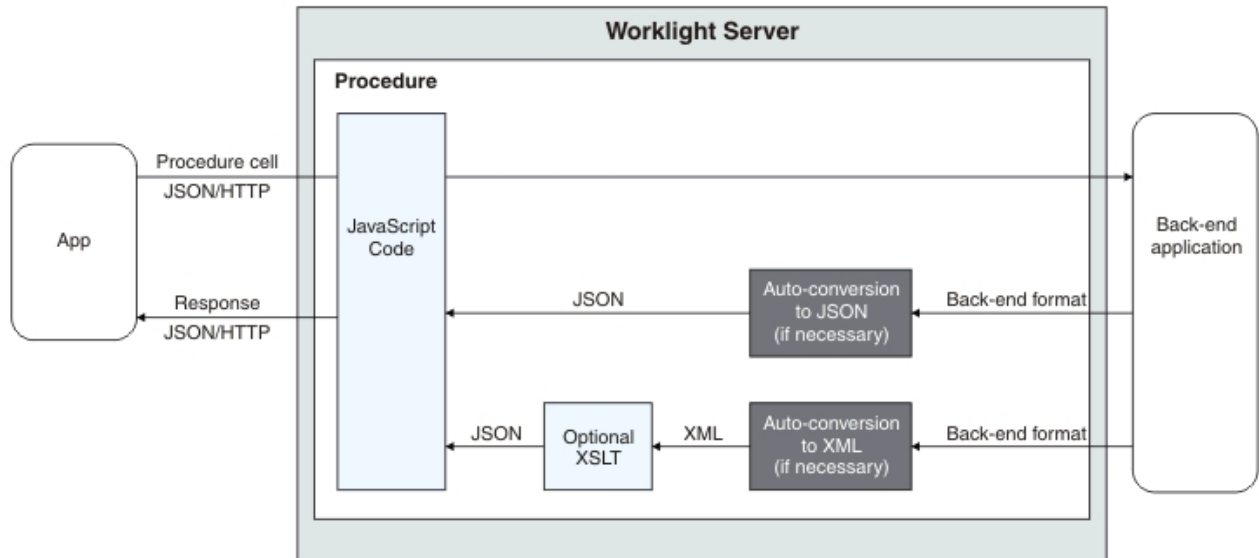


Figure 33. The adapter framework

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 by using JNDI environment entries. The keystore setup process is described in “SSL certificate keystore setup” on page 720.
- 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 by using JNDI environment entries.
  - 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 368.

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.

## 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 “WL.Server.createEventSource” on page 637.

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.

- 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:

Table 49. <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. <b>Note:</b> 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:w1 – 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 Worklight Studio.  at run time, this attribute has no effect.

The <adapter> element has the following subelements:

Table 50. <adapter> element subelements

Subelement	Description
<displayName>	<b>Note:</b> This element is deprecated.  Optional. The name of the adapter to be displayed in the Worklight Console.  If the <displayName> element is not specified, the value of the name attribute is used instead in the Worklight Console.
<description>	Optional. Additional information about the adapter, which is displayed in the 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 366.

Table 50. <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 367.

### <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 51. <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 368

The structure of the <ConnectionPolicy> element.

“The <connectionPolicy> element of the SQL adapter” on page 372

The <connectionPolicy> element of the SQL adapter configures how the adapter connects to an SQL database.

“The <connectionPolicy> element of the JMS adapter” on page 375

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.



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.

When deploying adapters to a cluster, set the value of this attribute to the maximum required load divided by the number of cluster members.

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 52. <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.

Table 52. <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: <ul style="list-style-type: none"> <li>server: Default. The connection to the back end is created according to the connection policy defined for the adapter.</li> <li>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.</li> </ul>
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 <i>Worklight Project Name/server/log/audit/audit.log</i> .  Valid values are as follows: <ul style="list-style-type: none"> <li>true: Calls to the procedure are logged in the audit log.</li> <li>false: Default. Calls to the procedure are not logged in the audit log.</li> </ul>
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>
  <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 53. <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: <ul style="list-style-type: none"> <li>• RFC_2109 (The default)</li> <li>• RFC_2965</li> <li>• NETSCAPE</li> <li>• IGNORE_COOKIES</li> </ul>
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 54. <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 720

Table 54. <ConnectionPolicy> element subelements (continued)

Subelement	Description
sslCertificatePassword	<p>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.</p> <p>Optional for regular HTTP authentication and simple SSL authentication. Mandatory for mutual SSL authentication.</p> <p>The keystore setup process is described in “SSL certificate keystore setup” on page 720</p>
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 371.

### 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 workstation="value"/>
</authentication>
```

The workstation attribute is optional, and denotes the name of the computer on which the IBM Worklight Server runs. Its default value is `${local.workstation}`.

- 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

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 <http://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> ${DOMAIN\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 370.

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=
```

```
"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 <connectionPolicy> element of the SQL adapter configures how the adapter connects to an SQL database.

The <connectionPolicy> element has two options for connecting:

- Using the <dataSourceDefinition> subelement
- Using the <dataSourceJNDIName> subelement

### Connecting by using the <dataSourceDefinition> subelement

When you use this option, you specify the URL of the data source, the user, the password, and the driver class. Note that this method is primarily intended for development mode. In production mode, it is better to use the <dataSourceJNDIName> subelement.

The following example shows the structure of the <connectionPolicy> element with the <dataSourceDefinition> subelement:

```
<connectionPolicy xsi:type="sql:SQLConnectionPolicy">
  <dataSourceDefinition>
    <driverClass>com.mysql.jdbc.Driver</driverClass>
    <url>jdbc:mysql://localhost:3306/mysqldbname</url>
    <user>mysqluser</user>
    <password>mysqlpassword</password>
  </dataSourceDefinition>
</connectionPolicy>
```

Table 55. <connectionPolicy> element attributes

Attribute	Description
xsi:type	Mandatory. The value of this attribute must be set to sql:SQLConnectionPolicy.

The <connectionPolicy> element has the following subelement:

Table 56. <ConnectionPolicy> element subelement

Subelement	Description
dataSourceDefinition	Mandatory. Contains the parameters needed to connect to a data source.

The parameters (**url**, **user**, **password**, and **driverClass**) can be externalized as custom Worklight properties, and can then be overridden by environment entries. For more information, see “Configuring an IBM Worklight project in production by using JNDI environment entries” on page 723.

The following example illustrates this process:

#### adapter.xml :

```
<connectionPolicy xsi:type="sql:SQLConnectionPolicy">
  <dataSourceDefinition>
    <driverClass>com.mysql.jdbc.Driver</driverClass>
    <url>${my-mysql-url}</url>
    <user>${my-mysql-user}</user>
    <password>${my-mysql-password}</password>
```

```
</dataSourceDefinition>
</connectionPolicy>
```

**worklight.properties:**

```
my-mysql-url=jdbc:mysql://localhost:3306/mysqldbname
my-mysql-user=worklight
my-mysql-password=worklight
```

### Connecting by using the <dataSourceJNDIName> subelement

You can also connect to the data source by using the JNDI name of a data source that is provided by the application server. Application servers provide a way to configure data sources. For more information, see “Creating and configuring the databases manually” on page 678.

When you configure a data source that is provided by the application server, the data source must have a JNDI name. This name can be used by applications that run inside the container, to get a reference to the data source, and to use it.

The following example shows the structure of the <connectionPolicy> element with the <dataSourceJNDIName> subelement:

**adapter.xml:**

```
<connectionPolicy xsi:type="sql:SQLConnectionPolicy">
  <dataSourceJNDIName>jdbc/myAdapterDS</dataSourceJNDIName>
</connectionPolicy>
```

In this example, a resource with the JNDI name: “jdbc/myAdapterDS” must be declared inside the container.

The <ConnectionPolicy> element has the following attribute:

Table 57. <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:

Table 58. <ConnectionPolicy> element subelement

Subelement	Description
dataSourceJNDIName	Mandatory. The JNDI name of the data source.

You also have the option to externalize the data source JNDI name and make it configurable from the server configuration:

**adapter.xml:**

```
<connectionPolicy xsi:type="sql:SQLConnectionPolicy">
  <dataSourceJNDIName>${my-adapter-ds}</dataSourceJNDIName>
</connectionPolicy>
```

**worklight.properties:**

```
my-adapter-ds=jdbc/myAdapterDS
```

For more information, see “Configuring an IBM Worklight project in production by using JNDI environment entries” on page 723.



## 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 59. <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 60. <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.

## 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 61. <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 62. <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."
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" on page 376.

## The <namingConnection> element of the JMS adapter

Use the <namingConnection> element to identify how the 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: <i>iiop://localhost</i> . The url syntax is dependent on the JNDI provider.
initialContextFactory	Mandatory. The initialContextFactory class name of the JNDI provider. For example: <i>com.ibm.Websphere.naming.WsnInitialContextFactory</i> . The driver, and any associated files, must be placed in the <i>/server/lib</i> directory. If you develop in the Eclipse environment, the driver and associated files must be placed in the <i>/lib</i> directory. <b>Note:</b> If you develop for WebSphere Application Server with WebSphere MQ, do not add the WebSphere MQ Java archive (JAR) files to the <i>/lib</i> 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 <i>guest</i> .
password	Optional. Password for the user specified in the user attribute. If user is not specified, the default password is <i>guest</i> .

### The <jmsConnection> element of the JMS adapter

Use the <jmsConnection> element to identify how the 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. <b>Note:</b> If you are deploying in WebSphere Application Server, the connection factory must be a global JNDI object. The object must be addressed without the <i>java:comp/env</i> context. For example: <i>jms/MyConnFactory</i> and not <i>java:comp/env/jms/MyConnFactory</i> . However, if you are deploying in Tomcat, the connection factory must be addressed including the <i>java:/comp/env</i> context. For example: <i>java:comp/env/jms/MyConnFactory</i> .
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, 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**.

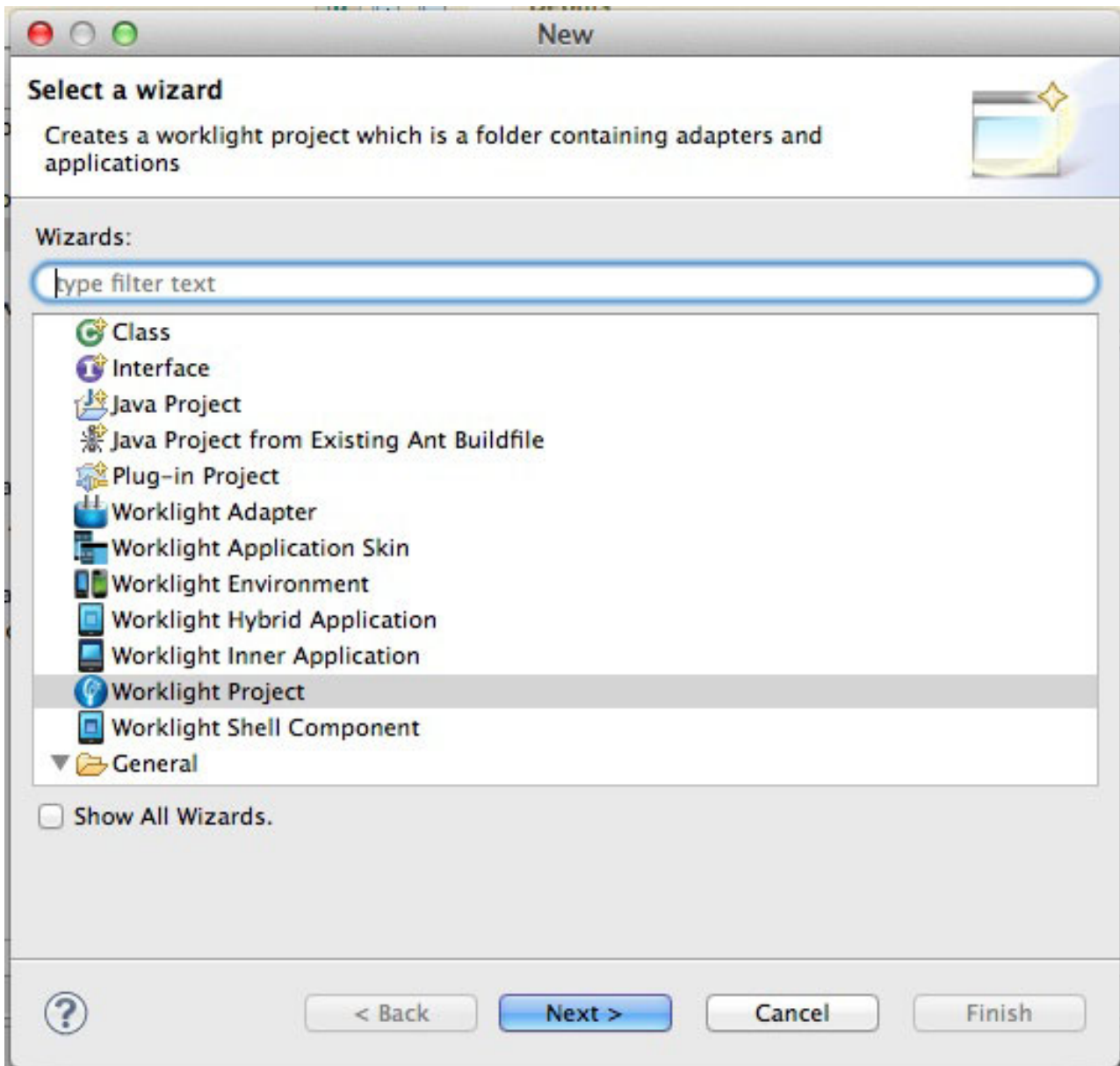


Figure 34. Creating an IBM Worklight project from the wizard.

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**.

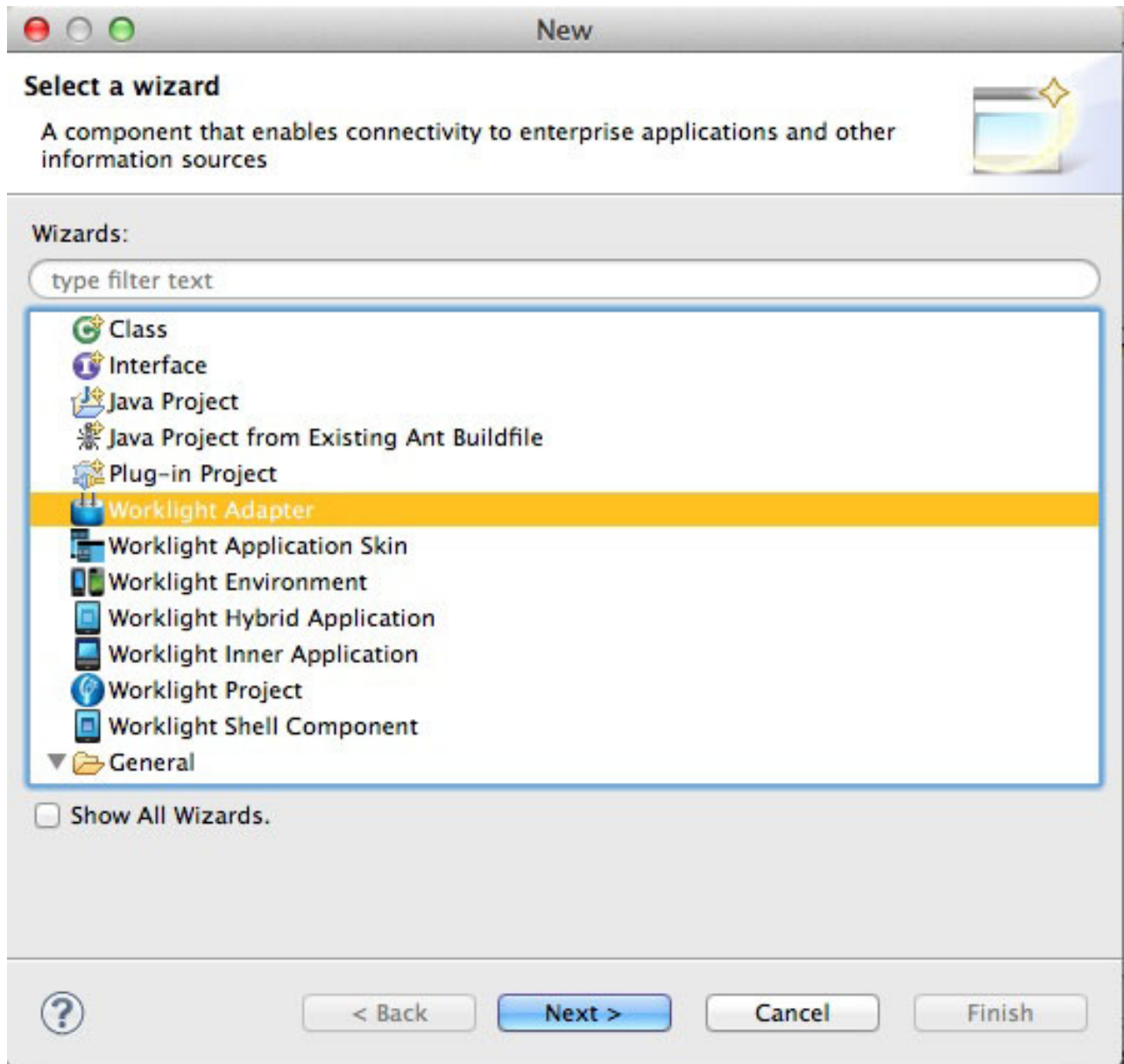


Figure 35. Configuring a new IBM Worklight adapter.

- 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**.

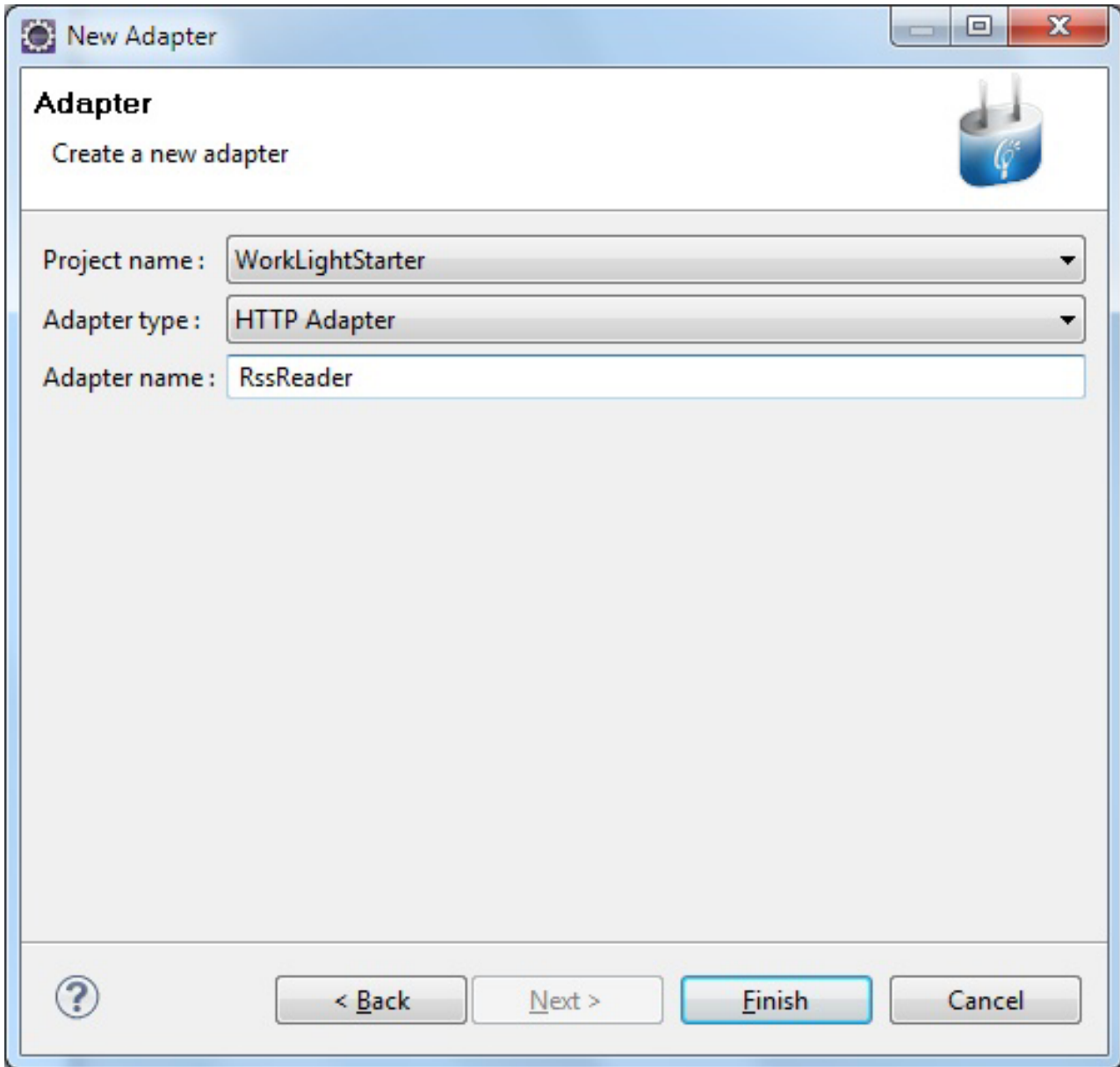


Figure 36. Selecting an adapter type.

## 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 63. Parameters for adapter invocation

Property	Description
<b>adapter</b>	The name of the adapter
<b>procedure</b>	The name of the procedure
<b>parameters</b>	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 “IBM Worklight security framework” on page 417 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 424 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:

```
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.SOAPMessageFactory
```

You must then restart the JVM.

**Important:** This workaround is only for IBM WebSphere.

**Related information:**

 [http://www.w3schools.com/xml/xml\\_e4x.asp](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 Worklight Studio automatically builds them and deploys them to the 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 Worklight Studio

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.

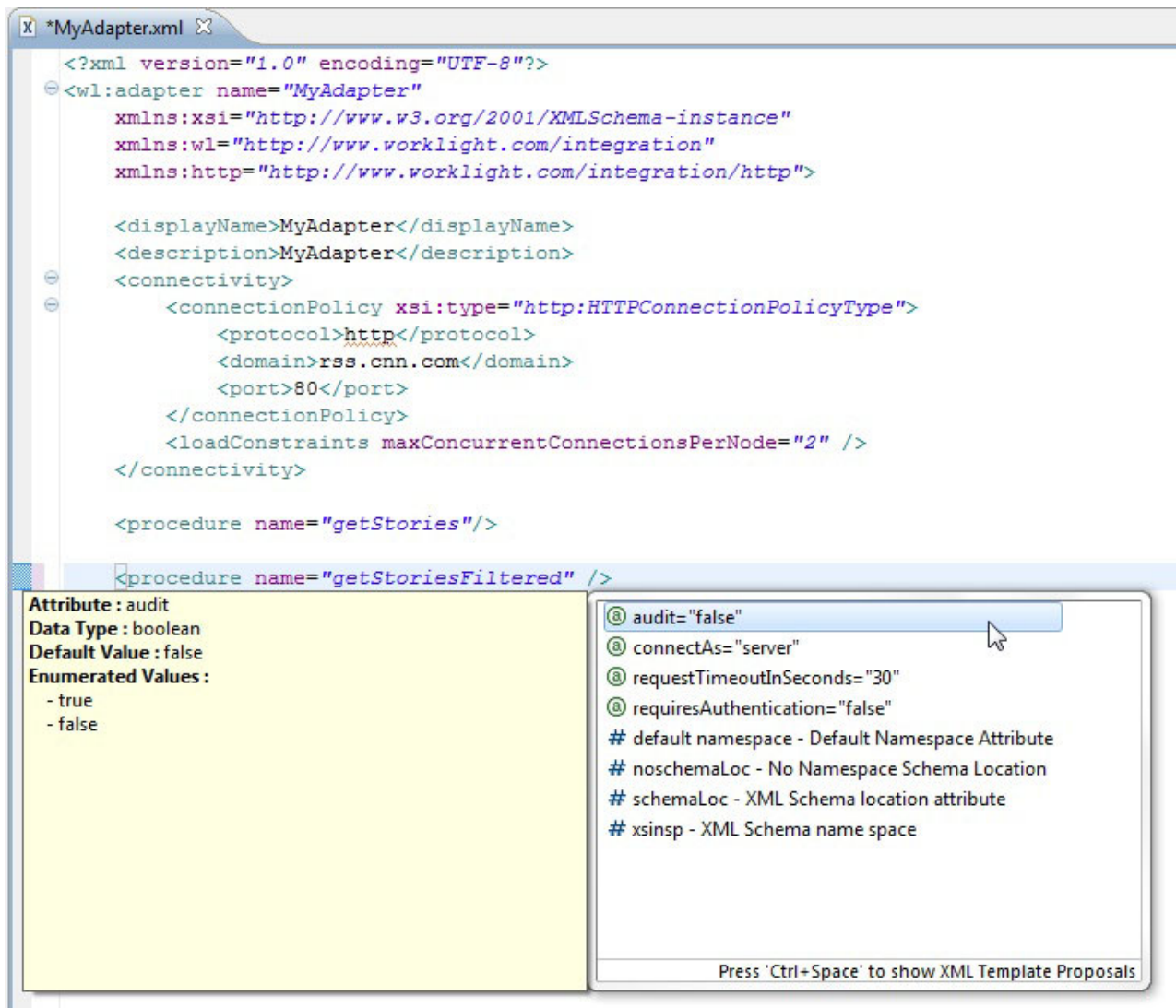


Figure 37. Adapter configuration through the auto-complete feature.

## Adapter validation

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

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

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 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.

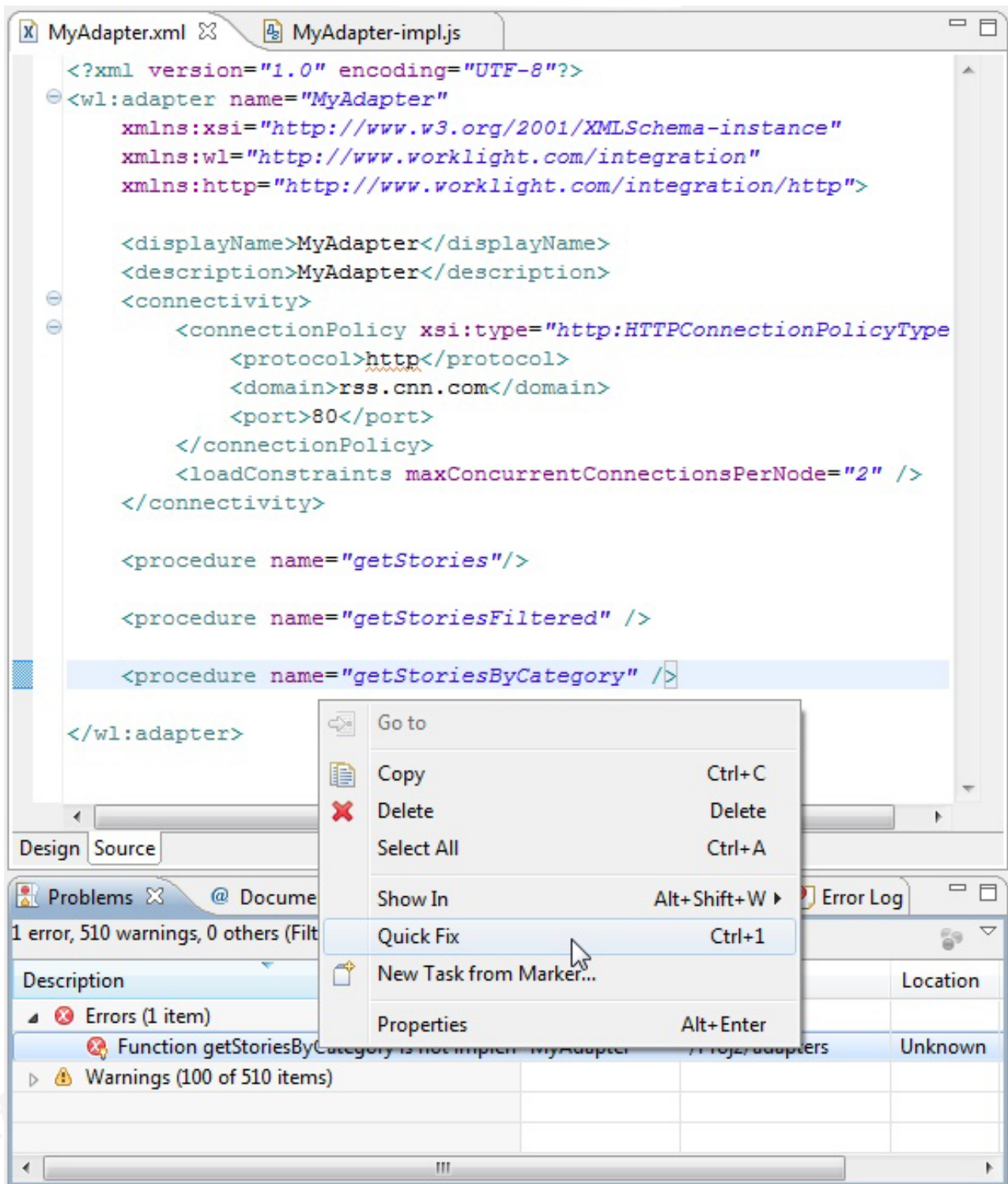


Figure 38. Quick Fix options for adapter configuration problems.



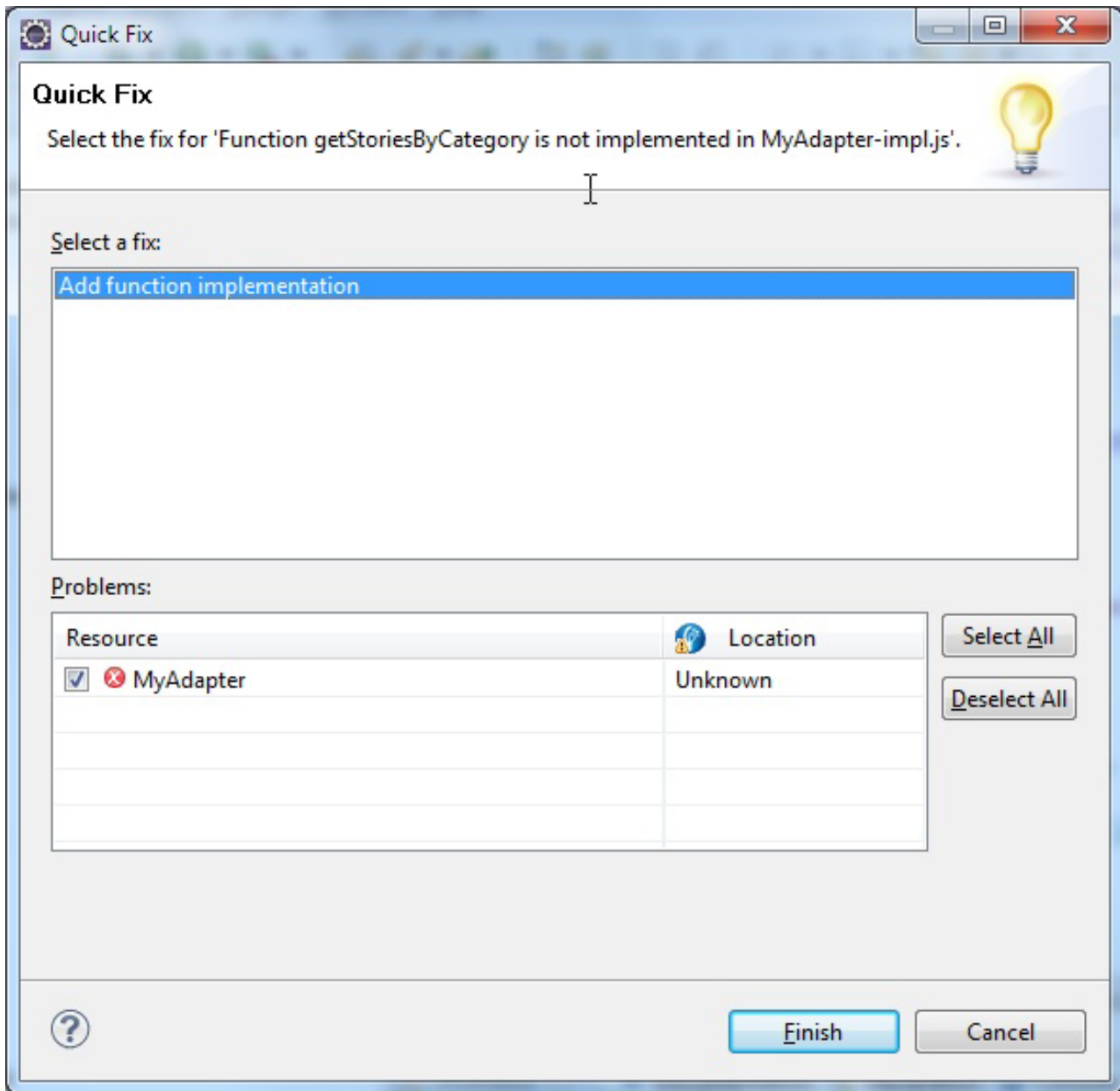


Figure 39. Quick Fix option for missing JavaScript functions.

Specifically, 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 Worklight Studio. It is not available when you run an adapter on a stand-alone server based on WebSphere Application Server or Tomcat.

In Worklight Studio, you can select a procedure, enter a set of parameters, and invoke the procedure on the Worklight Server. Only procedure invocations are supported, with results displayed in a browser window. For each invoked procedure, the Worklight Studio remembers the most recent parameter values, so you can reinvoke the procedure without re-entering parameter values.

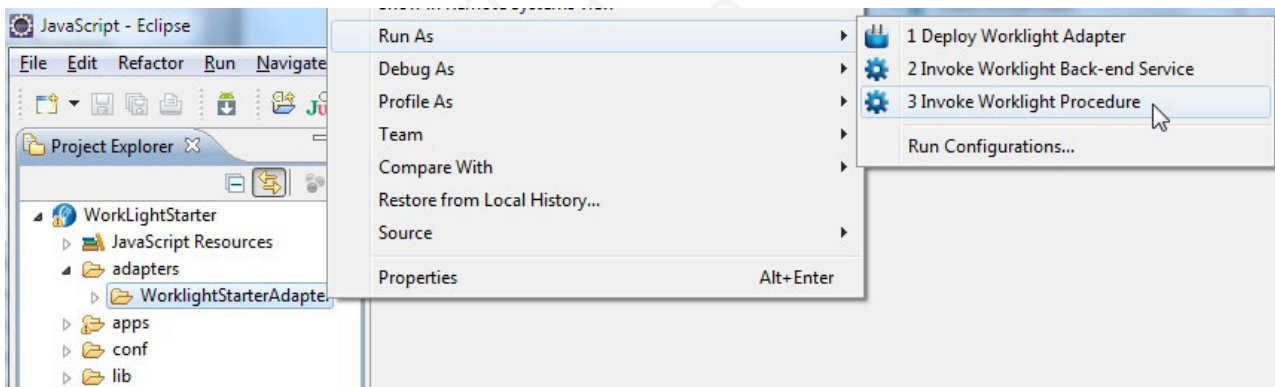


Figure 40. Invoking IBM Worklight procedures.

In the dialog box, provide a comma-separated list of procedure parameters.

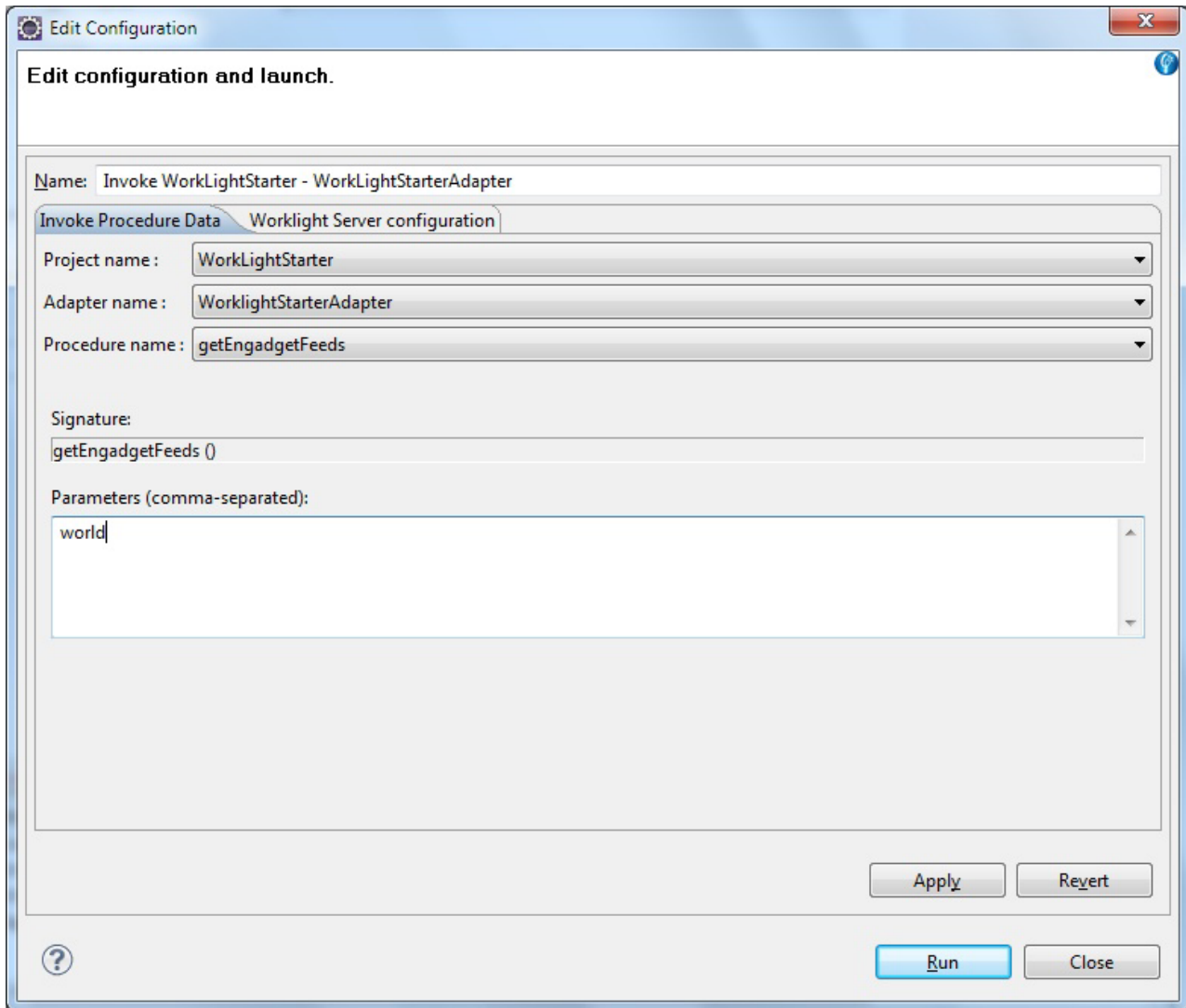


Figure 41. Launching the procedure.

## Invoking a back-end service

You can invoke a back-end service and receive the data retrieved by the service, in Worklight studio.

### About this task

**Note:** This feature is only available when running within Worklight Studio. It is not available when running an adapter on a stand-alone server based on WebSphere Application Server or Tomcat.

In 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.

## Procedure

To run a back-end service:

1. Right-click an adapter file, and select **Run As > Invoke Worklight Back-end Service**.

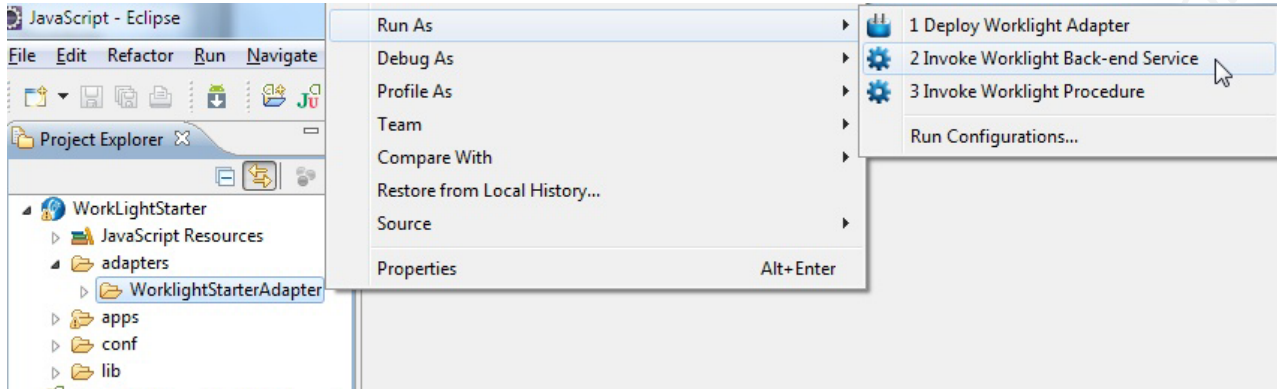


Figure 42. Invoking an IBM Worklight back-end service

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.

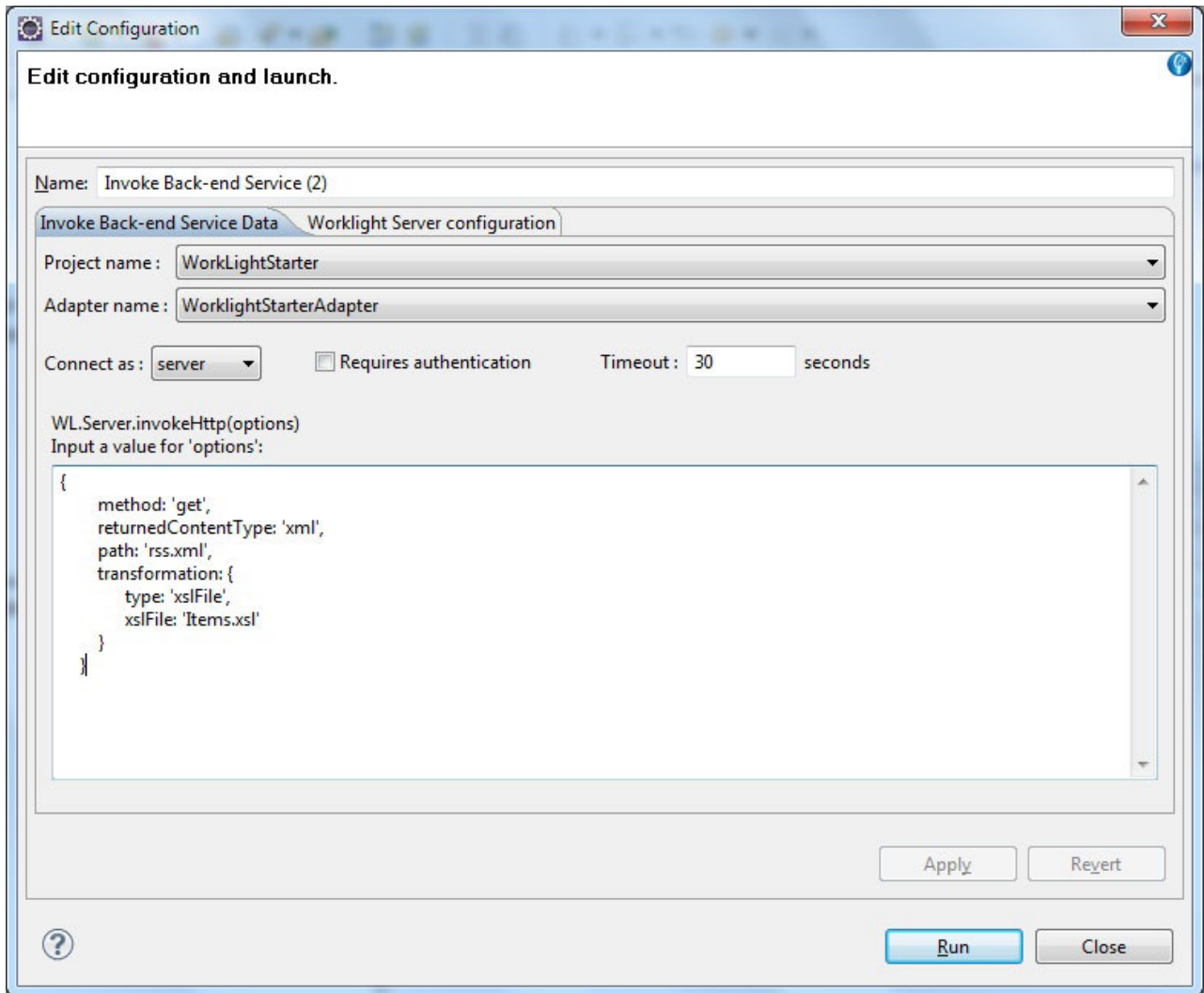


Figure 43. Invocation parameters.

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.

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://12.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:

```
{
  "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 1 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 1 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 44. Browser window, showing retrieved data in XML and JSON format.

## Deploying an adapter

In 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**.

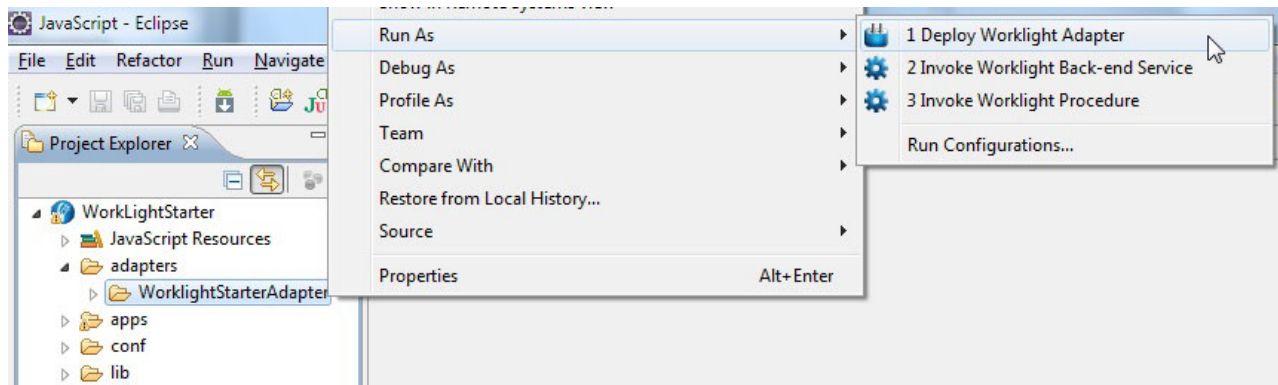


Figure 45. Deploying a Worklight adapter.

## JSONStore overview

JSONStore features add the ability to store JSON documents in IBM Worklight applications. Accessing JSONStore data from native code is not possible. JSONStore is only available in hybrid environments.

JSONStore is a lightweight, document-oriented storage system included as a feature of IBM Worklight, and enables persistent storage of JSON documents. Documents in an application are available in JSONStore even when the device running the application is offline. This persistent, always-available storage can be useful for customers, employees, or partners, to give them access to documents when for example there is no network connection to the device.

For JSONStore API reference information see “WL.JSONStore API” on page 531 in the API reference section.

Here is a high-level summary of what JSONStore provides:

- A developer-friendly JavaScript API to interact with JSONStore, giving developers the ability to populate the local store with documents, and to update, delete, and search across documents.
- Persistent, file-based storage matched the scope of the application.
- AES 256 encryption of stored data provides security and confidentiality. You can segment protection by user with password-protection, in the case of more than one user on a single device.
- Ability to integrate with IBM Worklight adapters to send changes and refresh local content in the JSONStore.

A single store can have many collections, and each collection can have many documents. It is also possible to have a IBM Worklight Application containing multiple stores. For information see “JSONStore multiple user support” on page 406. For a table summarizing the JSONStore features, see “WL.JSONStore API” on page 531



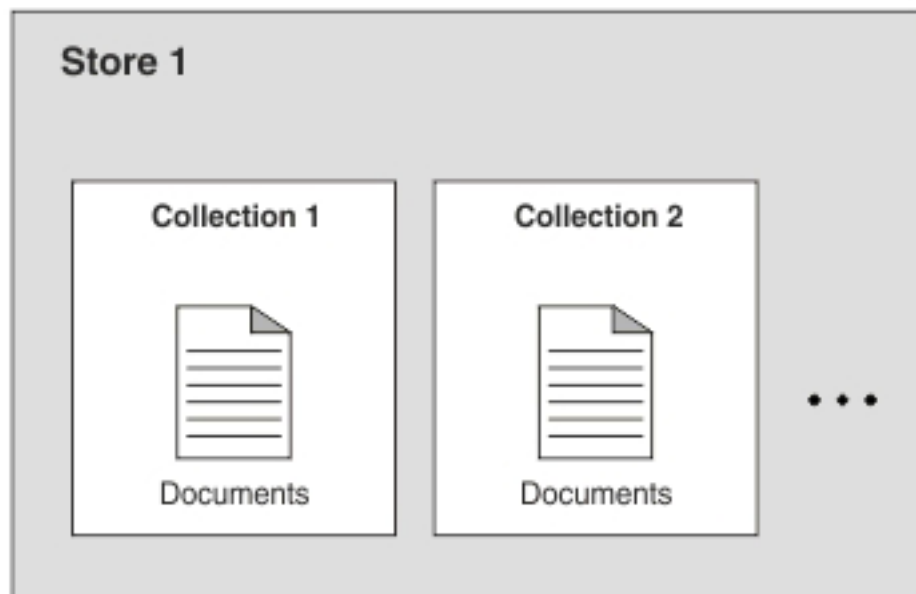


Figure 46. A basic graphic representation of JSONStore.

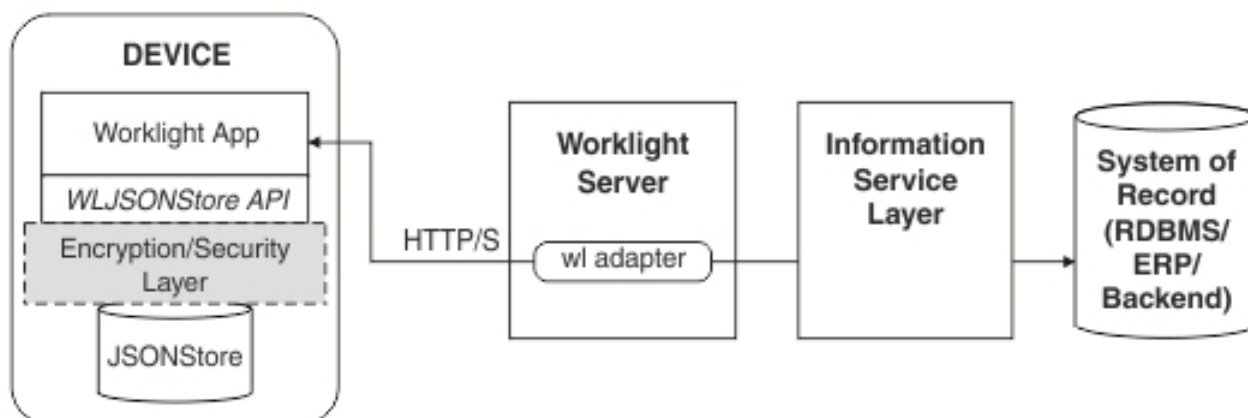


Figure 47. Components and their interaction with the server when using JSONStore for data synchronization.

**Note:** Because it is familiar to many developers, relational database terminology is used in this documentation at times to help explain JSONStore. There are many differences between a relational database and JSONStore however. For example, the strict schema used to store data in relational databases is different from JSONStore's approach, in which you can store any JSON content, and index the content that you need to search.

## JSONStore features comparison

Compare JSONStore features to those of other data storage technologies and formats.

JSONStore is a JavaScript API for storing data inside hybrid Worklight applications. It is similar to technologies such as LocalStorage, Indexed DB , Cordova Storage API, Cordova File API, and IBM Worklight Encrypted Cache. The



table shows how some features provided by JSONStore compare with other technologies. The JSONStore feature is only available on iOS and Android devices and simulators.

Table 64. A comparison of data storage technologies.

	JSONStore	Encrypted Cache	LocalStorage	IndexedDB	Cordova Storage	Cordova File
Android Support	⊆	⊆	⊆	⊆	⊆	⊆
iOS Support	⊆	⊆	⊆	⊆	⊆	⊆
Web	<i>Dev Only</i> (See Note 1)	⊆	⊆	⊆	-	-
Data encryption	⊆	⊆	-	-	-	-
Maximum Storage	Available Space	~5 MB	~5 MB	>5 MB	Available Space	Available Space
Reliable Storage (See Note 2)	⊆	-	-	-	⊆	⊆
Adapter integration	⊆	-	-	-	-	-
Multi-user support	⊆	-	-	-	-	-
Indexing	⊆	-	-	⊆	⊆	-
Type of Storage	JSON Documents	Key/Value Pairs	Key/Value Pairs	JSON Documents	Relational (SQL)	Strings

**Note:** 1. *Dev Only* means designed only for development, with no security features and a ~5 MB storage space limit.

**Note:** 2. *Reliable Storage* means that your data is not deleted unless one of the following events occurs:

- The application is removed from the device.
- One of the methods that removes data is called.

## Enabling JSONStore

To use JSONStore in IBM Worklight v6.0 you must take steps to enable it.

### About this task

JSONStore is an optional feature in IBM Worklight v6.0. To use JSONStore you must enable it by modifying the `application-descriptor.xml` file.

### Procedure

1. Using the Application Descriptor Editor, open the file `application-descriptor.xml`
2. Click the **Design** tab.
3. Under **Overview**, expand Application [your application's name].

4. Click **Optional Features**.
5. Click **Add**.
6. Select JSONStore.
7. Click **Ok**.
8. Under **Worklight Project**, right click the folder titled with your application name.
9. Select **Run As...**
10. Click **Build All and Deploy**.

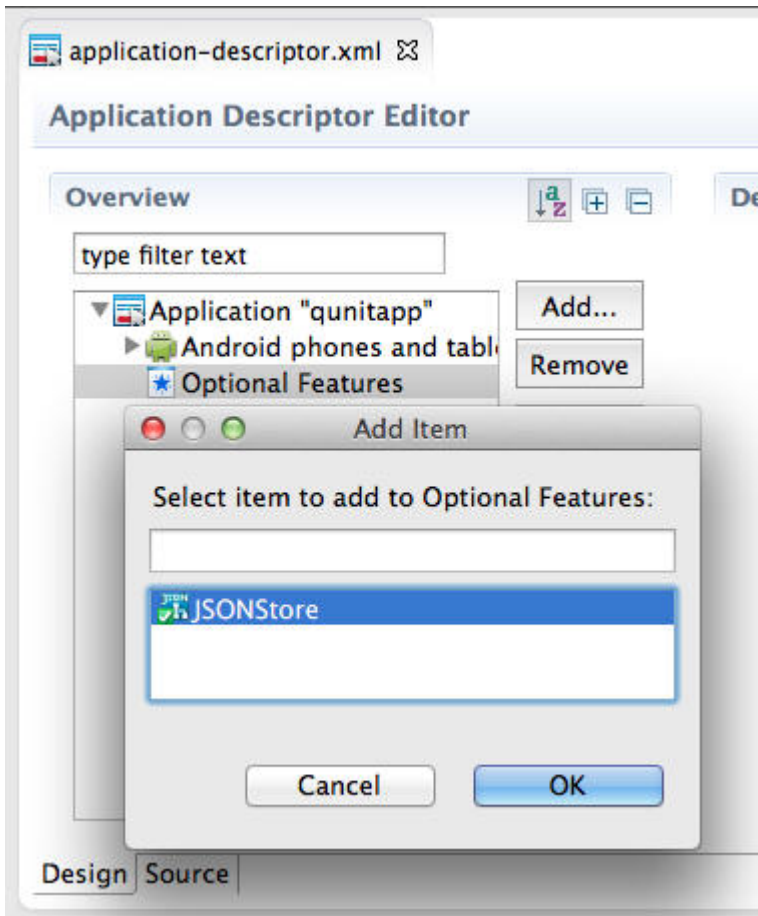


Figure 48. Enabling JSONStore

## JSONStore document

A document is the basic building block of JSONStore.

A JSONStore document is JavaScript object with an automatically generated identifier (`_id`) and JSON data. It is similar to a record or row in database terminology. The value of `_id` is always a unique integer inside a specific collection. Some functions like `WL.JSONStore.add` on page 531 `WL.JSONStore.replace` on page 546 and `WL.JSONStore.remove` on page 545 take an Array of Documents/Objects. This is useful to perform an operation on various Documents/Objects at a time.

## Example

Single document

```
var doc = { _id: 1, json: {name: 'carlos', age: 99} };
```

## Example

Array of documents

```
var docs = [  
  { _id: 1, json: {name: 'carlos', age: 99} },  
  { _id: 2, json: {name: 'tim', age: 100} }  
]
```

## JSONStore collection

A named logical grouping of related documents.

A JSONStore collection is similar to a table, in database terminology

## Example

Customer collection

```
[  
  { _id: 1, json: {name: 'carlos', age: 99} },  
  { _id: 2, json: {name: 'tim', age: 100} }  
]
```

This code is not the way that the documents are actually stored on disk, but it is a good way to visualize what a collection looks like at a high level.

## JSONStore store

A store is the persistent JSONStore file that contains one or more collections.

A store is similar to a relational database, in database terminology. This is also referred to as a JSONStore.

## JSONStore search fields

A search field is a JavaScript object with a key and a data type.

Search fields are keys that are indexed for fast lookup times, similar to column fields or attributes, in database terminology.

Additional search fields are keys that are indexed but that are not part of the JSON data that is stored. These fields define the key whose values (in a given JSON collection) are indexed and can be used to search more quickly.

Valid data types are: string, boolean, number, and integer. These are only type hints, there is no type validation. Furthermore, these types determine how indexable fields are stored. For example, {age: 'number'} will index 1 as 1.0 and {age: 'integer'} will index 1 as 1.

## Examples

Search fields and additional search fields.

```
var searchField = {name: 'string', age: 'integer'};  
var additionalSearchField = {key: 'string'};
```

It is only possible to index keys inside a JavaScript object, not the object itself. 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.

Indexing values inside an array.

```
var searchFields = {
  'people.name' : 'string', //matches carlos and tim on myObject
  'people.age' : 'integer' // matches 99 and 100 on myObject
};

var myObject = {
  people : [
    {name: 'carlos', age: 99},
    {name: 'tim', age: 100}
  ]
};
```

## JSONStore queries

Queries are JavaScript objects that use search fields or additional search fields to look for documents.

The example presumes that name and age are search fields, and that the types are string and integer, respectively.

### Examples

Find documents with name that matches carlos:

```
var query1 = {name: 'carlos'};
```

Find documents with name that matches carlos and age matches 99:

```
var query2 = {name: 'carlos', age: 99};
```

## Store internals

See an example of how JSONStore data is stored.

The key elements in this simplified example:

- `_id` is the unique identifier (for example, AUTO INCREMENT PRIMARY KEY).
- `json` contains an exact representation of the JSON object stored.
- `name` and `age` are search fields.
- `key` is an additional search field.

### Example

Table 65. Contents of a store in JSONStore

<code>_id</code>	<code>key</code>	<code>name</code>	<code>age</code>	<code>JSON</code>
1	c	carlos	99	{name: 'carlos', age: 99}
2	t	time	100	{name: 'tim', age: 100}

When you search using one of the following queries or a combination of them: `{_id : 1}`, `{name: 'carlos'}`, `{age: 99}`, `{key: 'c'}`, the returned document is `{_id: 1, json: {name: 'carlos', age: 99} }`.

## JSONStore asynchronicity, callbacks, and promises

Most of the operations that can be performed on a collection, such as add and find, are asynchronous.

Most of the operations that can be performed on a collection, such as add and find, are asynchronous. These asynchronous operations return a jQuery promise that is resolved when completed successfully and rejected when a failure occurs. These are similar to success and failure callbacks.

A jQuery Deferred is a promise that can be resolved or rejected. The two examples below are not specific to JSONStore, but are intended to help readers understand the usage of then that is used in the JSONStore examples.

### Example

Using a promise instead of a callback

```
function asyncOperation = function () {
    var deferred = $.Deferred();
    setTimeout(function () {
        deferred.resolve('Hello');
    }, 1000);
    return deferred.promise();
}
//The function passed to .then is executed after 1000ms
asyncOperation.then(function (response) {
    //response = 'Hello'
});
```

Using a callback instead of a promise

```
function asyncOperation = function (callback) {
    setTimeout(function () {
        callback('Hello');
    }, 1000);
}
asyncOperation(function (response) {
    //response = 'Hello'
});
```

Callbacks have been deprecated in the JSONStore API, but they are currently still supported for backwards compatibility. It is possible but not a good practice to pass an options object to a method like “WL.JSONStore.add” on page 531, “WL.JSONStore.find” on page 536, “WL.JSONStore.replace” on page 546 and “WL.JSONStore.remove” on page 545 with the failure and success callbacks. The format is an onSuccess or onFailure key with a function as the value that gets called when the operation succeeds or fails respectively.

## Chain JSONStore functions and concurrency

By using jQuery.when you can chain JSONStore functions and concurrency.

Every JSONStore API function that returns a promise (for example:

“WL.JSONStore.find” on page 536 and “WL.JSONStore.remove” on page 545) can take advantage of jQuery.when, which calls a function after all of the promises that were passed are resolved. It is also possible to chain various JSONStore functions by using then, passing either only a success callback or both a success callback and a failure callback.

## Example

Pseudocode

```
$.when(collection.find(...), collection.remove(...))

.then(function () {
    //Called when find and remove finished
    return collection.push(...); //push returns a promise
})

.then(function () {
    //Called when push finished
})

.fail(function () {
    //Called when find, remove or push fail.
});
```

Because promises are "aware" of whether they have been resolved or rejected, they can propagate errors, not calling any callback until an error handler is encountered. In the example the error callback function is at the end of the chain.

## JSONStore events

You can listen to the following events and capture successful and failure calls from the JSONStore API.

### Example

```
'WL/JSONSTORE/SUCCESS'
'WL/JSONSTORE/FAILURE'
```

The following example assumes jQuery >1.7 or using WLJQ (jQuery shipped with IBM Worklight).

```
$(document.body).on('WL/JSONSTORE/SUCCESS', function (evt, data, src, collectionName) {

    //evt - Contains information about the event
    //data - Data sent after the operation (e.g. add, find, etc.) finished
    //src - Name of the operation (e.g. find, push)
    //collectionName - Name of the collection
});
```

## JSONStore errors

JSONStore uses an error object to return messages about the cause of failures

### Example

When an error occurs during a JSONStore operation (for example "WL.JSONStore.find" on page 536 and "WL.JSONStore.add" on page 531) an error object is returned. It provides information about the cause of the failure. Possible JSONStore error codes that are returned are listed in "JSONStore error codes" on page 401.

Error object

```
var errorObject = {
    src: 'find', //operation that failed
    err: -50, //error code
    msg: 'PERSISTENT_STORE_FAILURE', //error message
    col: 'people', //collection name
```

```

    usr: 'jsonstore', //username
    doc: {_id: 1, {name: 'carlos', age: 99}}, //document related to the failure
    res: {...} //response from the server
}

```

Not all the key/value pairs are part of every error object. For example, the response from the server is only available when operations that use the network (for example “WL.JSONStore.push” on page 543) fail.

## JSONStore error codes

Definitions of the error codes related to JSONStore.

### -100 UNKNOWN\_FAILURE

Unrecognized error when building the error object.

### -50 PERSISTENT\_STORE\_NOT\_OPEN

JSONStore is closed. Try calling “WL.JSONStore.init” on page 540 first to enable access to the store.

### -40 FIPS\_ENABLEMENT\_FAILURE

Something is wrong with FIPS, try following the Getting Started module *Adapter framework overview*, under category 4, *Worklight server-side development*, in Chapter 3, “Tutorials and samples,” on page 25.

### -12 INVALID\_SEARCH\_FIELD\_TYPES

Check that the types that you are passing to the searchFields are **stringinteger**, **number**, or **orboolean**.

### -11 OPERATION\_FAILED\_ON\_SPECIFIC\_DOCUMENT

An operation on an array of documents, for example “WL.JSONStore.replace” on page 546, can fail while working with a specific document. The document that failed is returned and the transaction is rolled back.

### -10 ACCEPT\_CONDITION\_FAILED

The accept function that the user provided returned “false”.

### -9 OFFSET\_WITHOUT\_LIMIT

To use offset, you must also specify a limit.

### -8 INVALID\_LIMIT\_OR\_OFFSET

Validation error, must be a positive integer.

### -7 INVALID\_USERNAME

Validation error (Must be [A-Z] or [a-z] or [0-9] only).

### -6 USERNAME\_MISMATCH\_DETECTED

To log out, a JSONStore user must call “WL.JSONStore.closeAll” on page 533 first. There can be only one user at a time.

### -5 DESTROY\_REMOVE\_PERSISTENT\_STORE\_FAILED

A problem with “WL.JSONStore.destroy” on page 534 while trying to delete the file that holds the contents of the store.

### -4 DESTROY\_REMOVE\_KEYS\_FAILED

Problem with “WL.JSONStore.destroy” on page 534 while trying to clear the keychain (iOS) or shared user preferences (Android).

### -3 INVALID\_KEY\_ON\_PROVISION

Passed the wrong password to an encrypted store.

### -2 PROVISION\_TABLE\_SEARCH\_FIELDS\_MISMATCH

Search fields are not dynamic. It is not possible to change search fields without calling “WL.JSONStore.destroy” on page 534 or



“WL.JSONStore.removeCollection” on page 546 before calling “WL.JSONStore.init” on page 540 with the new search fields. This error can occur if you change the name or type of the search field. For example: {key: 'string'} to {key: 'number'} or {myKey: 'string'} to {theKey: 'string'}.

**-1 PERSISTENT\_STORE\_FAILURE**

Generic Error. A malfunction in native code, most likely calling “WL.JSONStore.init” on page 540.

**0 SUCCESS**

In some cases, JSONStore native code returns 0 to indicate success.

**1 BAD\_PARAMETER\_EXPECTED\_INT**

Validation error.

**2 BAD\_PARAMETER\_EXPECTED\_STRING**

Validation error.

**3 BAD\_PARAMETER\_EXPECTED\_FUNCTION**

Validation error.

**4 BAD\_PARAMETER\_EXPECTED\_ALPHANUMERIC\_STRING**

Validation error.

**5 BAD\_PARAMETER\_EXPECTED\_OBJECT**

Validation error.

**6 BAD\_PARAMETER\_EXPECTED\_SIMPLE\_OBJECT**

Validation error.

**7 BAD\_PARAMETER\_EXPECTED\_DOCUMENT**

Validation error.

**8 FAILED\_TO\_GET\_UNPUSHED\_DOCUMENTS\_FROM\_DB**

The query that selects all documents that are marked dirty failed. An example in SQL of the query would be: **SELECT \* FROM [collection] WHERE \_dirty > 0.**

**9 NO\_ADAPTER\_LINKED\_TO\_COLLECTION**

To use functions like “WL.JSONStore.push” on page 543 and “WL.JSONStore.load” on page 543, an adapter must be passed to “WL.JSONStore.init” on page 540

**10 BAD\_PARAMETER\_EXPECTED\_DOCUMENT\_OR\_ARRAY\_OF\_DOCUMENTS**

Validation error

**11**

**INVALID\_PASSWORD\_EXPECTED\_ALPHANUMERIC\_STRING\_WITH\_LENGTH\_GREATER\_THAN\_ZERO**

Validation error

**12 ADAPTER\_FAILURE**

Problem calling WL.Client.invokeProcedure, specifically a problem in connecting to the Worklight server adapter. This error is different from a failure in the adapter trying to call a backend.

**13 BAD\_PARAMETER\_EXPECTED\_DOCUMENT\_OR\_ID**

Validation error

**14 CAN\_NOT\_REPLACE\_DEFAULT\_FUNCTIONS**

Calling “WL.JSONStore.enhance” on page 535 to replace an existing function (“WL.JSONStore.find” on page 536, “WL.JSONStore.add” on page 531) is not allowed

#### 15 **COULD\_NOT\_MARK\_DOCUMENT\_PUSHED**

Push sends the document to an adapter but JSONStore fails to mark the document as not dirty.

#### 16 **COULD\_NOT\_GET\_SECURE\_KEY**

To initiate a collection with a password there must be connectivity to the Worklight Server because it returns a 'secure random token'. Worklight 5.0.6 and later allows developers to generate the secure random token locally passing {localKeyGen: true} to "WL.JSONStore.init" on page 540 via the options object.

#### 17 **FAILED\_TO\_LOAD\_INITIAL\_DATA\_FROM\_ADAPTER**

Could not load data because WL.Client.invokeProcedure called the failure callback.

#### 18 **FAILED\_TO\_LOAD\_INITIAL\_DATA\_FROM\_ADAPTER\_INVALID\_LOAD\_OBJ**

The load object that was passed to "WL.JSONStore.init" on page 540 did not pass the validation.

#### 19 **INVALID\_KEY\_IN\_LOAD\_OBJECT**

There is a problem with the key used in the load object when calling "WL.JSONStore.load" on page 543.

#### 20 **UNDEFINED\_PUSH\_OPERATION**

There is no procedure defined for pushing dirty documents to the server. For example: "WL.JSONStore.add" on page 531 (new document is dirty, operation = 'add') and "WL.JSONStore.push" on page 543 (finds the new document with operation = 'add') were called, but no add key with the add procedure was found in the adapter that is linked to the collection. Linking an adapter is done inside the "WL.JSONStore.init" on page 540 function.

#### 21 **INVALID\_ADD\_INDEX\_KEY**

Problem with additional search fields.

#### 22 **INVALID\_SEARCH\_FIELD**

One of your search fields is invalid. Verify none of the search fields passed are \_id,json,\_deleted, or \_operation.

#### 23 **ERROR\_CLOSING\_ALL**

Generic Error. An error occurred when native code called "WL.JSONStore.closeAll" on page 533.

#### 24 **ERROR\_CHANGING\_PASSWORD**

Unable to change the password. The old password passed was wrong, for example.

#### 25 **ERROR\_DURING\_DESTROY**

Generic Error. An error occurred when native code called "WL.JSONStore.destroy" on page 534.

#### 26 **ERROR\_CLEARING\_COLLECTION**

Generic Error. An error occurred in when native code called "WL.JSONStore.removeCollection" on page 546.

#### 27 **INVALID\_PARAMETER\_FOR\_FIND\_BY\_ID**

Validation error.

## JSONStore support

How to get support for JSONStore

- Read the JSONStore documentation and "WL.JSONStore API" on page 531 sections.

- Read the Chapter 3, “Tutorials and samples,” on page 25 and try the sample code.

## Provide information

It is better to provide more information than necessary than to risk providing too little information. This list is a good starting point for the information that is required to help with JSONStore issues.

- Operating System and Version (for example Windows XP SP3 Virtual Machine, Mac OSX 10.8.3)
- Eclipse Version (for example: Eclipse Indigo 3.7 Java EE)
- JDK Version (for example: Java(TM) SE Runtime Environment (build 1.7))
- Worklight Version (for example: Worklight 5.0.6 Developer Edition)
- iOS Version (for example: iOS Simulator 6.1, iPhone 4S iOS 6.0)
- Android Version (for example: Android Emulator 4.1.1, Samsung Galaxy Android 4.0 API Level 14)
- adb Version (for example: Android Debug Bridge version 1.0.31)

## Try to isolate the issue

Follow these steps to isolate the issue to more accurately report a problem.

- Reset the Simulator or Emulator and (or) call “WL.JSONStore.destroy” on page 534 to start with a clean system.
- Make sure you are running on a supported production environment:
  - Android >= 2.3 ARM/x86 Emulator or Devices
  - iOS >= 5.0 Simulator or Device
- Try turning encryption off (do not pass a password to “WL.JSONStore.init” on page 540).
- Look at the SQLite database file that is generated by JSONStore. This only works if encryption is off.

– Android:

```
$ adb shell
$ cd /data/data/com.[app-name]/databases/wljsonstore
$ sqlite3 jsonstore.sqlite
```

– iOS

```
$ cd ~/Library/Application Support/iPhone Simulator/6.1/Applications/[id]/Documents/wljsonstore
$ sqlite3 jsonstore.sqlite
```

Try looking at the searchFields with .schema and selecting data with SELECT \* FROM [collection-name];. To exit sqlite3 type .exit. If you are passing a username to “WL.JSONStore.init” on page 540, the file is called [username].sqlite, for example carlos.sqlite. Be aware that jsonstore is the default username (no username was passed).

- (Android Only) Enable verbose JSONStore.
 

```
adb shell setprop log.tag.jsonstore-core VERBOSE
adb shell getprop log.tag.jsonstore-core
```
- (iOS >=6.0 and Safari >=6.0 Only) Try to use the a JavaScript debugger. Set breakpoints inside jsonstore.js. Here are some helpful lines:
  - Bridge to Native code:
 

```
cdv.exec(options.onSuccess, options.onFailure, pluginName, nativeFunction, args);
```
  - Success Callbacks returning from Native code:

- ```
deferred.resolve(data, more);
```
- Failure Callbacks returning from Native code:

```
deferred.reject(new ErrorObject(errorObject));
```

## Common issues

Understanding the following JSONStore characteristics can help in resolving some of the common issues that you might encounter.

- The only way to store binary data in JSONStore is to first encode it in base64.
- Accessing JSONStore data from native code is not possible, JSONStore is only available in hybrid environments.
- There is no limit on how much data you can store inside JSONStore, beyond limits that are imposed by the mobile operating system. It is possible to use the Cordova File API to check the size of the store on disk before allowing new data to be added.
- JSONStore provides persistent data storage. It is not only stored in memory.
- The init function fails when the collection name starts with a digit or symbol. Worklight >5.0.6.1 returns a proper error (4 BAD\_PARAMETER\_EXPECTED\_ALPHANUMERIC\_STRING)
- There is a difference between number and integer in search fields. Numeric values like 1 and 2 are stored as 1.0 and 2.0 when the type is number, and are stored as 1 and 2 when the type is integer.

## JSONStore performance

Learn about the factors that can affect JSONStore performance.

### Network

- IBM Worklight provides an API for Android and iOS to give developers information about the network. It can be accessed by calling `WL.Device.getNetworkInfo`. Ideally getting and sending data from and to an IBM Worklight adapter should be done when `networkConnectionType` returns `WIFI`.
- Since IBM Worklight ships with Apache Cordova, it is possible to consult events such as `online` and `offline` to determine when the application has network connectivity. Using these events is more efficient than polling the network to check for connectivity.
- The amount of data that is sent over the network to a client heavily affects performance. Even if the IBM Worklight Server gets all the data from the backend, clients only get a minimal subset of the data that they need.

### Memory

- JSONStore documents are serialized and deserialized as Strings between the Native (Objective-C or Java) Layer and the JavaScript Layer. One way to mitigate possible memory issues is by using `limit` and `offset` when calling `WL.JSONStore.find` on page 536. Instead of trying to get all the documents from a local storage, only get a subset and implement pagination. Using `{exact: true}` to avoid fuzzy searching and setting a limit can also help improve `WL.JSONStore.find` on page 536 performance.
- Instead of using long key names that are eventually serialized and deserialized as Strings, consider mapping those long key names into smaller ones (for example: `myVeryVeryVerLongKeyName` to `k` or `key`). Ideally the developer maps

them to short key names when sending them from the adapter to the client, and maps them to the original long key names when sending data back to the backend.

- Consider splitting the data stored inside JSONStore into various collections, thus having small documents over various collections instead of monolithic documents in a single collection. This depends on how closely related the data is and the use cases for said data.
- When calling “WL.JSONStore.add” on page 531 with an array of objects or using “WL.JSONStore.load” on page 543 to store an array of objects from an adapter, it is possible to run into memory issues. To mitigate this issue, call these methods with fewer documents at a time. For example, pass an array of 10 documents to “WL.JSONStore.add” on page 531 instead of an array with 1000 documents.
- JavaScript engines have a garbage collector. Allow it to work, but don't depend on it entirely. Try to null references that are no longer used and use profiling tools to check that memory usage is going down when you expected to go down.

## CPU

- The amount of search fields and additional search fields that are passed to “WL.JSONStore.init” on page 540 affects performance when calling “WL.JSONStore.add” on page 531, which does the indexing. Only index the values that is used in queries for “WL.JSONStore.find” on page 536.
- Enabling security adds some overhead to init and other operations that work with documents inside the collection. Consider whether security is genuinely required.
- JSONStore by default keeps track of local changes to its documents. This behavior can be disabled, thus saving a few cycles, by passing the {push: false} flag to “WL.JSONStore.add” on page 531, “WL.JSONStore.remove” on page 545 and “WL.JSONStore.replace” on page 546.

## Miscellaneous

- Most of the core JSONStore API is asynchronous. Ideally, only block sections of the UI that depend on data from JSONStore, instead of blocking the whole User Interface with something like a WL.BusyIndicator. One approach is to append text (that is, 'Loading...') to a DOM element and then replace that text with real data from a JSONStore collection when, for example, “WL.JSONStore.find” on page 536 has finished.

## JSONStore multiple user support

With JSONStore, you can create multiple stores that contain different collections in a single IBM Worklight application.

The “WL.JSONStore.init” on page 540 function can take an options object with a user name. Separate stores are separate files in the file system. These separate stores can be encrypted with different passwords for security and privacy reasons. Calling the “WL.JSONStore.closeAll” on page 533 function removes access to all the collections. It is also possible to change the password of an encrypted store calling “WL.JSONStore.changePassword” on page 532.

An example use case would be various employees sharing the same physical device (for example an iPad or Android tablet) and IBM Worklight application. In addition, if the employees work different shifts and handle private data from different customers while using the IBM Worklight application, multiple user support is particularly useful.

## JSONStore security

You can secure all of the collections in a store by encrypting them.

To encrypt all of the collections in a store, pass a password to the “WL.JSONStore.init” on page 540 function. If no password is passed, none of the documents in the store collections are encrypted.

Some metadata / security artifacts (for example salt) are stored in the Keychain (iOS) or Shared Preferences (Android). The store is encrypted with a 256-bit Advanced Encryption Standard (AES) key. All keys are strengthened with Password-Based Key Derivation Function 2 (PBKDF2).

Data encryption is only available on Android and iOS environments. You can choose to encrypt collections for an application, but it is not possible to switch between encrypted and plain-text formats, or to mix formats within a store.

The key that protects the data in the store is based on the user password that you provide. The key does not expire, but you can change it by calling “WL.JSONStore.changePassword” on page 532.

## Worklight adapter integration for JSONStore

You can enable JSONStore data synchronization by linking a collection to an IBM Worklight Adapter.

This topic assumes that the reader is familiar with IBM Worklight adapters. For more information, see *Adapter framework overview*, under category 4, *Worklight server-side development*, in Chapter 3, “Tutorials and samples,” on page 25.

### Overview

Writing an application that maintains a local copy of its data and, on request, pushes the local updates to a back-end service can be achieved with JSONStore. The local copy is a store that holds JSON documents managed by the JSONStore API on the device.

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. Functions like “WL.JSONStore.add” on page 531, “WL.JSONStore.replace” on page 546, “WL.JSONStore.remove” on page 545, and most other operations are specific to the local copy of the data. JSONStore tracks modifications to data made locally. The exceptions are “WL.JSONStore.push” on page 543 and “WL.JSONStore.load” on page 543, which act on the local and remote data.

Linking a collection to an adapter allows JSONStore to:

- Send data from a collection to an IBM Worklight Adapter.
- Get data from an IBM Worklight Adapter into a collection.

**Note:** Those two goals can also be achieved using functions like “WL.Client.invokeProcedure” on page 513 and `jQuery.ajax` to transmit and receive data, and `getPushRequired` to get the changes.



## Example Use Case

- A mobile worker, whose job includes customer visits, downloads a large set of information to a mobile device when network conditions permit this. This can be for example while 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.
- JSONStore tracks any changes that are made to the local copy of the data.
- At an appropriate time, for example when the worker is 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 an IBM Worklight adapter that pushes the updates into a back-end system.

## Adapter integration workflow

The first task is to create an IBM Worklight Adapter, for example HTTP, SQL, JMS, and define the procedures to get, add, replace and remove data, for example: `getPeople`, `addPerson`, `replacePerson`, `removePerson`.

Next you must implement the procedures to get, add, replace, and remove data. Here is an example that just returns hardcoded data, and logs for simplicity. Read the IBM Worklight Adapter Documentation for details on how to contact the backend.

```
function getPeople () {
    return { peopleList : [{name: 'carlos', age: 100}, {name: 'tim', age: 99}] };
}

function addPerson (data) {
    return WL.Logger.debug('Got data from JSONStore to ADD: ' + data);
}

function replacePerson (data) {
    return WL.Logger.debug('Got data from JSONStore to REPLACE: ' + data);
}

function removePerson (data) {
    return WL.Logger.debug('Got data from JSONStore to REMOVE: ' + data);
}
```

To link a collection to an adapter you must specify the adapter option as part of the collection creation options when `init` is called. If an adapter is not specified for the collection, calls to `WL.JSONStore.push` on page 543 and `WL.JSONStore.load` on page 543 return an error.

When the `WL.JSONStore.load` on page 543 function is called, the adapter name, load procedure name, and the key are used to determine what to store. In the example, the key is `peopleList` because the goal is to store `{name: 'carlos', age: 100}` and `{name: 'tim', age: 99}` as two separate documents.

Calling `WL.JSONStore.getPushRequired` on page 539 will return an array of documents that have local only modifications, these are the documents will be sent to the IBM Worklight Adapter when push is called.

The push function sends the document that changed to the correct IBM Worklight Adapter procedure. This is based on the last operation associated with the document that changed (for example `addPerson` will be called with a document that was added locally).



The option `{push: false}` can be passed to `“WL.JSONStore.add”` on page 531, `“WL.JSONStore.replace”` on page 546, and `“WL.JSONStore.remove”` on page 545 to stop JSONStore from marking the documents as dirty (for example tracking local changes). Dirty means that the document has local modifications that do not exist on the backend. It's possible to check if a document is dirty by calling the `“WL.JSONStore.isPushRequired”` on page 542 function. Checking how many documents are dirty can be achieved with `“WL.JSONStore.pushRequiredCount”` on page 544 or counting the documents returned by `“WL.JSONStore.getPushRequired”` on page 539.

## IBM Worklight adapter wizard

The IBM Worklight wizard can help you create a template JavaScript file that is based on search fields selected from the backend that you provide.

### About this task

Before using the wizard, see `“Worklight adapter integration for JSONStore”` on page 407 for an overview. Using the wizard is optional. You can link a collection to an adapter by manually writing the procedure names when `“WL.JSONStore.init”` on page 540 is called.

### 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. Enter the appropriate information into the fields in this window and click **Finish**. A standard application structure is created.
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 opens.
  - 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 JSONstore:
  - a. In Worklight Studio, click **File > New > Worklight JSONStore** and select the project and app names. The Create JSON Collection wizard starts.
  - b. Follow the instructions in the wizard to start 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_nameCollection.js` in the application's `common/js` directory, where `collection_name` is the name you specified in the wizard.

5. Review the `collection_nameCollection.js` file and include its content manually in your application's `.js` file.

**Note:** The input data for the JSONStore wizard must be encoded with UTF-8. Other data encoding is not supported.

## Troubleshooting JSONStore and data synchronization

Find information to help resolve issues with JSONStore and data synchronization.

To get help in troubleshooting JSONStore and data synchronization problems:

- Read about common issues.
- Read a list of error codes.

### About stored data synchronization errors

The “`WL.JSONStore.push`” on page 543 and “`WL.JSONStore.pushSelected`” on page 551 methods operate similarly, with several differences. “`WL.JSONStore.push`” on page 543 takes no parameters, and synchronizes all the documents that were modified locally. “`WL.JSONStore.pushSelected`” on page 551 takes the ID of a specific document or documents to synchronize.

These methods can fail if they are supplied with the wrong data, or if something goes wrong while native code is running to get the documents, or verifying that they are unsynchronized.

After the documents are retrieved, an IBM Worklight adapter is called by using the appropriate option (“`WL.JSONStore.add`” on page 531, “`WL.JSONStore.replace`” on page 546, or “`WL.JSONStore.remove`” on page 545). The adapter then tries to contact the back-end server. This attempt can fail if the adapter or procedure name does not exist or if the IBM Worklight Server or back-end server cannot be reached.

If the server is contacted, you can check the status code from the synchronization procedure, and then 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 in “JSONStore error codes” on page 401.

---

## Push notification

The IBM Worklight unified push notification mechanism enables the sending of mobile notifications to mobile phones.

Notifications are sent through the vendor infrastructure. For example, iPhone notifications are sent from the Worklight Server to specialized Apple servers, and from there to the relevant phones.

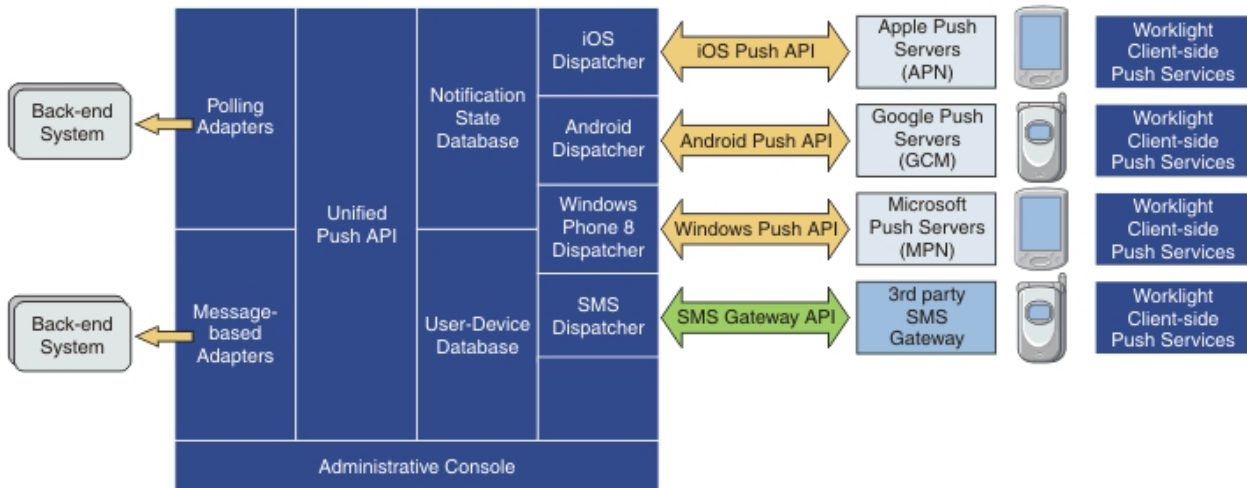


Figure 49. Push notification mechanism

**Note:** Push notification currently works for iOS, Android, and Windows Phone 8. iOS apps use the Apple Push Notification Service (APNS), Android apps use Google Cloud Messaging (GCM), and Windows Phone 8 apps use the authenticated and non-authenticated Microsoft Push Notification Service (MPNS). SMS push notifications are supported on iOS, Android, Windows Phone 7.5 (deprecated in IBM Worklight V6.0.0), Windows Phone 8, Java ME, 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 using the `push.apns.proxy.*` and `push.gcm.proxy.*` configuration properties. For further information, see “Configuration of IBM Worklight applications on the server” on page 714.

### Architecture

Unlike other IBM Worklight services, the push server requires outbound connections to Apple, Google, and Microsoft servers using ports defined by these companies.

When you are running a cluster of application servers, only one node actually sends push messages to Apple, Google, and Microsoft servers. This server is selected randomly.

For more information, see “Possible IBM Worklight push notification architectures.”

### Possible IBM Worklight push notification architectures

An explanation of two different methods of implementing push notifications, which are based on how the enterprise backend provides the messages to Worklight Server.

There are two common ways to create an IBM Worklight push notification architecture:

- JMS polling
- The enterprise backend calls a backend procedure

Each of these methods is explained in the following sections.

## JMS polling architecture

This architecture relies on the enterprise backend to deliver messages to a single instance of Worklight Server using a JMS message queue. The developer must create an IBM Worklight JMS adapter, which pulls messages from the queue and calls the IBM Worklight server-side push notification API to process the messages.

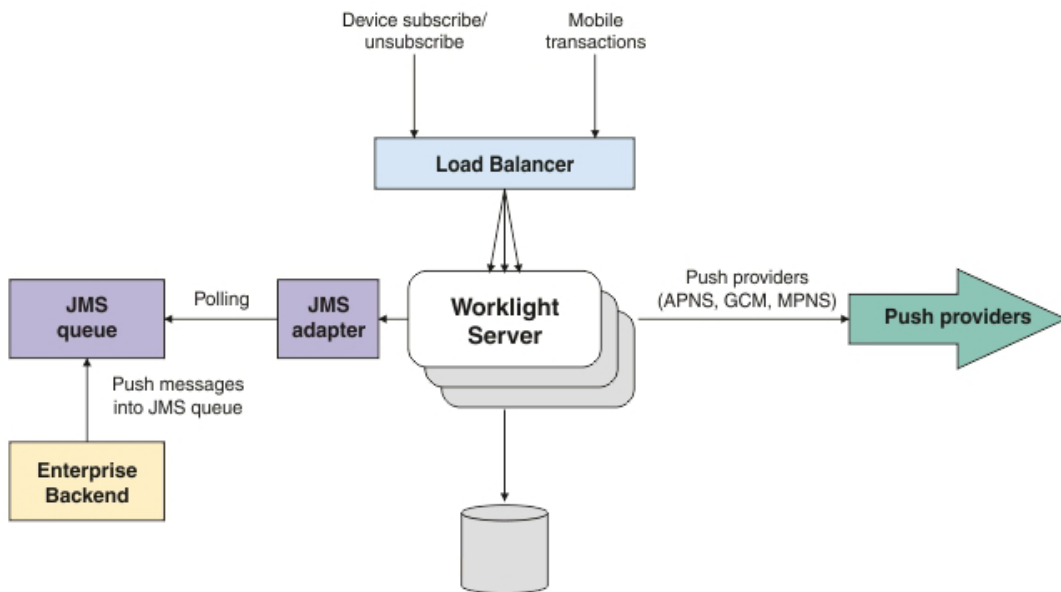


Figure 50. JMS polling push notification architecture

Using this architecture, the flow is as follows:

- Messages are put into the JMS queue by the enterprise backend.
- Worklight Server polls the JMS adapter, retrieving messages in short batches and sending them to the push providers.
- Only a single Worklight Server is pulling from the JMS queue and sending the push notifications.
- The process is implemented using a Worklight JMS adapter, which functions as follows:
  - The server is selected once randomly, using the IBM Worklight cluster-sync mechanism.
  - If the server that pulls from the JMS queue is shut down, another server takes its place.

This is the standard architecture. *Pros* of this method are that it involves an asynchronous queue, into which you can put the messages that you want to send. These messages are then processed and pulled later by the Worklight Server. *Cons* of this method are that only one server is sending the push notifications, so the maximum messages-per-second throughput is fixed.

## Enterprise backend calling the Worklight Server architecture

This architecture relies on the enterprise backend to deliver messages to a Worklight Server cluster by calling a Worklight adapter procedure.

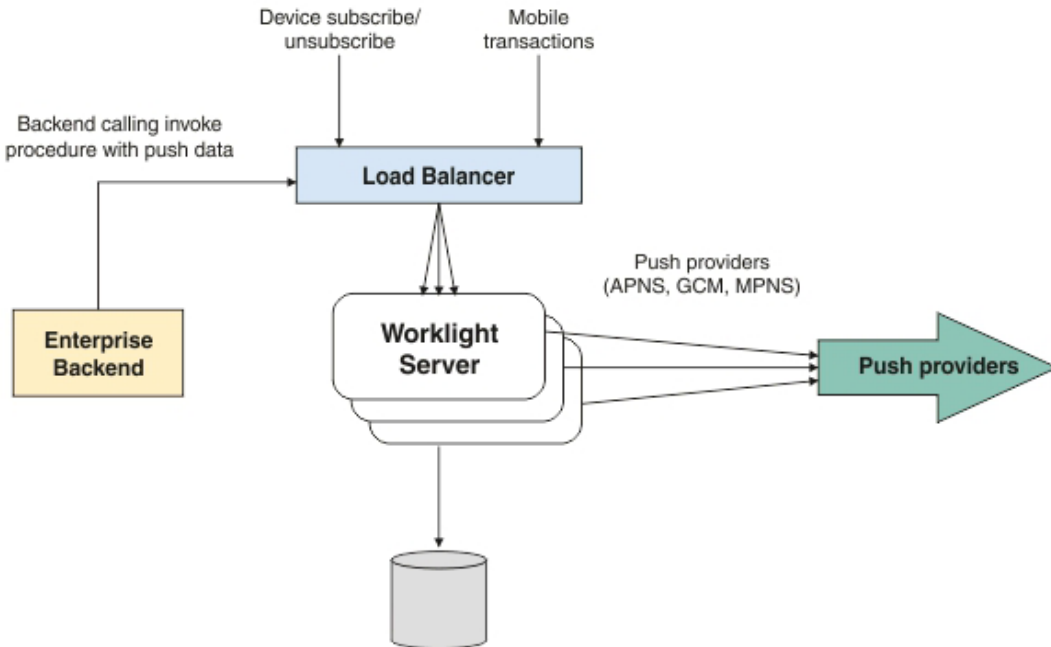


Figure 51. Enterprise backend push notification architecture

Using this architecture, the flow is as follows:

- The enterprise backend initiates calls to the load balancer.
- The request is routed to one of the Worklight Server instances, which sends a push message to a provider.
- In this flow all Worklight Server instances send push notifications.

*Pros* of this method are that all Worklight Servers can be used to send push notifications, so you can add more servers if you must send more messages per second. *Cons* of this method are that every push message is a transaction on the Worklight Server. You can mitigate this overhead by sending a number of messages together or by having the Worklight adapter procedure that is invoked call the backend for a batch of messages rather than single messages.

### 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 “WL.Server.createEventSource” on page 637 for information about creating an event source.

Table 66. Subscribe SMS servlet URL parameters

| URL parameter      | URL parameter description  |
|--------------------|--|
| <b>option</b>      | 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.   |
| <b>eventSource</b> | Mandatory string. The name of the event source. The event source name is in the format <i>AdapterName.EventSourceName</i> .  |
| <b>alias</b>       | Optional string. A short ID defining the event source during subscription. This ID is the same ID as provided in WL.Client.Push.subscribeSMS.  |
| <b>phoneNumber</b> | 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 (Δ) characters only.  |
| <b>userName</b>    | 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.   |
| <b>appId</b>       | 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 <b>appId</b> = 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:

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>
```

## Windows Phone 8 push notifications

You can set up a web service to provide authenticated web services on Windows Phone 8. Authenticated web services have no daily limit on the number of push notifications they can send, whereas unauthenticated web services are throttled at a rate of 500 push notifications per subscription per day.

### Authenticated push for Windows Phone 8

If you want to use an authenticated push notifications service, you must first set up a Secure Sockets Layer (SSL) certificate keystore. For more information, see “SSL certificate keystore setup” on page 720. The keystore can contain several certificates, one of which is the certificate for authenticated push notifications to MPNS.

You must also authenticate your web service with Microsoft, as documented in the Windows Phone Development Center at <http://dev.windowsphone.com/en-us/develop/>.



Next, in the application descriptor, for <windowsPhone8> set the following attributes for the <pushSender> element:

Attribute	Setting
<b>serviceName</b>	The common name (CN) found in the MPNS certificate's Subject value.
<b>keyAlias</b>	The alias used to access the keystore specified by the following properties in the <code>worklight.properties</code> file: <ul style="list-style-type: none"> <li>• <code>ssl.keystore.path</code></li> <li>• <code>ssl.keystore.type</code></li> <li>• <code>ssl.keystore.password</code></li> </ul>
<b>keyAliasPassword</b>	The password for your key alias.

The **serviceName** attribute from the application descriptor is passed to the application's client side, and is used when a new notification channel is created. The URI token of the notification channel starts with "https", rather than "http". The **keyAlias** and **keyAliasPassword** attributes are used by Worklight Server to extract the certificate from the Java keystore file, so that it can be used in the handshake process with Microsoft Push Notification Service (MPNS). Any push notifications submitted to the application will be authenticated and secure.

### Example

```
<windowsPhone8>
  <pushSender>
    <authenticatedPush serviceName="myservice"
      keyAlias="janedoe"
      keyAliasPassword="a1b2c3d4"></authenticatedPush>
  </pushSender>
  ...
</windowsPhone8>
```

### MPNS response codes

In response to push notification requests, MPNS returns a response code and a status. If the request is successful, the response code is 200, and the status is Received. For details of other response codes, go to the MSDN website at [msdn.microsoft.com](http://msdn.microsoft.com), and search for "push notification service response codes".

## Sending push notifications from WebSphere Application Server – IBM DB2

To issue push notifications from a WebSphere Application Server that uses IBM DB2 as its database, a custom property must be added.

### About this task

If you use WebSphere Application Server with an IBM DB2 database, errors can arise when you try to send push notifications. To resolve this situation, you must add a custom property in WebSphere Application Server, at the data source level.

### Procedure

1. Log in to the WebSphere Application Server admin console.
2. Select **Resources > JDBC > Data sources > DataSource name > Custom properties** and click **New**.

3. In the **Name** field, enter `allowNextOnExhaustedResultSet`.
4. In the **Value** field, type `1`.
5. Change the type to `java.lang.Integer`.
6. Click **OK** to save your changes.
7. Select custom property `resultSetHoldability`.
8. In the **Value** field, type `1`.
9. Click **OK** to save your changes.

---

## IBM Worklight security framework

This collection of topics contains information about and tasks for using IBM Worklight security framework in applications.

### IBM Worklight Security Overview

An overview of security features within IBM Worklight.

The following sections provide high-level information about the IBM Worklight security model.

#### Goals and structure of IBM Worklight security framework

The IBM Worklight security framework serves two main goals. It controls access to the protected resources, and it propagates the user (or server) identity to the backend systems through the adapter framework.

It is key to the success of the application that the Worklight security framework does not include its own user registry, credentials storage, or access control management. Instead, it delegates all those functions to the existing enterprise security infrastructure. This delegation allows Worklight Server to integrate smoothly as a presentation tier into the existing enterprise landscape. Integration with the existing security infrastructure is an important feature of the Worklight security framework, and supports custom extensions that allow integration with virtually any security mechanism.

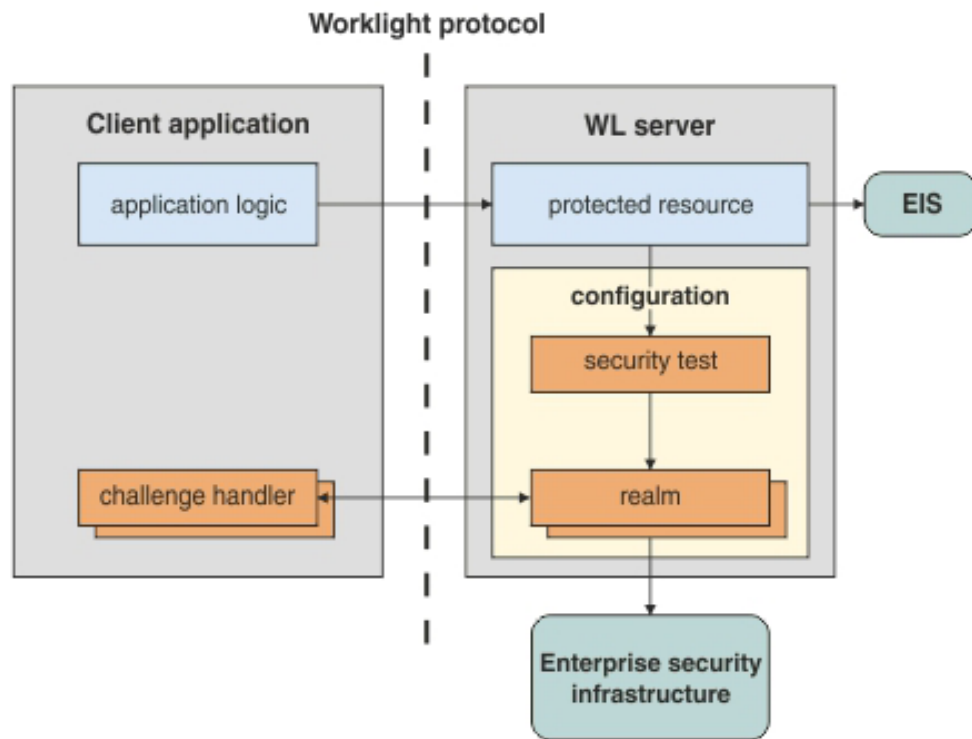
Another feature of the Worklight security framework is support of multi-factor authentication. It means that any protected resource can require multiple checks to control access. A typical example of multi-factor authentication is the combination of device, application, and user authentication.

Each type of security check has its own configuration, and a configured check is called a *realm*. Multiple realms can be grouped in a named entity that is called a *security test*. Each protected resource refers to the security test. All the configuration entities are defined in a single configuration file so that the definitions can be reused across different protected resources.

An implementation of security checks usually includes a client part and a server part. The two parts interact with each other according to their private protocol. This protocol is usually a sequence of 1) challenges that are sent by the server and 2) responses that are returned by the client.

The Worklight security framework provides a *wire protocol*. This protocol allows the combination of challenges and responses of multiple security checks during a single request-and-response round trip. The protocol serves two important

purposes: it allows the number of extra round trips between the client and server to be minimized, and it separates the application logic and the security checks implementation.



## Protected resources and authentication context

A protected resource can be any of the following items:

- **Application**  
Any request to the application requires successful authentication in all realms of the security test that is defined in the application descriptor.
- **Adapter procedure**  
Procedure invocation requires successful authentication in all realms of the security test that is defined in the adapter descriptor. The user identity and credentials that are obtained during such authentication can be propagated to the enterprise information system represented by this adapter.
- **Event source**  
Subscription to push notifications requires successful authentication in all realms of the security test that are defined in the event source definition (in adapter JavaScript).
- **Static resource**  
Static resources are defined as URL patterns in the authentication configuration file. They allow protection of "static" web applications such as the Worklight Console.

During the session, an application can access different resources. The results of the authentication in different realms are stored in the session authentication context. These results are then shared among all of the protected resources in the scope of the current session.

## Realms and security tests

A realm represents a fully configured security check that must be completed before it can allow access to a protected resource. The semantics of the checks are not limited to the authentication, but can implement any logic that can serve as protection for the server-side application resources, for example:

- User authentication
- Device authentication and provisioning
- Application authenticity check
- Remote disable of the ability to connect to Worklight Server
- Direct update
- Anti-XSRF check (cross-site request forgery)

The realms are defined in the authentication configuration file on the Worklight project level. A realm consists of two parts: the *authenticator* and the *login module*. The authenticator obtains the credentials from the client, and the login module validates those credentials, and builds the user identity.

The realms are grouped into security tests, which are defined in the same authentication configuration files. The security test defines not only the group of realms, but also the order in which they must be checked. For example, it often makes sense not to ask for the user credentials until you make sure the application itself is authentic.

Since some of realms are relevant only to mobile or only to web environments, the configuration of a security test can be non-trivial. Worklight provides simplified security test configurations for mobile and web environments. It is also possible to create a custom security test from scratch.

## Worklight protocol and client challenge handlers

Each security check defines its own protocol, which is a sequence of challenges that are sent by the server and responses that are sent by the client. On the server side, the component that implements this private protocol is the *authenticator*. On the client side, the corresponding component is called the *challenge handler*.

When the client request tries to access to a protected resource, Worklight Server checks all the appropriate realms. Several realms can decide to send a challenge. Challenges from multiple realms are composed into a single response and sent back to the client.

Worklight client infrastructure extracts the individual challenges from the response, and routes them to the appropriate challenge handlers. When a challenge handler finishes the processing, it submits its response to the Worklight client infrastructure. As an example, this occurs when it obtains the user name and password from a login user interface. When all the responses are received, the Worklight client infrastructure resends the original request with all the challenge responses.

Worklight Server extracts those responses from the request and passes them to the appropriate authenticators. If an authenticator is satisfied, it reports a success, and Worklight Server calls the login module. If the login module succeeds in validating

all of the credentials, the realm is considered successfully authenticated. If all the realms of the security test are successfully authenticated, Worklight Server allows the request processing to proceed.

If a realm check fails, its authenticator sends another (or the same) challenge to the client, and the whole process repeats.

Combining multiple challenges and responses into a single response and request minimizes security efficiency by reducing the number of extra round trips. For example, the checks for device authentication, application authenticity, and direct update can be done in a single round trip.

The fact the Worklight client infrastructure automatically resends the original request with the challenge responses allows separation between the application logic and security aspects. Though any application request can result in a security challenge, the application logic must not include any special processing for that case. The challenge handlers are not related to the application context and can focus on the security-related logic.

### **Integration with container security**

Worklight Server is technically a web application hosted by an application server (such as WebSphere Application Server). Thus, it is often desirable to reuse authentication capabilities of the application server for Worklight Server, and vice versa. This task can be non-trivial, and one must understand the differences between Worklight and Web Container authentication models.

The Java Platform, Enterprise Edition model allows only one authentication scheme for a web application, with multiple resource collections that are defined by URL patterns with authentication constraints defined by a white list of role names.

The Worklight model allows protection of each resource by multiple authentication checks, and the resources are not necessarily identified by the URL pattern. In some cases authentication can be triggered dynamically during the request processing.

As a result, the authentication integration between Worklight Server and the Java Platform, Enterprise Edition container is implemented as a custom Worklight realm. This realm can interact with the container and obtain and set its authenticated principal.

Worklight Server includes a set of login modules and authenticators for WebSphere Application Server Full Profile and WebSphere Application Server Liberty Profile that implement this integration with LTPA tokens. The integration works as follows:

- If the caller *principal* (an entity that can be authenticated) of the servlet request is already set, the container authentication was successful, and the same principal is set as the Worklight user identity. This case assumes that the Worklight WAR file has appropriate login configuration and resource collection definitions. Including this information can be tricky because the `web.xml` file for Worklight project is generated automatically, and those definitions would be overwritten in every build.

- If the incoming request contains a Lightweight Third Party Authentication (LTPA) token, the login module validates it, and creates the Worklight user identity.
- If the request does not contain an LTPA token, the authenticator requests the user name and password from the client. The login module validates them and creates the Worklight user identity. In addition, it creates the LTPA token, and sends it back to the client as a cookie.

In this case, the authentication capabilities of WebSphere Application Server are reused by Worklight realms in the form of Java utilities that implement validation and building of an LTPA token.

## Integration with web gateways

Web gateways like DataPower and ISAM provide user authentication so that only authenticated requests can reach the internal applications. The internal applications can obtain the result of the authentication that is done by the gateway from a special header, for example, an LTPA token.

When Worklight Server is protected by a web gateway, this means that the client requests first encounter the gateway. The gateway sends back a login form and validates the credentials, and if the validation is successful, submits the request to the Worklight Server. This sequence implies the following requirements on the Worklight security elements:

- The client-side challenge handler must be able to present the gateway's login form, submit the credentials, and recognize the login failure and success.
- The authentication configuration must include the realm that can obtain and validate the token that is provided by the gateway
- The security test configuration must take into account that the user authentication is always done first. For example, there is no point in using the device SSO feature because the user credentials are requested by the gateway.

For more information on security as it is implemented in IBM Worklight, see the following overview of security features, with links to the relevant sections of the documentation pertaining to them.

## 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.



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, from the Worklight Console, to remotely disable the ability for the app to connect to Worklight Server.

#### **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.



**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 52. 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 53. 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. Both authenticators and login modules are components of the application's *realm*.

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 `authenticationConfig.xml` authentication configuration file, which is in the `/server/conf` directory of your IBM Worklight project. A web security test or mobile security test must contain a `<testUser>` element that specifies the realm name. The definition of a realm includes the class name of an authenticator, and a reference to a login module, and refers to a

collection of resource managers that honors a common set of user credentials and authorizations. Authenticators are the entities that authenticate clients. Authenticators collect client information, and then use login modules to verify this information.

Table 67. Predefined realms: properties of the <test realm> element.

Authenticator class name	Login module reference	Description
<b>wl_antiXSRFRealm</b>	<b>WLANtiXSRFLoginModule</b>	This realm is used to avoid cross-site request forgery attacks. When a new session is initiated, the first request to Worklight Server gets an HTTP 401 response that contains the WL-Instance-Id token. The Worklight framework extracts this token and uses it as a header on all subsequent requests. If this header is not present in these subsequent requests, HTTP 401 is returned again. This security mechanism makes sure that all subsequent requests are coming from the same source as the initial one.
<b>wl_deviceNoProvisioningRealm</b>	<b>WLDeviceNoProvisioningLoginModule</b>	The default <i>device identity</i> realm. Device identity is similar to user identity, but it is provided by the device itself. Device identity is relevant for hybrid and native smartphone environments only. The device identity is a must for functionality such as push notifications, and reports. This parameter means that the obtained device identity is used as is, without provisioning.
<b>wl_deviceAutoProvisioningRealm</b>	<b>WLDeviceAutoProvisioningLoginModule</b>	Description of this parameter is the same as for <b>wl_deviceNoProvisioningRealm</b> , but the obtained device identity is automatically provisioned by the Worklight Server. This realm must be used with <b>wl_authenticityRealm</b> .

wl_authenticityRealm	wl_authenticityLoginModule	This realm is used to verify that application is authentic and it was not modified by a third party. The authenticity check is based on certificates that are used to sign applications. This functionality is only available on customer and enterprise versions of IBM Worklight, and is supported by iOS and Android environments only.
----------------------	----------------------------	--

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.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.worklight.com/auth/config http://www.worklight.com/auth/config/worklight-authentication-config.xsd">
  <staticResources>
    <resource> ... </resource>
    <resource> ... </resource>
  </staticResources>
  <securityTests>
    <customSecurityTest> ... </customSecurityTest>
    <customSecurityTest> ... </customSecurityTest>
  </securityTests>
  <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 “Configuring the Application Center after installation” on page 118.

## Configuring authenticators and realms

Authenticators are defined within the realm that uses them.

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 68. The <realm> element attributes

Attribute	Description
name	Mandatory. The unique name by which the realm is referenced by the protected resources.
loginModule	Mandatory. The name of the login module that is used by the realm.

The <realm> element has the following subelements:

Table 69. 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 that are passed to the authenticator upon instantiation.  This element might be displayed 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. Basic authentication is an industry-standard method used to collect user name and password information.

**Note:** You can use the basic authenticator only for web applications, not for mobile applications.

### 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
<b>login-page</b>	<p>Path to a user-defined login page template, relative to the web application context under the <code>conf</code> directory. A sample <code>login.html</code> template file is provided under this directory when creating a Worklight project in Worklight Studio.</p> <p>The authenticator renders the login page template with the error messages. To display the error message, use the placeholder <code>\${errorMessage}</code> within your login page template, as depicted in the example.</p>
<b>auth-redirect</b>	Path to a user defined login page ( <code>html/jsp</code> ) relative to the web application context. Worklight redirects to this page when the user credentials are needed.

Both the **login-page** and **auth-redirect** parameter are optional, but if used, they cannot be used together. It is also possible to use neither of them; if no parameters are supplied, Worklight uses its default template.

```
<realm name="AppAuthRealm" loginModule="DatabaseLoginModule">
<className> com.worklight.core.auth.ext.FormBasedAuthenticator </className>
<parameter name="login-page" value="login.html" />
</realm>
```

If no parameters are specified, Worklight will use its default login page template.

## 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
<code>&lt;cookie-name&gt;</code>	Optional. The name of the persistent cookie. If this parameter is not specified, the default name, <code>WL_PERSISTENT_COOKIE</code> , 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

### 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 432.

## Class Name

com.worklight.integration.auth.AdapterAuthenticator

## Parameters

The adapter authenticator class has the following parameters:

Parameter	Description
<b>&lt;login-function&gt;</b>	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 <code>WL.Client.login</code> 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.
<b>&lt;logout-function&gt;</b>	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 <code>WL.Client.logout</code> 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.

### 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
<b>login-page</b>	Mandatory. The login page URL relative to the web application context.
<b>error-page</b>	Mandatory. The error page URL relative to the web application context.
<b>cookie-domain</b>	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.
<b>httponly-cookie</b>	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.
<b>cookie-name</b>	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>
```

## Configuring login modules

Login modules are defined in <loginModule> elements in the authenticationConfig.xml file.

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.

Attribute	Description
audit	<p>Optional. Defines whether login attempts that use the login module are logged in the audit log. The log file is <i>Worklight Project Name/server/log/audit/audit.log</i>.</p> <p>Valid values are:</p> <p><b>true</b> Login and logout attempts are logged in the audit log.</p> <p><b>false</b> Default. Login and logout attempts are not logged in the audit log.</p>

The <loginModule> element has the following subelements:

Element	Description
<className>	<p>Mandatory. The class name of the login module.</p> <p>For details of the supported login modules, see the following topics.</p>
<parameter>	<p>Optional. An initialization property of the login module. The supported properties and their semantics depend on the login module class.</p> <p>This element can occur multiple times.</p>

## 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.

### 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: <code>SELECT UserID, Password, DisplayName FROM UsersTable WHERE</code> where: <code>UserID, Password, and DisplayName</code> are the names of the columns that contain user login names, passwords, and display names <code>UsersTable</code> 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. Use this module only for test purposes.

### Class Name

`com.worklight.core.auth.ext.SingleIdentityLoginModule`

### Parameters

None

### Configuration

The `worklight.properties` file must contain the following properties:

Key	Description
<code>console.username</code>	Name of the user who can access the Console
<code>console.password</code>	Password of the user who can access the Console. The password can be encrypted as indicated in "Storing properties in encrypted format" on page 721.

```
<loginModule name="Console" canBeResourceLogin="false" isIdentityAssociationKey="false">
<className> com.worklight.core.auth.ext.SingleIdentityLoginModule </className>
</loginModule>
```

## 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.

## Class Name

com.worklight.core.auth.ext.LdapLoginModule

## Parameters

You must set the following parameters for the LDAP login module:

Parameter	Description	Sample values
ldapProviderUrl	Mandatory. The IP address or the URL of the LDAP server.	ldap://10.0.1.2 ldaps://10.0.1.3
ldapTimeoutMs	Mandatory. The connection timeout to the LDAP server in milliseconds.	2000
ldapSecurityAuthentication	Mandatory. The LDAP security authentication type. The value is usually simple. Consult your LDAP administrator to obtain the relevant authentication type.	none simple strong
validationType	Mandatory. The type of validation. The value can be exists, searchPattern, or custom. See the following table for more details.	exists searchPattern custom
ldapSecurityPrincipalPattern	Mandatory. Depending on the LDAP server type, this parameter might require security credentials that you must supply in several formats. Some LDAP servers require only the user name, for example <i>john</i> , and others require the user name and the domain, for example <i>john@server.com</i> . You use this property to define the pattern to create your user name based credentials. You can use the {username} placeholder.	{username} {username}@myserver.com CN={username},DC=myserver,DC=com
ldapSearchFilterPattern	Optional. This parameter is required only if the value of the validationType parameter is searchPattern. You use this parameter to define a search filter pattern that is run when a successful LDAP binding is established. The user validation is successful if the search returns one or more entries. You can use the {username} placeholder. The syntax might change depending on the LDAP server type.	(sAMAccountName={username}) (&(objectClass=user)(sAMAccountName={username})(OU=MyCompany,DC=myserver,DC=com))
ldapSearchBase	Optional. This parameter is required only if the validationType parameter is searchPattern. Use this parameter to define the base of the LDAP search.	dc=myserver,dc=com

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})(OU=MyCompany,DC=myserver,DC=com))"/>
  <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.
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.

## Mobile device provisioning

Provisioning is the process of obtaining a security certificate. There are three modes of the provisioning process.

When an IBM 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*.

When a certificate is obtained, the app can then 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 IBM 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 must 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:

- 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: an IBM 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.

## Unique device ID

The unique device ID is used by IBM Worklight for device ID-related features, such as security, device SSO, reports, and push notifications.

- on iOS and Android:
  - To calculate the unique device ID, a globally unique ID (GUID) is used that is generated during device authentication process.
  - The unique device ID can be unique either to the application or to all applications from the same vendor.
    - on iOS:
      - The unique device ID is stored in the device keychain.
    - on Android:
      - The unique device ID is stored in the application keystore, which is a file in the application sandbox folder.
- on BlackBerry and Windows Phone:
  - To calculate the unique device ID, the ID that is provided by the operating system is used.
  - The unique device ID is global to the device.

**Note:** The availability of the unique device ID depends on the operating system of the device, and on the application vendor. A vendor that provides multiple applications that can be installed on the same device might then choose whether to require provisioning for each individual application or for a group of applications.

If several applications are from the same vendor, they can have the same unique device ID. If these applications are from different vendors, they have different unique device IDs.

To access the unique device ID on the device and on the IBM Worklight back-end server, some security controls are performed. The device ID is not a secret data, and can be passed to the server in one of the two following ways:

- As is, for a non-secure device authentication.
- Accompanied with credentials, for a secure device authentication. In that case, the device ID is digitally signed with a X509 certificate that is obtained as the result of the provisioning process that takes place at the first time the application runs on the device.

The unique device ID is stored in the raw data reports that are generated by IBM Worklight. There are no special access controls available on these reports, as the unique device ID is not considered sensitive data. For more information about raw data reports, see “Using raw data reports” on page 861.

For more information about mobile device provisioning, see the module *Device provisioning concepts*, under category 8. *Authentication and security*, in Chapter 3, “Tutorials and samples,” on page 25.

## 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. For information about how to specify IBM Worklight configuration properties, see “Configuration of IBM Worklight applications on the server” on page 714

#### **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`.

#### **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 Chapter 3, “Tutorials and samples,” on page 25.

## 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-enabled login module, the user gains access to all resources that are using the same 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.

### Implementing a custom authentication to support SSO

To allow SSO to operate on your custom authentication classes (authenticator and loginModule) you must:

1. Make all fields in your class transient except for those fields that are being used by the following methods:
  - WorklightAuthenticator.processRequestAlreadyAuthenticated(HttpServletRequest, HttpServletResponse)
  - WorklightAuthLoginModule.logout()
2. Mark the authenticator and loginModule classes (and any class referred to by those classes that is not transient after you perform step 1) with the class annotation @DeviceSSO(supported = true) .

## Configuring device single sign-on

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>
```

- 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 255.

- 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, use the `sso.cleanup.taskFrequencyInSeconds` server property and assign the required task interval value expressed in seconds. For information about how to specify IBM Worklight configuration properties, see “Configuration of IBM Worklight applications on the server” on page 714.

## IBM Worklight application authenticity overview

An overview of application authenticity features and procedures within IBM Worklight.

The IBM Worklight framework provides a number of security mechanisms. One of them is an application authenticity security test. Most Worklight security mechanisms are based on the same concept: obtaining identity through challenge handling. Just as the user authentication realm is used to obtain and validate a user’s identity, an application authenticity realm is used to obtain and validate an application’s identity. Therefore, this process is referred to as an *application authenticity*.

HTTP services (APIs) that are exposed by a Worklight Server can be accessed by any entity by issuing an HTTP request. This is why it is suggested that you protect relevant services with a number of security tests. Application authenticity makes sure that any application that tries to connect to a Worklight Server is authentic and was not tampered with or modified by some third-party attacker.

## Authenticity details

An application authenticity check uses the same transport protocol as other Worklight authentication framework realms:

1. The application makes an initial request to Worklight Server.
2. Worklight Server goes through the authentication configuration and finds that this application must be protected by an application authenticity realm.
3. Worklight Server generates a challenge token and returns it to application.
4. The application receives the challenge token.
5. The application processes the challenge token and generates a challenge response.
6. The application submits the challenge response to the Worklight Server.
7. If the challenge response is valid, Worklight Server serves the application with the required data.
8. If the challenge response is invalid, Worklight Server refuses to serve the application.

The two most important things to understand about step #5 are:

- The token is not processed by JavaScript; instead it is processed with compiled native code. This procedure ensures that a third-party attacker is not able to see the logic behind the token processing.
- Application authenticity is based on certificate keys that are used to sign the application bundle. Only the developer or enterprise who has access to the original private key that was used to sign the application is able to modify, repackage, and resign the bundle. This process makes this security measure bulletproof.

**Note:** In a cluster environment, the application authenticity setting is not synchronized between nodes. If you do need to modify the application authenticity setting in a production environment, you must do it on each cluster node separately.

## Enabling an application authenticity check (example)

Currently, application authenticity is supported only on iOS and Android platforms.

**Note:** The free Worklight Studio Developer Edition does not contain application authenticity-related components. You must use either Worklight Studio Consumer Edition or Worklight Studio Enterprise Edition to enable application authenticity.

The following sections present an example of how application authenticity is enabled for iOS and Android:

1. The first step in enabling application authenticity is to modify your `authenticationConfig.xml` file to add relevant authenticity realms to your security tests:



- If you use `<mobileSecurityTest>`, you must add the `<testAppAuthenticity/>` child element to this file.
- If you use `<customSecurityTest>`, you must add `<test realm="wl_authenticityRealm"/>` child element to the file.

Remember to rebuild and redeploy your `.war` file when you have updated your `authenticationConfig.xml` file.

- The second step is to modify the `application-descriptor.xml` file of your application. The procedure is different for Android and for iOS environments.
  - To enable an application authenticity check for an iOS environment, you must specify the `bundleId` of your application exactly as you defined it in the Apple Developer portal:



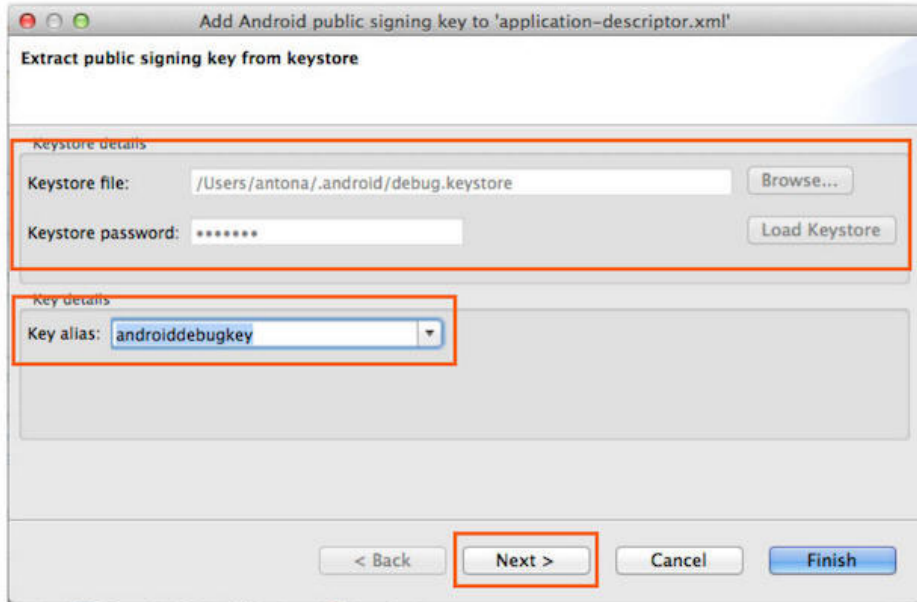
- To enable an application authenticity check for the Android environment, you must extract the public signing key of the certificate that is used to sign the application bundle (`.apk` file).

Worklight Studio provides tools to simplify this process. If you are building your application for distribution (production), you must extract the public key from the certificate you are using to sign your production-ready application. If you are building your application in a development environment, you can use the public key from a default development certificate that is supplied by Android. The development certificate can be found in a keystore that is located under `{user-home}/.android/debug.keystore`.

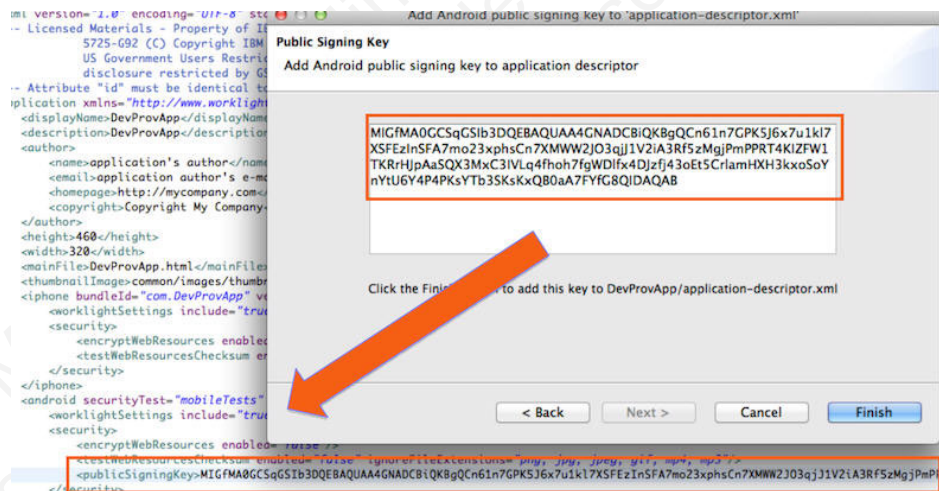
You can either extract the public key manually or use a wizard that is provided by Worklight Studio. To do the latter:

- Right-click your Android environment and select **Extract public signing key**.
- Specify the location and password of the keystore file and click **Load Keystore**.
- The default password for `debug.keystore` is **android**. Select key alias and click **Next**.





- The public key is displayed on a window:



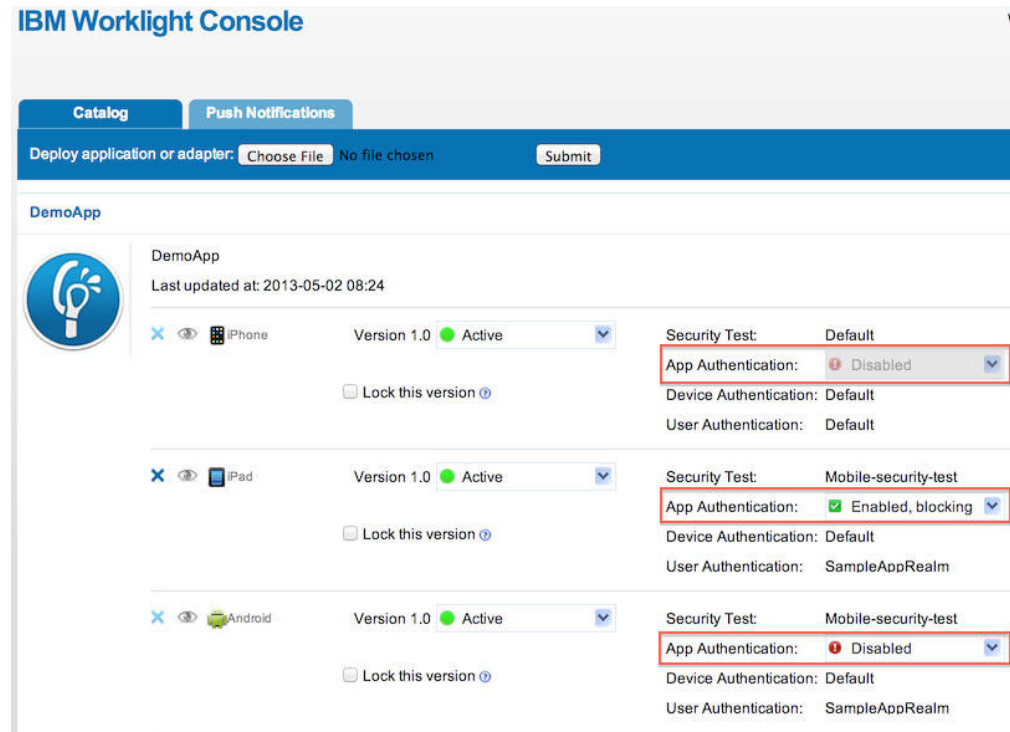
When you click **Finish**, the public key is automatically pasted into the relevant section of application-descriptor.xml.

3. After you have updated the required elements, remember to rebuild and redeploy your application to the Worklight Server.

## Controlling application authenticity from Worklight Console

Worklight Console allows enabling or disabling the application authenticity realm in run time. This feature is useful for the Development and QA environments. There are three modes you can set:

- **Enabled, blocking** – means that the application authenticity check is enabled. If the application fails the check, it is not served by a Worklight server.
- **Enabled, serving** – means that the application authenticity check is enabled. If the application fails the check, it is still served by a Worklight server.
- **Disabled** – means that the application authenticity check is disabled.



## Developing globalized hybrid applications

To develop globalized hybrid applications, learn about globalization in JavaScript frameworks and IBM Worklight, and about globalizing web services and push notifications.

Applications that are developed and uploaded to application stores must support multiple languages if they are to be used globally. IBM Worklight provides capabilities for you to develop globalized hybrid applications. This series of topics describes how to globalize your applications when using JavaScript frameworks and IBM Worklight, and how to globalize web services and push notifications.

### Globalization in JavaScript frameworks

You can use several JavaScript frameworks to globalize your applications: Dojo, jQuery, and Sencha Touch.

You can use the capabilities of different JavaScript frameworks to globalize your applications. Dojo, jQuery, and Sencha Touch each provide globalization functions that are based on resource bundles and resource files, and they can switch to different resource bundles based on current locale information. In addition, Dojo also provides string and date format utilities that are based on user locale information.

#### Dojo globalization framework

You can use the Dojo globalization framework to globalize your application.

The following example application demonstrates how to use the Dojo Mobile JavaScript API to construct a globalized application with a native look and feel. Dojo Mobile provides globalization functions for detecting locale, loading and accessing resource bundles, and simple formatting utilities for culture-sensitive

display. Figure 1 and Figure 2 show pages of the application that display the resource bundles loaded and the string format that is determined by the user preferences on the device.



Figure 54. Dojo globalization application



Figure 55. Dojo cultural formatting

In the example, the Dojo library is loaded as shown in Listing 1: Including Dojo Mobile. The required modules must be loaded before you can use the Dojo globalization API.

#### Listing 1: Including Dojo Mobile

```
<script type="text/javascript">
  var dojoConfig = {
    parseOnLoad: false,
    packages: [{
      name: "resource",
      location: "../../bundles"
    }]
  };
</script>
<script type="text/javascript" src="libs/dojox/mobile/deviceTheme.js"></script>
<script type="text/javascript" src="libs/dojo/dojo.js"></script>
```

Figure 3 shows how to load Dojo resource bundles by defining a package that maps the location of resource files within your hybrid application to a package name.

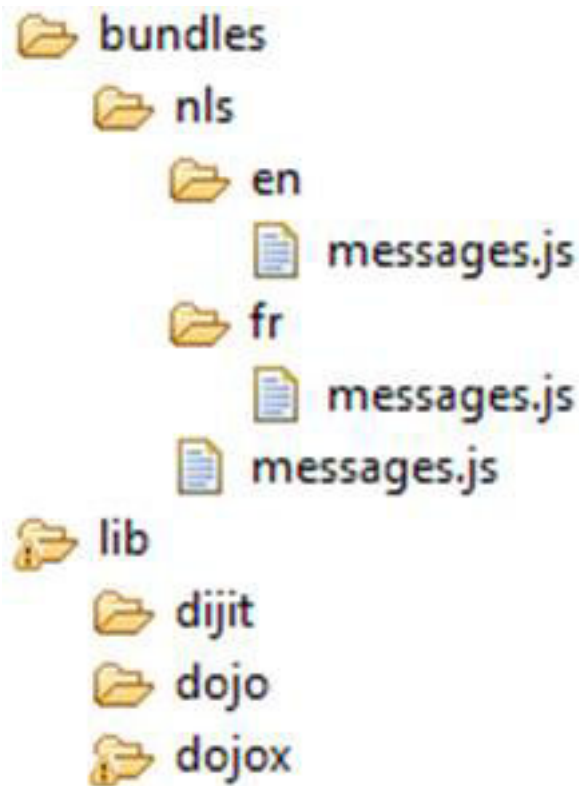


Figure 56. Dojo resource bundle structure

In the example, the language resource bundles are part of the application package, and are stored on the client-side instead of being supplied dynamically from the server or inserted directly into the HTML markup. Storing the language resource bundles on the client-side enables the application to be used offline. The resource files are encoded as JSON files.

### Listing 2: Globalization application Views

This listing shows the code to generate the pages as simple HTML markup. The following strings are the strings that you globalize in the application.

```

<!-- Main page -->
<div id="globalization" data-dojo-type="dojox.mobile.View" selected="true">
  <h1 id="globalization_heading" data-dojo-type="dojox.mobile.Heading" label="msg_globalization"></h1>
  <ul data-dojo-type="dojox.mobile.RoundRectList">
    <li id="months_choice" data-dojo-type="dojox.mobile.ListItem" moveTo="months" callback="getMonths"></li>
    <li id="days_choice" data-dojo-type="dojox.mobile.ListItem" moveTo="days" callback="getDays" label="msg_days"></li>
    <li id="formats_choice" data-dojo-type="dojox.mobile.ListItem" moveTo="formats" callback="getFormats"></li>
    <li id="icon_choice" data-dojo-type="dojox.mobile.ListItem" label="msg_icon"></li>
  </ul>
  <h1 id="globalization_footer" data-dojo-type="dojox.mobile.Heading" fixed="bottom" label="msg_footer"></h1>
</div>

<!-- The "Icon" sub-page -->
<div id="icons" data-dojo-type="dojox.mobile.View">
  <h1 id="icon_heading" data-dojo-type="dojox.mobile.Heading" moveTo="globalization" label="msg_icons"></h1>
</div>

<!-- The "Months" sub-page -->
<div id="months" data-dojo-type="dojox.mobile.View">
  <h1 id="months_heading" data-dojo-type="dojox.mobile.Heading" moveTo="globalization" label="msg_months"></h1>
</div>
  
```

```

<!-- The "Days" sub-page -->
<div id="days" data-dojo-type="dojox.mobile.View">
  <h1 id="days_heading" data-dojo-type="dojox.mobile.Heading" moveTo="globalization" label="msg_da
</div>

<!-- The "Formats" sub-page -->
<div id="formats" data-dojo-type="dojox.mobile.View">
  <h1 id="formats_heading" data-dojo-type="dojox.mobile.Heading" moveTo="globalization" label="msg
</div>

```

### Listing 3: Loading modules and resource files with the Dojo Mobile resource bundle API

This listing shows the resources files that are loaded by the `dojo/i18n` plug-in using Asynchronous Module Definition (AMD).

```

require(
  [
    "dojo/domReady",           // Make sure DOM are ready
    "dojo/i18n!resource/nls/messages", // Load our resource bundle
    "dojox/mobile/parser",    // This mobile app uses declarative programming
    "dojox/mobile",          // This is a mobile app
    "dojox/mobile/i18n",      // Load resources bundle declaratively
    "dojox/mobile/compat",    // This mobile app supports running on desktop browsers
  ],
  function(ready, messages, parser, mobile, mi18n) {
    ready( function() {
      // demonstrates how to load resources declaratively
      // dojox.mobile.i18n.load() must be called before dojox.mobile.parser.parse()
      mi18n.load("resource", "messages");
      parser.parse();
    });
  }
);

```

**Note:** The `dojo.18n.getLocalization` API is deprecated. Use `dojox/mobile/i18n` to load resources declaratively. The `dojox/mobile/i18n load()` method treats text in all mobile widgets as resource keys, and automatically replaces those keys with the actual resources. If you want to explicitly control how these resources are used, they can be loaded programmatically. The following listings show how to load these resources.

### Listing 4: Explicitly using the loaded resources

This listing shows how to use an argument such as `resource` to retrieve loaded resources.

```

require(
  [
    "dojo/domReady",           // Make sure DOM are ready
    "dojo/i18n!resource/nls/messages", // Load our resource bundle
    "dijit/registry",          // For registry.byId
    "dojox/mobile/parser",    // This mobile app uses declarative programming
    "dojox/mobile",          // This is a mobile app
    "dojox/mobile/compat",    // This mobile app supports running on desktop browsers
  ],
  function(ready, messages, parser, registry) {
    ready( function() {
      parser.parse();
      registry.byId("globalization_heading").set("label", messages["msg_globalization"]);
      registry.byId("months_choice").set("label", messages["msg_months"]);
      registry.byId("days_choice").set("label", messages["msg_days"]);
      registry.byId("formats_choice").set("label", messages["msg_formats"]);
      // get locale by dojo
    });
  }
);

```

```

        var footer_msg = bundle["msg_footer"] + dojo.locale;
        registry.byId("globalization_footer").set("label", footer_msg);
    });
}
);

```

### Listing 5: Dojo cultural formatting

This listing shows the Dojo cultural formatting functions.

```

function getFormats(){
    var formatsView = dojo.byId("formats");
    require(
        [
            "dojox/mobile/RoundRectList",
            "dojox/mobile/ListItem",
            "dojo/date/locale",
            "dojo/number",
            "dojo/currency"
        ],
        function(RoundRectList, ListItem, localeDate, localeNumber, localeCurrency){
            var formatsList = new RoundRectList({id: "formats_list"}).placeAt(formatsView);
            // get locale by dojo
            var myLocale = dojo.locale;
            // format locale date by dojo/date/locale
            var date = localeDate.format(new Date(), {locale: myLocale});
            var formattedDate = new ListItem({label: "Date: " + date});
            formatsList.addChild(formattedDate);
            // format with parameter
            date = localeDate.format(new Date(), {selector: 'date', formatLength: 'full'});
            formattedDate = new ListItem({label: "Date: " + date});
            formatsList.addChild(formattedDate);
            // format number
            var number = localeNumber.format(1234567.89);
            var formattedNumber = new ListItem({label: "Number: " + number});
            formatsList.addChild(formattedNumber);
            // format currency
            var currency = localeCurrency.format(1234.567, {currency: "USD"});
            var formattedCurrency = new ListItem({label: "Currency: " + currency});
            formatsList.addChild(formattedCurrency);
        }
    );
};

```

For more information about globalization with Dojo Mobile, see <http://dojotoolkit.org/reference-guide/1.9/dojox/mobile/internationalization.html#dojox-mobile-internationalization>.

### jQuery Mobile globalization plug-in

You can use jQuery globalization functions with the jQuery Mobile globalization plug-in.

There are no official jQuery globalization bundles. Here, the `jquery.i18n.properties-1.0.9.js` jQuery globalization plug-in is used to demonstrate jQuery globalization functions. The `jquery.i18n.properties-1.0.9.js` jQuery globalization plug-in can be downloaded from <http://code.google.com/p/jquery-i18n-properties/>.

The example application does not show the jQuery globalization string format feature because there is no official globalization string formatting plug-in for jQuery frameworks.



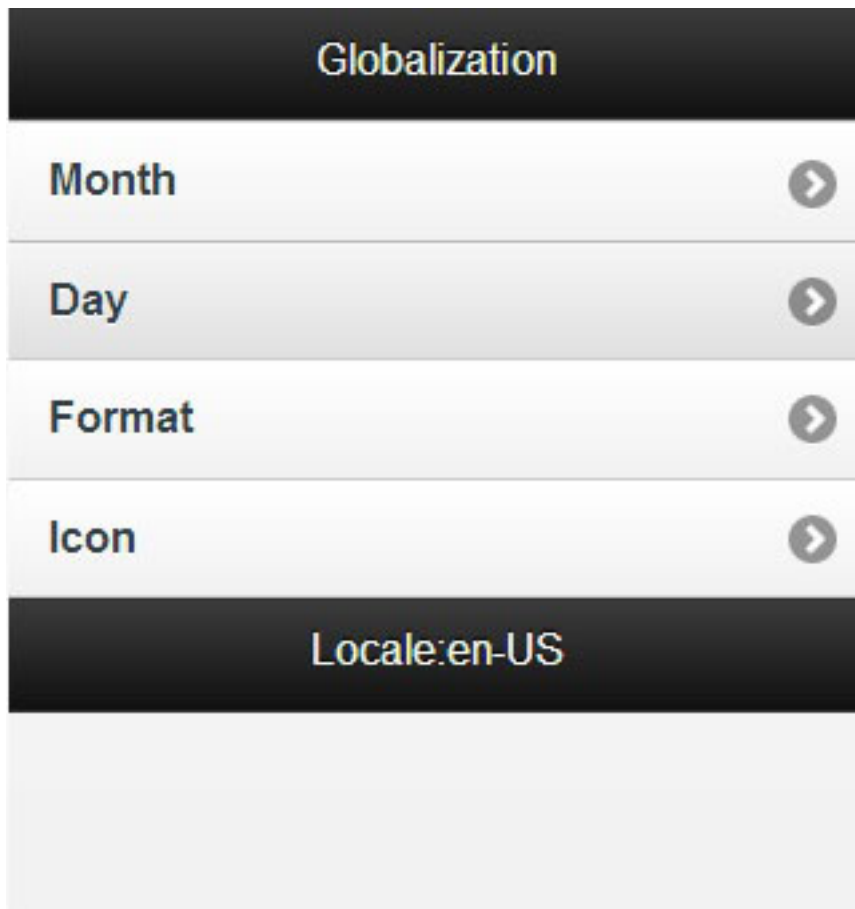


Figure 57. jQuery globalization application

### Listing 1: Load Cordova, jQuery mobile, and jQuery globalization plug-in

This listing shows the scripts for loading Cordova, jQuery mobile, and the jquery.i18n.properties-1.0.9.js jQuery globalization plug-in.

```

<script
  type="text/javascript"
  src="js/CordovaGlobalization.js">
</script>
<script
  type="text/javascript"
  src="js/messages.js">
</script>
<script
  type="text/javascript"
  src="js/jquery.mobile-1.1.1.min.js">
</script>
<script
  type="text/javascript"
  src="js/jquery.i18n.properties-min-1.0.9.js">
</script>

```

The resource bundle structures in jQuery and Dojo are different. Dojo resource files have the same file name but are in separate folders corresponding to the locale name. jQuery resource files are in one folder but the file names include the locale information. Figure 2 shows the structure of the jQuery resource files.

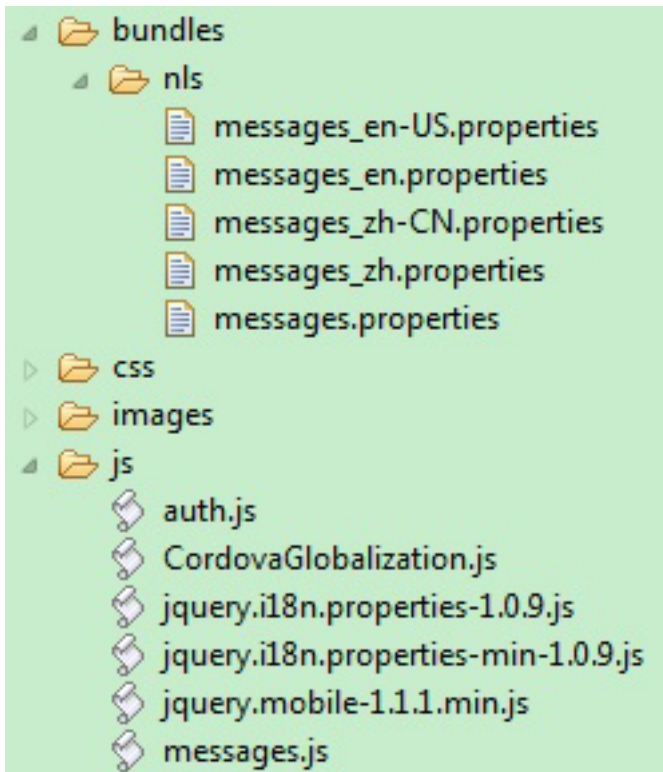


Figure 58. jQuery resource bundle structure

### Listing 2: Load and use resource by jQuery globalization plug-in

This listing shows the jQuery scripts to initialize the globalization plug-in.

```
function doGlobalization(){
  $.i18n.properties({
    name: 'messages',
    path: 'bundles/nls/',
    mode: 'both',
    // language: 'zh',
    callback: function(){
      // Main page
      $('#globalization_heading').empty().
        append($.i18n.prop('msg_globalization'));
      $('#msg_months').empty().append($.i18n.prop('msg_months'));
      $('#msg_days').empty().append($.i18n.prop('msg_days'));
      $('#msg_formats').empty().append($.i18n.prop('msg_formats'));
      $('#msg_icon').empty().append($.i18n.prop('msg_icon'));
      // Sub page heading
      $('#icon_heading').empty().append($.i18n.prop('msg_icon'));
      $('#months_heading').empty().append($.i18n.prop('msg_months'));
      $('#days_heading').empty().append($.i18n.prop('msg_days'));
      $('#formats_heading').empty().append($.i18n.prop('msg_formats'));
      $('#words_heading').empty().append($.i18n.prop('msg_words'));
      //Back buttons
      var items = $('a[data-rel="back"]');
      $.each(items, function(i){
        $(items[i]).empty().append($.i18n.prop('msg_previous'));
      });
      //Show locale by jQuery i18n plug-in
      $('#globalization_footer').empty().
        append($.i18n.prop('msg_footer') + $.i18n.browserLang());
    }
  });
};
```

## Sencha Touch globalization plug-in

You can use Sencha Touch globalization functions with the Sencha Touch globalization plug-in.

There are no official Sencha Touch globalization bundles. Here, the Ext.i18n.bundle-touch Sencha Touch globalization plug-in is being used to demonstrate globalization functions. The Ext.i18n.bundle-touch globalization plug-in can be downloaded from <http://gaver.dyndns.org/elmasse/site/index.php/download-menu/9-sencha-touch/21-exti18nbundle-touch-downloads>.

The example application does not show the Sencha Touch globalization string format feature because there is no official globalization string formatting plug-in for Sencha frameworks.

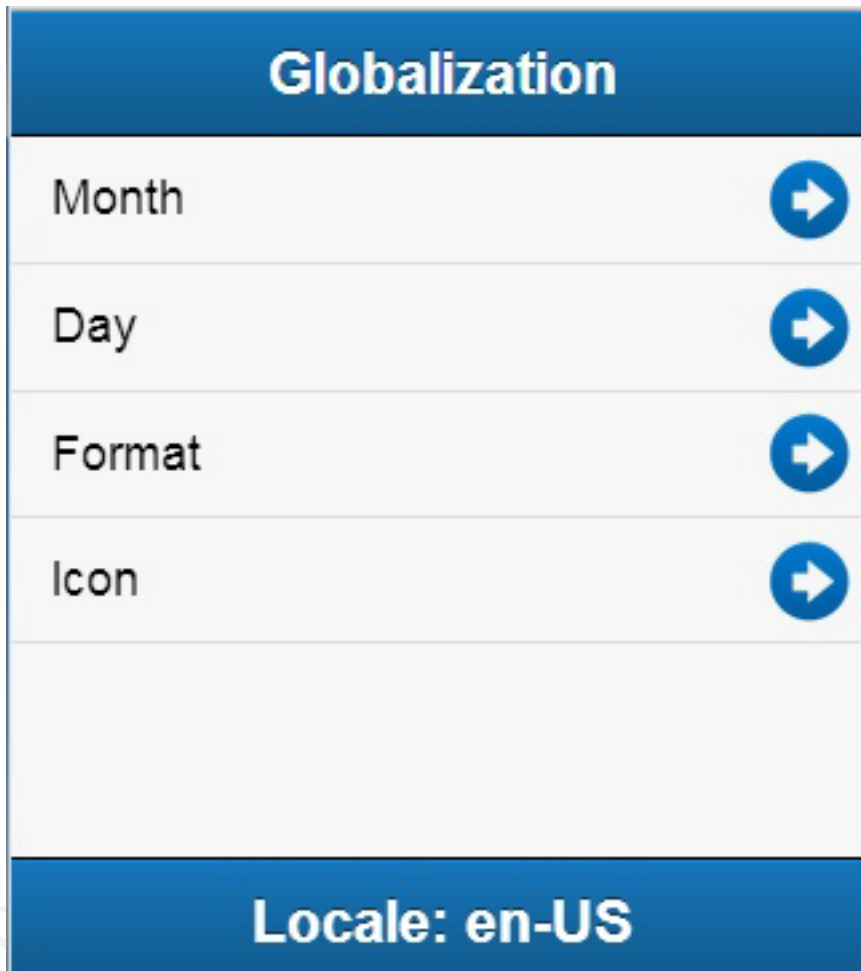


Figure 59. Sencha Touch globalization application

### Listing 1: Load Sencha Touch and globalization plug-in

This listing shows the scripts for loading Sencha Touch and the Ext.i18n.bundle-touch globalization plug-in.

```
<script src="js/sencha-touch-all.js"></script>
<script>
  Ext.Loader.setConfig({
    enabled: true,
    paths: {
```

```

        'Ext.i18n': 'js/i18n',
        'patch': 'js/patch'
    }
    });
</script>
<script src="js/SenchaGlobalization.js"></script>
<script src="js/messages.js"></script>
<script src="js/auth.js"></script>

```

Figure 2 shows the structure of the Sencha Touch resource files.

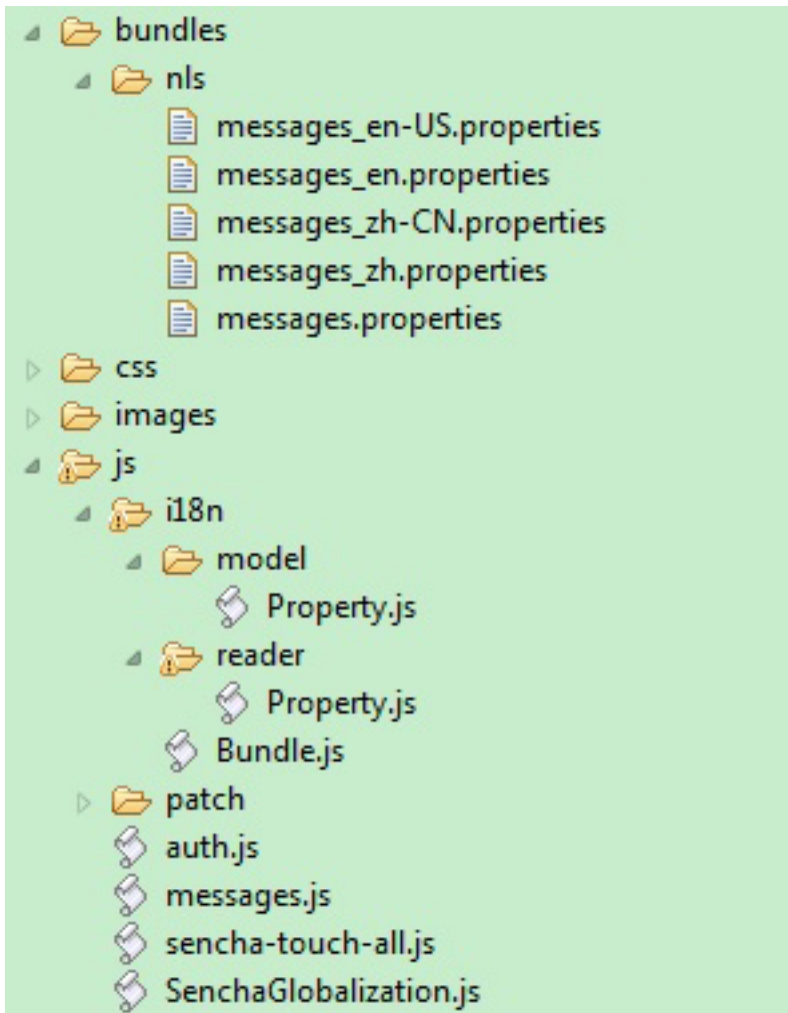


Figure 60. Sencha Touch resource bundle structure

Sencha Touch also provides a convenient API to retrieve the message in the resource bundle and set the value to the UI component.

### Listing 2: Load and use resource by Sencha Touch globalization plug-in

```

function loadResource(){
    Ext.require('Ext.i18n.Bundle', function(){
        Ext.i18n.appBundle = Ext.create('Ext.i18n.Bundle', {
            bundle: 'messages',
            path: 'bundles/nls',
            noCache: true
        });
    });
}

```

```

});
Ext.application({
    name: "Sencha Touch Globalization",
    launch: function(){
        Ext.i18n.appBundle.onReady(function(){doGlobalization();});
    }
});
};

function doGlobalization(){
    // global header
    var globalHeader = Ext.create('Ext.Toolbar', {
        docked: 'top',
        xtype: 'toolbar',
        title: '<div width="100px">' +
            Ext.i18n.appBundle.getMsg('msg_globalization') + '</div>'
    });
    // show locale by Ext.i18n.Bundle
    var globalFooter = Ext.create('Ext.Toolbar', {
        docked: 'bottom',
        xtype: 'toolbar',
        title: '<div width="100px">' +
            Ext.i18n.appBundle.getMsg("msg_footer") +
            Ext.i18n.appBundle.language + '</div>'
    });
    // main list data model
    Ext.define('mainListModel', {
        extend: 'Ext.data.Model',
        config: {fields: ['index', 'type']}
    });
    // main list data store
    var mainListStore = Ext.create('Ext.data.Store', {
        model: 'mainListModel',
        sorters: 'index',
        proxy: {
            type: 'localStorage',
            id: 'mainListStore'
        },
        data: [
            {
                index: '1',
                type: Ext.i18n.appBundle.getMsg('msg_months')
            },
            {
                index: '2',
                type: Ext.i18n.appBundle.getMsg('msg_days')
            },
            {
                index: '3',
                type: Ext.i18n.appBundle.getMsg('msg_formats')
            },
            {
                index: '4',
                type: Ext.i18n.appBundle.getMsg('msg_words')
            },
            {
                index: '5',
                type: Ext.i18n.appBundle.getMsg('msg_icon')
            }
        ]
    });
    // main list view
    var mainList = Ext.create('Ext.List', {
        itemTpl: '{type}',
        store: mainListStore,
        onItemDisclosure: function(record, btn, index){

```

```

        showSecondContainer(record, btn, index);
    }
    });
}

```

## Globalization mechanisms in IBM Worklight

IBM Worklight automatically translates application strings according to a designated file. Multi-language translation is implemented by using JavaScript.

### Cordova globalization API

The Cordova globalization API provides enhanced globalization capabilities that mirror existing JavaScript globalization functions, where possible, without duplicating functions already present in JavaScript. The emphasis of the Cordova globalization API is on parsing and formatting culturally sensitive data. The Cordova API uses native functions in the underlying operating system, where possible, rather than re-creating these functions in JavaScript. Table 1 summarizes the Cordova globalization API functions provided.

Table 70. Cordova globalization API summary

Function Name	Purpose
getPreferredLanguage	The current language of the client.
getLocaleName	The client current locale setting on the device.
dateToString	A date that is formatted as a string, according to the locale and timezone of the client.
stringToDate	A string that is parsed as a date, according to the client's user preferences.
getDatePattern	A pattern string for formatting and parsing dates.
getDateNames	The names of the months, or the days of the week.
isDayLightSavingsTime	Whether daylight saving time is in effect for a specified date.
getFirstDayOfWeek	The first day of the week.
numberToString	A number that is formatted as a string, according to the user preferences.
stringToNumber	A string that is parsed as a number, according to the user preferences.
getNumberPattern	A pattern string for formatting and parsing numbers.
getCurrencyPattern	A pattern string for formatting and parsing currencies.

The Cordova globalization API is an independent globalization framework, which can be integrated with any JavaScript libraries to provide globalization functions. The Cordova globalization API is different from other globalization libraries. The Cordova globalization API does not provide a parameter to indicate a locale. The set of supported locales is determined by the device and its SDK and not by Cordova.

The Cordova globalization API uses the client locale setting and any default values that are overridden. This design greatly simplifies the use of the globalization API while still providing robust support. It is important to note that, although the set of interfaces remains constant across the devices that Cordova supports, the results can vary across the devices.

The Cordova framework does not provide access to graphical widgets that are present in device SDKs. The Cordova framework is used in concert with other JavaScript widget libraries, such as Dojo, to build complete mobile applications. The Cordova globalization API is interoperable with Dojo Mobile, jQuery Mobile, and Sencha Touch. It is an asynchronous implementation to prevent blocking JavaScript execution in user interface code. The following listings and figures show the Cordova globalization API. Dojo is used to demonstrate the user interface.

*Table 71. Listing 1: Using the Cordova globalization API*

```
function onDeviceReady(){
    g11n = window.plugins.globalization;
}
```

The code to generate the names of the months, days of the week, and format the current date is shown in Listing 2, Listing 3, and Listing 4.

*Table 72. Listing 2: Month names*

```
function getMonths(){
    g11n.getDateNames(function(data){
        var items = data.value;
        var monthsView = document.getElementById('monthsView');
        for (var i = 0; i < items.length; i++) {
            monthsView.append('<li>' + items[i] + '</li>');
        }
    },
    function(code){
        alert("Error: " + code);
    });
};
```





Figure 61. Using Cordova to show locale-based months

Table 73. Listing 3: Days of the week

```
function getDays(){
  g11n.getDateNames(function(data){
    var items = data.value;
    var daysView = document.getElementById('daysView');
    for (var i = 0; i < items.length; i++) {
      daysView.append('<li>' + items[i] + '</li>');
    }
  },
  function(code){
    alert("Error: " + code);
  },
  {
    item: "days"
  });
}
```



Figure 62. Using Cordova to show the days of week

Table 74. Listing 4: Formatting current date

```
function getFormats(){
  var formatsView = document.getElementById('formatsView');
  g11n.dateToString(
    new Date(),
    function(date){
      formatsView.append('<li>' + date.value + '</li>');
    },
    function(code){alert("Error: " + code);},
    {selector: "date", formatLength: "full"}
  );
  g11n.getDatePattern(
    function(date){
      formatsView.append('<li>' + date.pattern + '</li>');
      var timeZone = date.timezone;
      formatsView.append('<li>' + timeZone + '</li>');
      var offset = date.utc_offset;
      formatsView.append('<li>' + offset + '</li>');
      var dstoffset = date.dst_offset;
      formatsView.append('<li>' + dstoffset + '</li>');
    },
    function(code){
      alert("Error: " + code);
    },
    {selector: "date", formatLength: "full"}
  );
  g11n.isDayLightSavingsTime(
    new Date(),
    function(date){
      var dst = date.dst;
      formatsView.append('<li>' + dst + '</li>');
    },
    function(code){
      alert("Error: " + code);
    }
  );
  g11n.numberToString(
    1234.56,
    function(number){
      formatsView.append('<li>' + number.value + '</li>');
    },
    function(code){
      alert("Error: " + code);
    },
    {type: "decimal"}
  );
}
```

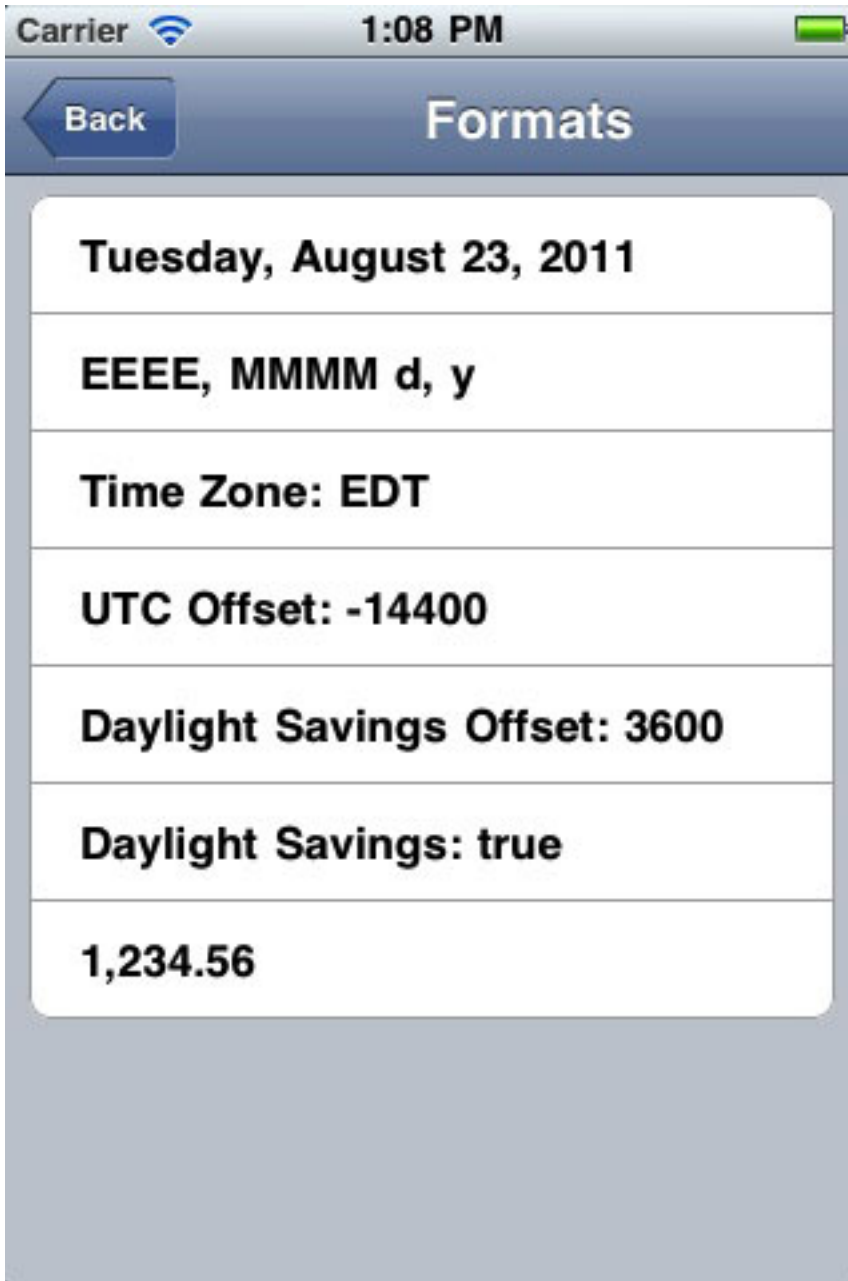


Figure 63. Using Cordova for cultural formatting

### Enabling translation of application strings

messages.js is the file that is designated for application strings and can be found in the common/js folder. If you use Dojo, jQuery, or Sencha Touch in your application, use the translation resource loading mechanisms and file formats from these JavaScript technologies instead of mechanisms that are provided by IBM Worklight. Use IBM Worklight application messages only when the JavaScript graphical toolkit used in your application does not provide these services.

```

Messages = {
  headerText: "Default header",
  actionsLabel: "Default action label",
  sampleText: "Default sample text",
  englishLanguage : "English",
  frenchLanguage : "French",
}

```

Application messages that are stored in `messages.js` can be referenced in two ways:

- As a JavaScript object property; for example, `Messages.header` or `Messages.sampleText`.
- As the ID of an HTML element with `class="translate"`.

```

<div id="header">
  <h1 id="headerText" class="translate"></h1>
</div>

```

**Note:** A string that is defined in `Messages.headerText` is automatically used here.

## Implementing multi-language translation

You can implement multi-language translation for your applications by using JavaScript.

1. Define default application strings in `messages.js` as shown in the following code snippet:

```

Messages = {
  headerText: "Default header",
  actionsLabel: "Default action label",
  sampleText: "Default sample text",
  englishLanguage : "English",
  frenchLanguage : "French",
  russianLanguage : "Russian",
  hebrewLanguage : "Hebrew"
};

```

2. Override some or all of the default application strings with JavaScript. The following two code snippets define JavaScript functions that are used to override the default strings that are defined in `messages.js`:

```

function setFrench(){
    Messages.headerText = "Traduction";
    Messages.actionsLabel = "Sélectionnez langue:";
    Messages.sampleText = "ceci est un exemple de texte en français.";
}
function setRussian(){
    Messages.headerText = "Перевод";
    Messages.actionsLabel = "Выбор языка:";
    Messages.sampleText = "Это пример текста на русском языке.";
}

function languageChanged(lang){
    if (typeof(lang)!="string") lang = $("#languages").val();
    switch (lang){
        case "english":
            setEnglish();
            break;
        case "french":
            setFrench();
            break;
        case "russian":
            setRussian();
            break;
        case "hebrew":
            setHebrew();
            break;
    }
    if ($("#languages").val()=="hebrew") $("#AppBody").css({direction: 'rtl'});
    else $("#AppBody").css({direction: 'ltr'});

    $("#sampleText").html(Messages.sampleText);
    $("#headerText").html(Messages.headerText);
    $("#actionsLabel").html(Messages.actionsLabel);
}

```

A language parameter is passed to the `languageChanged()` JavaScript function. The `languageChanged()` function calls the corresponding function to override the default English language string.

### Detecting device-specific information

You can detect the locale and language of your device by using `WL.App.getDeviceLocale()` and `WL.App.getDeviceLanguage()`.



```

var locale = WL.App.getDeviceLocale();
var lang = WL.App.getDeviceLanguage();
WL.Logger.debug(">> Detected locale: " + locale);
WL.Logger.debug(">> Detected language: " + lang);

```

The following screen capture shows the print output:

```

05-12 12:27:19.685 D 26294 before: app init onSuccess
05-12 12:27:19.735 D 26294 >> Detected locale: en_US
05-12 12:27:19.745 D 26294 >> Detected language: en
05-12 12:27:19.775 D 26294 after: app init onSuccess

```

## Globalization of web services

You can use the Cordova globalization method to get the user locale preference, and check what user locale is used.

In some situations, localized results are obtained by calling web services. The Cordova globalization method `getLocaleName` returns the user locale preference, which can be used in client-driven service calls.

The following listing shows how the user locale is used to collate a list of words. The locale of the returned word list can be checked to verify that the user locale was used or a substitute locale was used instead.

### Listing: Locale-based service call

```

function getWords(){
    var services;
    require(
        ["dojox/rpc/Service"],
        function(Service){
            services = new Service({
                target: "{Your Web Service URL}",
                transport: "POST",
                envelope: "JSON-RPC-1.0",
                contentType:
                    "application/json",
                services: {
                    "sorter.getWordList": {
                        returns: {"type": "object"},
                        parameters: [{"type": "string"}]
                    }
                }
            });
        }
    );
    g11n.getLocaleName(
        function(locale){
            // invoke the JSON web service to get the list of sorted words
            var deferred = services.sorter.getWordList(locale.value);
            deferred.addCallback(
                function(result){
                    var wordsView = dojo.byId("words");
                    require(
                        [
                            "dojox/mobile/RoundRectList",

```

```

        "dojox/mobile/ListItem",
        "dojox/mobile/Heading"
    ],
    function(RoundRectList, ListItem, Heading){
        var wordsList = new RoundRectList({
            id: "words_list").placeAt(wordsView);
        items = result.words.list;
        for (var i = 0; i < items.length; ++i) {
            var word = new ListItem({label: items[i]});
            wordsList.addChild(word);
        }
        var wordsFooter = new Heading({
            label: result.localeName}).placeAt(wordsView);
    });
}
)
},
function(code){
    alert("Error: " + code);
}
);
};

```

## Globalization of push notifications

With IBM Worklight, you can globalize push notifications so that push notifications are displayed in the language of the user. You use different methods to globalize push notifications, depending on the way the application runs: in the foreground, in the background, or not running at all.

Mobile applications frequently rely on server-side services to provide data to the mobile application. However, there are occasions when the application is not running or is not connected to the server. Push notifications are short messages that provide a mechanism for notifying mobile applications that a server has data that can be downloaded or information that can be viewed by the mobile application when the application is either not running or not running in the foreground.

Translate push notification messages so that the correct language is displayed to the user. How the application runs, such as, in the foreground, in the background, or not running at all, determines your choice of architectural pattern.

- When the application is running in the foreground, it uses the language and cultural settings on the device to determine the appropriate language to display. To support this pattern, messages must be stored in the resource files of the mobile application, and not in the resource files of the server application, even though messages are generated on the server-side.
- When a notification is sent to a mobile application, send the notification resource key and not the actual text of the message.
- When a mobile application receives the notification, or message, use the key that was sent in the notification message to look up the text of the message from its resource file, as shown in Figure 1.

**Note:** iOS uses Apple Push Notification Service (APNS) to push notifications to mobile applications. Android uses Google Cloud Messaging (GCM) to push notifications to mobile applications.

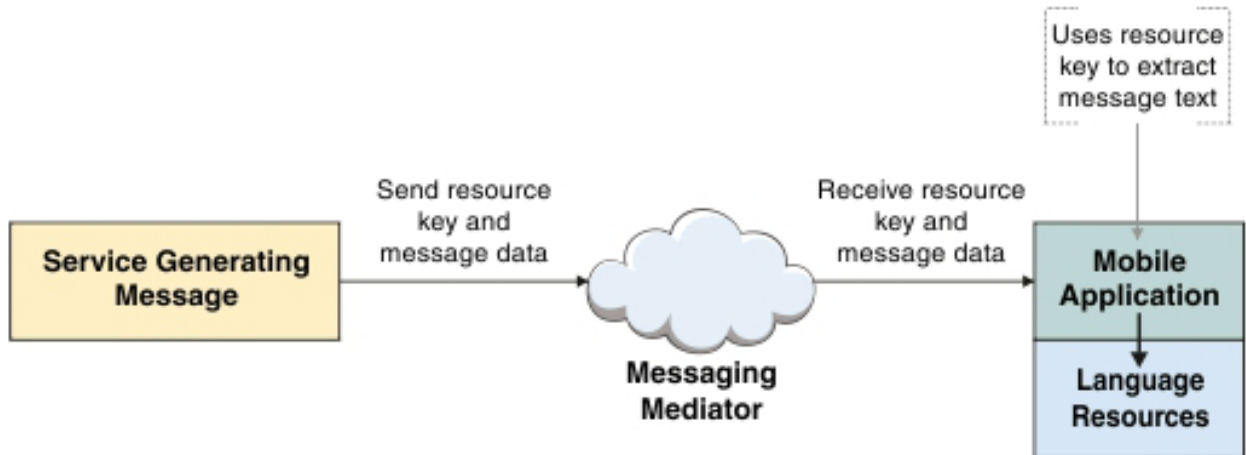


Figure 64. Using the resource key

Listing 1, Listing 2, and Listing 3 show sample code that can be used when the mobile application is running in the foreground.

First, create an IBM Worklight adapter to send the notification. For more information about how to create an adapter, see the module *Push Notifications*, under category 9, *Advanced topics*, in Chapter 3, “Tutorials and samples,” on page 25

### Listing 1: Send notification using created adapter

This listing shows how to send a notification with the created adapter. The target message, or resource key, to be translated on the client-side, is specified in the payload.

```

function sendNotificationOTA(userId, notificationText) {
  var userSubscription = WL.Server.getUserNotificationSubscription(
    'mysuranceAdapter.mysuranceEventSource', userId);
  WL.Server.notifyAllDevices(
    userSubscription,
    {
      badge : 1,
      sound : "",
      alert : notificationText,
      payload : { globalizeString : 'notificationText' }
    }
  );
}

```

### Listing 2: Client-side subscription code

This listing shows the code that is required on the client-side to subscribe to push notification.

```

WL.Client.Push.onReadyToSubscribe = function(){
  WL.Client.Push.registerEventSourceCallback(
    "mysurancePush",
    "mysuranceAdapter",
    "mysuranceEventSource",
    pushNotificationReceived);
};

```

After successful subscription, the callback method is implemented. The callback method is responsible for retrieving the data from the payload, retrieving application locale preferences, retrieving the message by using the resource key, and formatting the message.

### Listing 3: Callback method

This listing shows how to retrieve the locale information and load the corresponding translated message with Dojo by using the resource key that is stored in the payload object.

```
function pushNotificationReceived(props, payload){
  if (payload.globalizeString != "undefined"){
    require(
      ["dojox/mobile/i18n", "dojo/number"],
      function(mi18n, number){
        bundle = mi18n.load("resource", "messages");
        // get globalization text by dojo mobile i18n
        var notificationText = bundle[payload.globalizeString];
        // format number by device locale
        var num = number.format(1234567890, {
          places: 2, locale: WL.App.getDeviceLocale()});
        num = bundle["amount"] + num;
        //display globalization message
        alert(notificationText + "\n" +num);
      }
    );
  }
}
```

- If a notification provides data in addition to the message, then send the data in a locale-neutral format. When the application retrieves the message, the data can be formatted based on the user cultural preferences at the time the message is received.
- An application that is running in the background, or not running at all, can elect to use a previously registered user profile to access the appropriate language and cultural settings for push notifications. To support this pattern, the server sends the translated message and data in a format that is determined by the user cultural and language preferences that are stored in the profile, as shown in Figure 2. The push notifications are then processed differently by the mobile application. Processing is based on the native operating system that the application is running on.

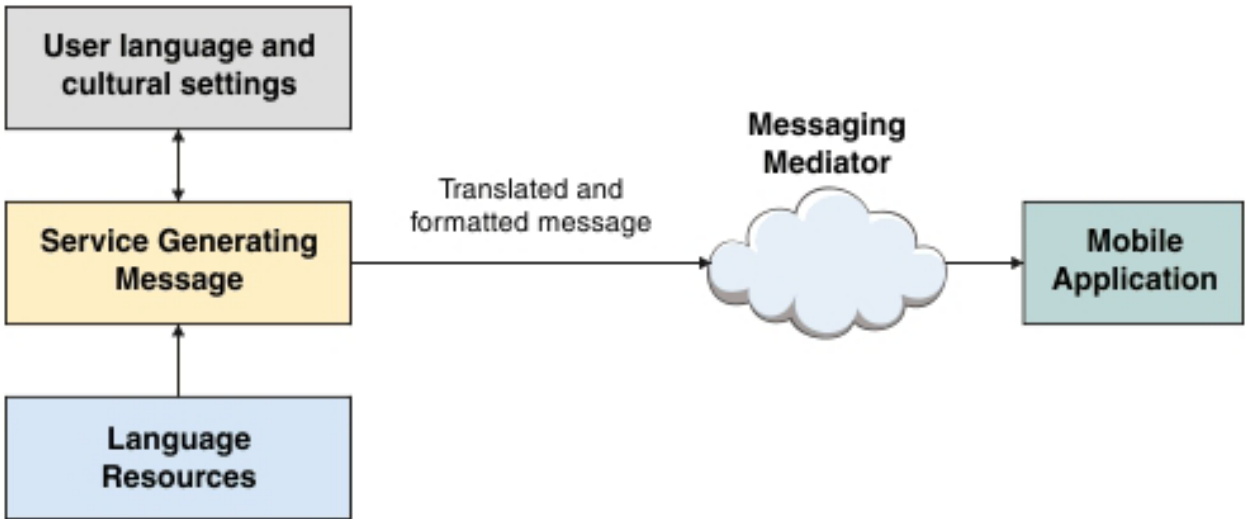


Figure 65. Sending push notifications according to the user's settings

- On Android, notification messages wake up Android applications, and the applications directly access the language and cultural preferences so that the correct translation and formatting can be applied.
- On iOS, notification messages do not wake up iOS applications, therefore the native operating system automatically selects the appropriate language to use for notifications. The iOS operating system automatically attempts to locate and load the correct language resource.
- In hybrid applications that are built using IBM Worklight, notifications are not directly processed by the application when the application is not running in the foreground. In this case, the user language and cultural profile that was previously established is used.

## Developing accessible applications

To develop *accessible* applications, easily used by people with disabilities, this topic helps you to learn about resources available to improve the accessibility of your apps.

When you build an application for your business, it is important to consider the user experience of individuals with a disability or impairment. Taking steps to consider enablement of tools like screen magnification, audio assistance, or other assistive technologies can extend the reach of your business.

In general, mobile applications can be made highly accessible. This following sections provides a number of resources to help you make your mobile application as accessible as possible. IBM Worklight provides a strong foundation for building accessible applications because it supports industry standards and allows you to leverage them. But accessibility features vary among target environments, depending on the Native OS or the Hybrid library vendor.

## Native application accessibility

If your application is native, the ability to make it accessible is determined by the capabilities of the target platform itself. The links that follow provide excellent resources for the supported mobile platforms, laying out available options and capabilities.

- **iOS**
  - Accessibility in iOS
  - Understanding Accessibility on iOS
  - iOS. A wide range of features for a wide range of needs.
- **Android**
  - Accessibility
- **BlackBerry**
  - Accessibility
  - Introduction to the Accessibility API
  - Accessibility API concepts
  - Developing accessible BlackBerry device applications by using the Accessibility API
  - Test an accessible BlackBerry device application
- **Windows Phone**
  - Accessibility on Windows Phone

## Hybrid application accessibility

If your application is a hybrid, options are available from a number of JavaScript libraries. Dojo Mobile and jQuery Mobile are popular examples, but there are several others. Useful references for writing accessible hybrid applications are provided below. Note that if you are using Dojo Mobile, version 1.9 or newer is highly suggested because it has better accessibility coverage than previous versions.

- **Dojo Mobile**
  - Dojo Accessibility Design Requirements
  - Dojo Accessibility
- **jQuery Mobile**
  - Accessibility

---

## Location services

Location services in IBM Worklight provide you with the opportunity to create differentiated services that are based on a user location, by collecting geolocational and WiFi data, and by feeding the location data and triggers to business processes, decision management systems, and analytics systems.

Geolocation is a powerful differentiator of mobile apps. Yet because geolocation coordinates must be constantly polled to understand where a mobile device is located, the resulting stream of geographic information can be difficult to manage without exhausting resources such as battery and network. Worklight includes location services that handle multiple geo modalities such as GPS, WiFi sampling, and interpolation. You can set policies for acquiring geolocation data and transmitting it to the server in order to optimize battery and network usage.

With location services in IBM Worklight, you can use data that is acquired by a mobile device to trigger events that benefit both the owner of the device and the enterprise that has received the data. For example:

- A fast food outlet could start preparing food for a customer, based on data collected regarding his geographical location, so that the food is ready just as the customer arrives to collect.
- A warehouse could improve the efficiency of its processes by using locational data from its delivery vehicles to ensure that goods are removed from storage and made ready for collection.
- Shopping outlets could respond more readily to the needs of regular customers by using geo-locational data.

Location services can also be used to improve internal efficiency within an organization, for example, by understanding behavior and trends in application usage, and thus driving ongoing improvement.

Location services are currently supported on hybrid Android and iOS.

The following figure shows how the location services feature works:

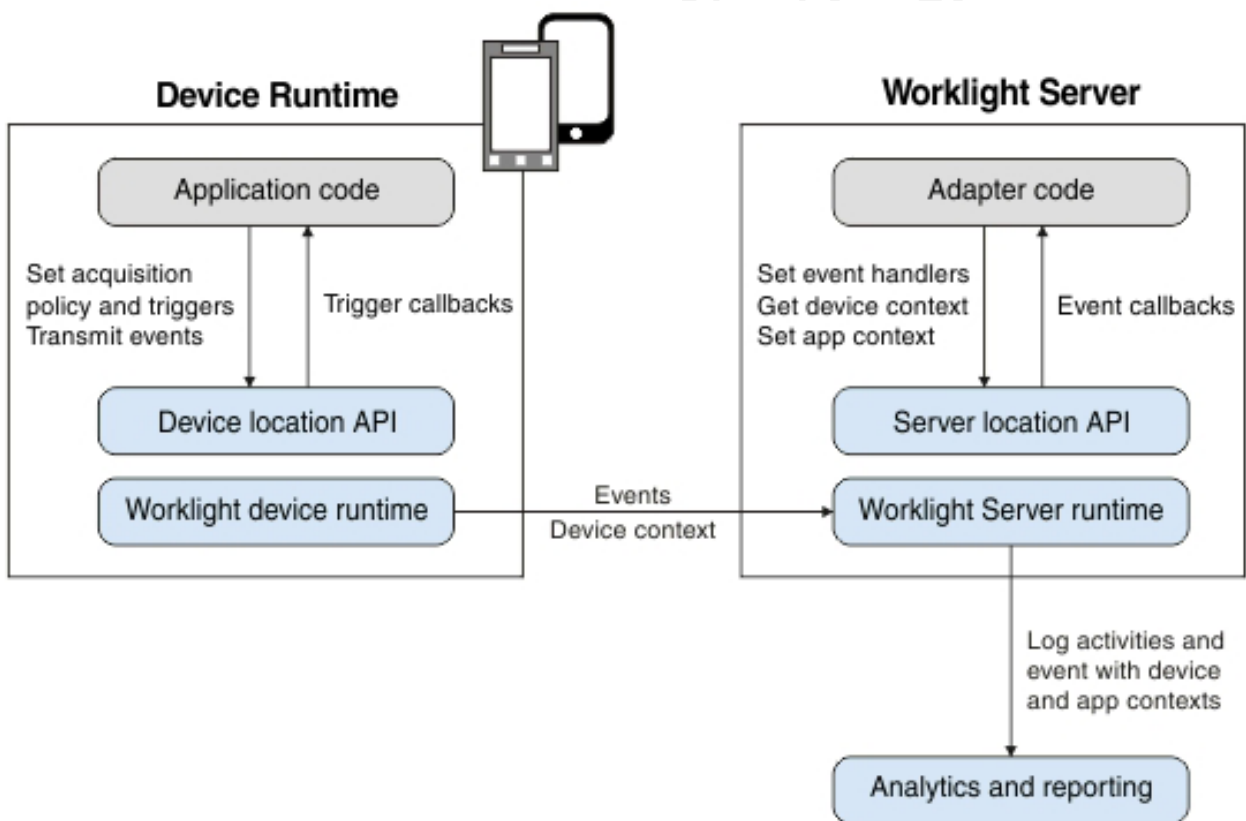


Figure 66. Location services architecture

Application code on the mobile device, in the form of an *acquisition policy*, controls the collection of data from device sensors. The collected data is referred to as the *device context*. When a change occurs in the device context, such as a change in the geolocation of the device, or the fact that it has just entered a WiFi zone, triggers can be activated. The triggers specify that an action should occur: either a callback function is called, or an event is sent to the server, based on the device context.



Events are created by triggers and application code, and include a snapshot of the device context at the time of their creation. Events are buffered on the client, and are transmitted to the server periodically. The server might process the event much later. During the event transmission process, the device context is synched transparently to the server.

To handle the events, the server uses adapter application code. This code sets up event handlers on the server, which filter event data and pass matching events to a callback function. The code also accesses the client's device context (its location and WiFi network information) and sets an application context. Server activities and received events are logged, together with the device and application contexts, for future reporting and analytics.

## Platform support for location services

Location services are supported for hybrid applications on iOS and Android in IBM Worklight V6.0. However, the level of support for each platform is slightly different.

The following table lists the features of location services where support differs for iOS and Android:

Feature	iOS	Android
Geo acquisition policy	minChangeTime is not supported.	highAccuracyOptions: iOSBestAccuracy is not supported.
WiFi visible access points	Not supported. Only the Connect and Disconnect triggers are supported for iOS WiFi.	
Connected WiFi signal strength	Not supported.	
KeepAliveInBackground	Not supported. Use standard iOS options for acquiring location data while in the background.	

For information about iOS and Android permissions, see “Location services API” on page 553.

## Triggers

A trigger is a mechanism that detects an occurrence, and can cause additional processing in response. Triggers are activated when a change occurs in the device context.

Triggers can be activated for changes in Geo or WiFi data.

### Geo triggers

For Geo data, two types of regions, also known as *geofences*, are considered: circles and polygons. The following trigger types are available for Geo data.

Trigger type	Description
PositionChange	The trigger is activated when the position of the device changes by at least a specified distance.
Enter	The trigger is activated when the device enters a region.
Exit	The trigger is activated when the device leaves a region.
DwellInside	The trigger is activated when the device remains inside a region for a given period of time.
DwellOutside	The trigger is activated when the device remains outside a region for a given period of time.

For Enter, Exit, DwellInside, and DwellOutside, you can increase or decrease the size of the region by altering the buffer zone width. Sensor accuracy is measured by using GPS coordinates and network accuracy.

You can control trigger activation based on confidence levels. For example, if you choose a confidence level of low, accuracy is not taken into account when you are determining whether a geo-locational coordinate acquired from a device is inside or outside a region. If you choose a confidence level of medium, accuracy is taken into account, and you can be sure that the coordinate lies within, or outside of, the region at approximately a 70% confidence level. If you choose a confidence level of high, accuracy is taken into account, and you can be sure that the coordinate lies within, or outside of, the region at approximately a 95% confidence level.

## WiFi triggers

For WiFi data, triggers are activated based on a change in visible access points. Access points are defined by using SSIDs (service set identifiers) and MACs (media access control addresses). The following trigger types are available for WiFi data.

Trigger type	Description
VisibleAccessPointsChange	The trigger is activated when the visible access points that define a WiFi area change by a specified amount.
Enter	The trigger is activated when the device enters a WiFi area.
Exit	The trigger is activated when the device leaves a WiFi area.
DwellInside	The trigger is activated when the device remains inside a WiFi area for a given period of time.
DwellOutside	The trigger is activated when the device remains outside a WiFi area for a given period of time.
Connect	The trigger is activated when the device connects to a WiFi access point.
Disconnect	The trigger is activated when the device gets disconnected from a WiFi access point.

For detailed information about the parameters for the trigger types, see “WL.Device.startAcquisition” on page 561.

## Setting an acquisition policy

You can set up a location services acquisition policy that is based on your requirements. For example, your policy could be set up to maximize positional accuracy, but with the capability of reducing accuracy if the device is known to be low on charge, to conserve battery usage.

### About this task

An acquisition policy controls how data is collected from a sensor of a mobile device, using GPS positions and WiFi access points. To manage battery life appropriately, you should match the policy used to your needs. For example, while you might want to have a very accurate position for a geofence trigger, you may be able to save power by using a different policy when the device is far away from the area of interest.

You set up an acquisition policy by using the `WL.Device.startAcquisition` API.

You can specify a preset geo policy to use in the `WL.Device.startAcquisition` API. You do this by using the `WL.Device.Geo.Profiles` API, in which you can specify one of the following functions, based on your requirements:

- `LiveTracking`. Use to get the most accurate and timely position information, but with heavy battery use.
- `RoughTracking`. Use to track devices, but when you do not need the most accurate or timely information. Power usage is less than for `LiveTracking`.
- `PowerSaving`. Use to get infrequent positional data at low accuracy levels, but with very good power conservation.

For information about the preset values for each function, see `WL.Device.Geo.Profiles`.

In addition to these three functions, you can specify many other configuration options as part of the `WL.Device.startAcquisition` API. At the most basic level, you can decide whether you want to allow for GPS use. This option is controlled by the `enableHighAccuracy` parameter. Note that you should set your permissions appropriately if you want to use GPS. For information about permissions, see Location services API. If you decide not to use GPS, then a lower-power and less accurate position provider is used.

When the device for which you are acquiring data is plugged in, you might want to use the `LiveTracking` profile. Then, at different battery levels, switch to other options that save power. You might want similar behavior when the application goes to the background, or resumes. To fulfill these requirements, you can use Apache Cordova, and register for the appropriate event. Apache Cordova events provide you with the ability to monitor battery status, and respond appropriately based on the status. For more information, see the Apache Cordova documentation at <http://cordova.apache.org/docs/en/2.6.0/index.html>, and search for "events".

### Procedure

1. Decide on the requirements for your application policy.
2. Optional: Call the `WL.Device.Geo.Profiles` API, specifying the required function.

3. Optional: Use Apache Cordova to monitor your battery status.
4. Call the `WL.Device.startAcquisition` API.

## Example

In the code, *triggers* is a variable that stores the currently defined triggers, and *failureFunctions* is a variable that stores the functions to be called when acquisition fails.

```
window.addEventListener("batterylow", goToPowerSaveMode, false);

function goToPowerSaveMode() {
  WL.Device.startAcquisition(
    { Geo: WL.Device.Geo.Profiles.PowerSaving() },
    triggers,
    failureFunctions
  );
}
```

## Working with geofences and triggers

You can use geofences and triggers to identify users entering, exiting, or staying inside or outside a geographical area. You can initiate actions, such as improving responsiveness for privileged guests at a hotel chain, based on data relating to the geofence.

### Before you begin

Acquisition of geolocation data must be started before you can receive triggers related to geofences. For more information, see “Setting an acquisition policy” on page 474.

### About this task

A geofence is a geographical area, defined in the form of a circle or polygon. You can increase or decrease the size of the area by changing the value of the **bufferZoneWidth** parameter in the `WL.Device.startAcquisition` API.

Triggers are used to identify users entering, exiting, or staying inside or outside a geofence. For the entering and exiting triggers, the user must have been previously outside or inside the area, including the buffer zone, for the trigger to occur.

Confidence levels are used to help determine whether the trigger condition is met, and can be used to trade off sensitivity, correctness, and battery usage. A confidence level of *low*, which is the default, uses the acquired position and does not take into account the accuracy of the measurement. The *medium* and *high* confidence levels do take accuracy into account. The *medium* confidence level indicates that the system is approximately 70% confident that the condition is met. The *high* confidence level corresponds to a level of approximately 95%.

A *low* confidence level indicates that the condition is met more often, although there is a higher likelihood of it being mistaken. A *high* confidence level indicates that the condition is met less often, however it is less likely to be mistaken.

Note that after an *Enter* trigger has been activated, it will not activate again until the user leaves the “activated” area, which includes the buffer zone. For the *entering* and *dwelling* inside triggers, this means that the user must exit the area.

For the exiting and dwelling outside triggers, this means that the user must enter the area.

## Procedure

1. Start acquiring geolocation data, by using the `WL.Device.startAcquisition` API.
2. Include trigger definitions for geofence triggers: Enter, Exit, DwellInside, and DwellOutside. Set confidence levels for the triggers.
3. Set events to be transmitted when triggers are activated.

## Example

```
function wlCommonInit(){  
  
    /*  
    * Application is started in offline mode as defined by a connectOnStartup property in initOptions  
    * In order to begin communicating with Worklight Server you need to either:  
    *  
    * 1. Change connectOnStartup property in initOptions.js to true.  
    * This will make Worklight framework automatically attempt to connect to Worklight Server as a  
    * Keep in mind - this may increase application start-up time.  
    *  
    * 2. Use WL.Client.connect() API once connectivity to a Worklight Server is required.  
    * This API needs to be called only once, before any other WL.Client methods that communicate w  
    * Don't forget to specify and implement onSuccess and onFailure callback functions for WL.Clien  
    *  
    * WL.Client.connect({  
    *     onSuccess: onConnectSuccess,  
    *     onFailure: onConnectFailure  
    * });  
    *  
    */  
  
    // Common initialization code goes here  
  
}  
  
var triggers = {  
  Geo: {  
    centralPark: {  
      type: "DwellInside",  
      polygon: [  
        {longitude: -73.95824432373092, latitude: 40.80062106285157},  
        {longitude: -73.94948959350631, latitude: 40.79691751000037},  
        {longitude: -73.97309303283704, latitude: 40.764486356929645},  
        {longitude: -73.98167610168441, latitude: 40.76799670467469}  
      ],  
      dwellingTime: 600000, // 10 minutes  
      bufferZoneWidth: -100, // at least 100 meters within the park  
      callback: after10MinsInCentralPark  
    },  
    statueOfLiberty: {  
      type: "Enter",  
      circle: {  
        longitude: -74.044444,  
        latitude: 40.689167,  
        radius: 5000 // 5km  
      },  
      confidenceLevel: "high", // ~95% confidence that you are in the circle  
      eventToTransmit: {  
        event: {  
          nearAttraction: "statue_of_liberty"  
        },  
        transmitImmediately: true  
      }  
    }  
  }  
};
```

```
    }  
  }  
};
```

## Differentiating between indoor areas

You can use visible access points to identify areas in an indoor location such as a shopping mall. After transmitting this data to a server, together with the device context, you can use it for auditing, reporting, and analysis.

### About this task

The process of acquiring data that identifies discrete areas in an indoor location, where the GPS signal might be poor or non-existent, involves acquiring WiFi data, and using WiFi triggers to initiate events.

### Procedure

1. Scan the area to determine which access points are visible from each area that you are interested in, and then record the access points.

To scan the area, create a small application that has the following elements:

- A WiFi acquisition policy for appropriate SSIDs. In the policy, specify MAC: "\*" to see each access point.
  - A data entry function, for specifying the various indoor areas of interest, and submitting the data. This data entry function calls `WL.Client.transmitEvent` to send the location, together with the device context, to the server for logging and subsequent analysis.
2. Analyze the data, and use the analysis to determine which access points are visible in each of the regions.
  3. In the application, use the `accessPointFilters` parameter to define the same visible access points that were used previously.
  4. Define WiFi-fence triggers for each region.

### Example

This example shows the use of two small applications.

The first defines which networks are to be scanned, and lets the user define named regions as the client device moves around the indoor area. For example, when the user enters the food court, they could specify that the region is called "FoodCourt". Upon leaving it, they could either clear the current region, or enter the name of the adjacent region they are entering, such as "MallEntrance5". In order to implement this process, adapter logic is implemented on the server side. It updates the application context with the region information and handles all received events. In this way, all the information is written out to the raw reports database, where each row includes the region name in the APP\_CONTEXT column, and the visible access points under WIFI\_APS.

The data can then be gathered to define triggers to implement the desired application logic. For example, in the triggers that are defined at the end of the example, the two specific access points are identified, which should be visible when the device is in the food court. The example shows the identification of a global trigger for entering the mall; instead, a trigger could have been defined for each of the mall entrances based on the access points visible at each location.

## Application to set up acquisition policy, including triggers

```
function wlCommonInit(){

    /*
    * Application is started in offline mode as defined by a connectOnStartup property in initOptions
    * In order to begin communicating with Worklight Server you must either:
    *
    * 1. Change connectOnStartup property in initOptions.js to true.
    * This will make Worklight framework automatically attempt to connect to Worklight Server as a
    * Keep in mind - this may increase application start-up time.
    *
    * 2. Use WL.Client.connect() API when the connectivity to a Worklight Server is required.
    * This API needs to be called only once, before any other WL.Client methods that communicate w
    * You must specify and implement onSuccess and onFailure callback functions for WL.Client.conn
    *
    * WL.Client.connect({
    *     onSuccess: onConnectSuccess,
    *     onFailure: onConnectFailure
    * });
    */

    // Common initialization code goes here.
}

var SSIDs = [];

function addNetworkToBeScanned(ssid) {
    if (SSIDs.indexOf(ssid) < 0)
        SSIDs.push(ssid);
}

function removeNetwork(ssid) {
    var idx = SSIDs.indexOf(ssid);
    if (idx > 0)
        SSIDs.splice(idx, 1);
}

function startScanning() {
    var filters = [];
    for (var i = 0; i < SSIDs.length; i++) {
        var ssid = SSIDs[i];
        filters.push({SSID: ssid, MAC: "*"});
    }

    var policy = {
        Wifi: {
            interval: 3000,
            accessPointFilters: filters
        }
    };

    var triggers = {
        Wifi: {
            change: {
                type: "VisibleAccessPointsChange",
                eventToTransmit: {
                    event: {
                        name: "moved"
                    }
                }
            }
        }
    };

    var onFailure = {
```



```

        Wifi: onWifiFailure
    };

    WL.Device.startAcquisition(policy, triggers, onFailure);
}

function stopScanning() {
    WL.Device.stopAcquisition();
}

function onWifiFailure(code) {
    // show an error message to the user...
}

// receives a string, indicating the name of the region
function setCurrentRegion(region) {
    WL.Server.invokeProcedure(
        {
            adapter: "HT_WifiScan",
            procedure: "setAppContext",
            parameters: [JSON.stringify({regionName: region})]
        },
        {
            onSuccess: function() {
                // update UI, indicating success
            },
            onFailure: function() {
                // update UI, indicating error
            }
        }
    );
}
}

```

### Adapter logic to update application context and handle events

```

// defined as a procedure:
function setAppContext(context) {
    WL.Server.setApplicationContext(JSON.parse(context));
}

function handleEvent(event) {
    // nothing specific to do, the event device context will be logged to raw reports db in any case
}

// log all events
WL.Server.setEventHandlers([WL.Server.createEventHandler({}, handleEvent)]);

```

### Example of Enter trigger

```

var triggers = {
    Wifi: {
        welcomeToMall: {
            type: "Enter",
            areaAccessPoints: [{SSID: "FreeMallWifi"}]
            callback: showWelcome
        }
        foodCourt: {
            type: "Enter",
            areaAccessPoints: [{SSID: "FreeMallWifi", MAC: "12:34:56:78:9A:BC"}, {SSID: "FreeMallWifi",
            callback: showFoodCoupons
        }
    }
};

```

## Securing server resources based on location

Device context data can tell you whether a user's device is connected to a secure network. If it is not connected, the device context can tell you whether the device is within a desired geofence. This data can be used to restrict access to sensitive information or to prevent running specific program logic. It can also be used to require that additional authentication mechanisms, such as one-time pads, be used.

### About this task

In many environments it is important to ensure that sensitive resources are secure, but can be easily accessed by authorized users who are on site. You can use the `WL.Server.getClientDeviceContext` API to obtain a device context from an authorized user. You can then validate the device context by checking whether a user's device is connected to a secure network, or is within a designated desired geofence.

For example, in a hospital, patient records must be secure and confidential, but must be accessible by authorized personnel such as doctors and nurses.

### Procedure

1. While the acquisition is running, the device context reflects the most up-to-date information regarding the user's location. The user's device context is transparently synchronized to the server, so that `WL.Device.getContext` and `WL.Server.getClientDeviceContext` return the same result.

**Note:** The developer must call `WL.Device.startAcquisition` to benefit from the synchronization and validation. Until the developer calls `WL.Device.startAcquisition`, the result is null.

2. Based on the information in the device context, the adapter logic can check whether the user is connected to a specific network. Additionally, by using the `WL.Geo` functions, the adapter logic can validate whether the user is in a specific, desired geographical location.

### Example

This example performs the following tasks:

1. An attempt is made to verify the location. The device context information is acquired, by using both `Geo` and `WiFi` data. A check is made to ensure that the data is current (acquired within the last 5 minutes), and that the device is within the area that is defined by the `legalPolygon` variable. Time calculations are done by using UTC time.
2. If the location cannot be verified, the message `not in an authorized location` is thrown.
3. If the location is verified, further processing takes place.

```
var legalPolygon = loadFromDB();
var secureNetworks = ['Secure1', 'Secure2'];

function loadFromDB() {
  // invoke Cast Iron or load from a database, etc.
  // for this example: showing a triangle
  return [{longitude: 0, latitude: 1}, {longitude: 1, latitude: 0}, {longitude: -1, latitude: 0}];
}

function verifyLocation() {
  // get the server's copy of the client's device context
  var deviceContext = WL.Server.getClientDeviceContext();
```

```

if (deviceContext == null)
    throw 'acquisition not started';

// is the device connected to a WiFi access point?
if (deviceContext.Wifi && deviceContext.Wifi.connectedAccessPoint) {
// is the connected access point a secure one?
if (secureNetworks.indexOf(deviceContext.Wifi.connectedAccessPoint.SSID) >= 0)
    return;
}

// has a geolocation been acquired?
if (deviceContext.Geo && deviceContext.Geo.coords) {
// verify the information:
var timestamp = deviceContext.Geo.timestamp;
var offset = deviceContext.timezoneOffset;
var utcTime = timestamp + offset;

var now = new Date();
var nowTime = now.getTime() + now.getTimezoneOffset();

if (nowTime - utcTime <= 5*60*1000) { // time is within last 5 minutes
    if (WL.Geo.isInsidePolygon(deviceContext.Geo.coords, legalPolygon))
        return;
    }
}

throw 'not in an authorized location';
}

function aProcedure() {
    verifyLocation();

    // rest of logic:
    // ...
}

```

## Tracking the current location of devices

You can track the location of devices by ensuring that ongoing acquisition of geo-locational data is taking place. When the position of the device changes, a trigger is activated.

### About this task

You acquire geo-locational data from a device by using the `WL.Device.startAcquisition` API. The `PositionChange` trigger is activated if the position of the device changes significantly, and events can then be sent to the server. The server handles these events by setting up an event handler.

For example, a warehouse could improve the efficiency of its processes by using locational data from its delivery vehicles to guide the vehicles to the correct docks, and notify warehouse personnel so that they can be prepared for the arrival of the vehicles.

### Procedure

1. The acquisition of geo-locational data is initiated by the `WL.Device.startAcquisition` API.
2. The `PositionChange` trigger in the API is used to emit events that are then transmitted to the server. For "live" views, either the transmission interval that is set in the `WL.Client.setEventTransmissionPolicy` API should be small, or the `transmitImmediately` parameter must be set to true.

3. An event handler is set up on the server by using the `WL.Server.createEventHandler(filter,handlerFunction)` API. The filter is a literal object that is used to match only the events that you want the handler function to handle.
4. The events that are transmitted to the server contain the client's device context at the time the trigger was activated. The handler can pass this, or other information, to external systems where, for example, the data could be displayed on a map.

## Example

### Adapter code

```
function handleDeviceLocationChange(event) {
    // do something with event
}

function handleDeliveryTruckMoved(event) {
    // do something with event
}

function handleRefrigeratedDeliveryTruckMoved(event) {
    // do something with event
}

var deviceMoveHandler = WL.Server.createEventHandler(
    {},
    handleDeviceLocationChange
);

var deliveryTruckMovedHandler = WL.Server.createEventHandler(
    {vehicle: "DeliveryTruck"},
    handleDeliveryTruckMoved
);

var coolTruckMovedHandler = WL.Server.createEventHandler(
    {
        vehicle: "DeliveryTruck",
        refrigeration: true
    },
    handleRefrigeratedDeliveryTruckMoved
);

WL.Server.setEventHandlers(
    [
        deviceMoveHandler,
        deliveryTruckMovedHandler,
        coolTruckMovedHandler
    ]
);
```

### Mobile application logic

```
function w1CommonInit(){

    // Common initialization code goes here.
    // get truck id (for example from the user) -- for this example, using a hard-coded value.
    var truckId = 123;
    var driverName = "John Smith";

    var policy = {
        Geo: {
            enableHighAccuracy: true,
```

```

        timeout: 10000
    }
};

var triggers = {
    Geo: {
        tracking: {
            type: "PositionChange",
            minChangeDistance: 100, // 100 meters
            eventToTransmit: {
                event: {
                    vehicle: "DeliveryTruck",
                    id: truckId,
                    driverName: driverName
                }
            }
        }
    }
};

WL.Device.startAcquisition(policy, triggers);
}

```

## Keeping the application running in the background

When you are tracking a device by acquiring geolocation data, it is important to keep an application running in the background so that data can continue to be acquired.

### About this task

If you are using Android or iOS, you can keep an application running in the background, even when the device owner is using another application, such as checking email.

The process for each platform is described in the following procedure.

### Procedure

- For Android devices, to ensure that the application will continue to run in the background use `WL.App.setKeepAliveInBackground(true, options)`. Using this API binds the application to a foreground service. By default, if no options are specified, the application's name and icon are displayed. Tapping on the notification takes the user back to the last activity that made the call to `WL.App.setKeepAliveInBackground(true)`. The notification is present until the app exits, or `WL.App.setKeepAliveInBackground(false)` is called. For details on using the options to change the text, the icon, or which activity gets called when the user presses on the notification, see “`WL.App.setKeepAliveInBackground`” on page 554.
- For iOS devices, you must set up your `info.plist` file to indicate that you want to use background location services when `enableHighAccuracy=true`. To do this, you must set the location string on the `UIBackgroundModes` key in the `info.plist` file.

---

## 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.
- 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 620.

### 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 620.

### 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 573.

## 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 381.

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":      "wlap",
         "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
         "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="w|_unprotected" audit="true" />
```

The security test must be `w|_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});
```

## 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 485. 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 620.

**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 620.

### 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 589.

## 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.environment == "android") {  
        return {capture: true};  
    }  
    return {capture: false};  
}
```

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



# Mobile testing for IBM Worklight

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

<b>Testing with IBM Worklight</b> . . . . .	<b>1</b>
Installing the mobile test client . . . . .	5
Software requirements . . . . .	5
Installing the mobile test client on an Android device . . . . .	5
Installing the mobile test client for Android with adb . . . . .	6
Installing the mobile test client on an Android emulator. . . . .	7
Installing the mobile test client on the iOS Simulator . . . . .	8
Configuring the mobile test client . . . . .	9
Configuring the mobile test client for Android . . . . .	9
Configuring the iOS mobile test client . . . . .	10
Creating a mobile test project from the IBM Worklight project creation wizard . . . . .	11
Initiating mobile testing from Android, iPad, and iPhone environments in Worklight Studio . . . . .	12
Using the Application Center and the Mobile Test Workbench to share applications . . . . .	14
Publishing test-ready iOS applications to the Application Center . . . . .	15
Managing mobile applications . . . . .	16
Adding applications in the workbench . . . . .	17
Uploading Android apps from the mobile test client . . . . .	18
Instrumenting Android applications in a shell-sharing environment . . . . .	19
Instrumenting an iOS application . . . . .	19
Installing an instrumented iOS application on mobile devices . . . . .	20
Increasing memory allocation to upload applications . . . . .	21
Creating mobile tests . . . . .	21
Recording tests from the Android mobile test client . . . . .	21

Recording tests from the iOS mobile test client . . . . .	23
Recording tests from the test workbench. . . . .	24
Editing mobile tests . . . . .	24
Creating verification points in a test . . . . .	25
Adding user actions in a test . . . . .	26
Creating application stubs in tests for Android apps. . . . .	26
Defining a variable to run a test with a selected mobile device. . . . .	27
Assigning a test variable to an object's property . . . . .	28
Adding hardware actions in a test. . . . .	29
Splitting a test . . . . .	29
Activating web UI actions . . . . .	30
Actions from the Mobile data view . . . . .	30
Running mobile tests . . . . .	32
Running tests from an Android mobile test client . . . . .	32
Running tests from the iOS mobile test client . . . . .	33
Running tests from the test workbench . . . . .	34
Running a test with different localized strings. . . . .	34
Evaluating results . . . . .	36
Viewing mobile reports . . . . .	37
Managing logs for Android mobile test client . . . . .	38
Compound tests . . . . .	39
Creating a compound test . . . . .	40
Viewing compound tests . . . . .	40
Adding tests into a compound test . . . . .	41
Modifying a compound test . . . . .	42
Running compound tests . . . . .	42
Generating compound test result reports . . . . .	43
Adding a compound test to a Test Workbench project . . . . .	44
Extending Rational Test Workbench Eclipse Client . . . . .	45
Extending test execution with custom code. . . . .	45

<b>Index</b> . . . . .	<b>69</b>
------------------------	-----------

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



---

## Testing with IBM Worklight

The mobile testing capabilities of IBM® Mobile Test Workbench for Worklight® automate the creation, execution, and analysis of functional tests for IBM Worklight native and hybrid applications on Android and iOS devices.

IBM Worklight includes IBM Mobile Test Workbench for Worklight for testing mobile applications. You can only test IBM Worklight applications. To test mobile applications that are not developed with IBM Worklight, consider purchasing IBM Rational® Test Workbench.

You can test Android and iOS applications with IBM Mobile Test Workbench for Worklight. You cannot currently use it to test BlackBerry applications.

The tool chain for preparing and testing mobile applications in IBM Worklight includes:

- Worklight Studio for developing your application, preparing it for test, and uploading it to the mobile test workbench for testing by the developer.
- Application Center for sharing applications when the person who develops the application is different from the person who tests the application.
- IBM Mobile Test Workbench for Worklight for testing the application.

The mobile testing component provided within IBM Worklight is also available in Rational Test Workbench. Therefore, it is sometimes referred to as “Rational Test Workbench” in this documentation, and in particular when it concerns the mobile test client.

**Note:** The mobile test client runs on the mobile device, and the test workbench runs on a Windows or Linux computer. The two work together.

**Note:** The test workbench is a component that can be added to Worklight Studio to test Worklight mobile applications, but it can also be used independently to test mobile applications created outside of IBM Worklight, as well as Selenium, HTTP, SAP, Citrix, and other types of applications, as is the case for the Rational Test Workbench product.

### Installation

To use the test workbench, you must install it as an extra component in Worklight Studio. To know how to install the test workbench within Worklight Studio, see [Installing IBM Mobile Test Workbench for Worklight](#).

**Note:** Testing Android applications with the test workbench requires a JDK. Make sure to also add the path to the JDK in **Window > Preferences > Java > Installed JREs**, and to set it as the default JRE by selecting its corresponding check box.

### Mobile testing tools

The following main components are designed specifically to help you test mobile apps:

- A mobile test client is available on the Android and iOS platforms. This client is used to upload apps to the test workbench, to record, to run tests, and to view reports.
- A test navigator lists test projects, tests, mobile devices, and the mobile incoming recordings that are used to generate tests.
- A device editor lists the devices that are connected to the test workbench. This editor displays detailed specifications of each device, therefore you can select the hardware platforms on which you can deploy and run your tests.
- An application editor lists the managed apps that are uploaded and prepared for testing.
- A test editor enables you to edit test scripts in natural language, and add actions, verification points, data pools, test variables, or stubs in your script steps.
- A mobile data view displays the screen captures that were uploaded from the mobile device during the recording. Use this view to display and select user interface elements, and optionally to add verification points to the test script.

## Stages in the testing process

The goal of mobile testing is to ensure that your mobile application meets the requirements that guided its design and development. To help you meet this goal, IBM Mobile Test Workbench for Worklight implements the following stages in the testing process:

- **Configuration:** Set up your test environment with IBM Mobile Test Workbench for Worklight and the SDKs for the mobile operating systems. Install the mobile test client on one or several mobile devices. Ensure that the mobile devices have connectivity through WiFi, 3G, or 4G, and add those devices to the test workbench.
- **Application preparation:** Import the application that you want to test into the test workbench, or use the device to upload the application under test to the test workbench.
- **Test recording:** Run the app from the mobile test client to start a recording. The recorder records all user interactions, sensor inputs, and application behavior, and uploads the recorded data to the test workbench, where it can be converted into a mobile test.
- **Test editing:** After recording, you can edit the test in the natural language editor. You can use the mobile data view to display and select UI elements from the recorded applications. You can replace recorded test values with variable test data, or add dynamic data to the test.
- **Testing:** You can deploy and run automated tests on multiple devices to ensure that the app matches the expected behavior that is defined in *verification points*. During the run, each verification point is checked and receives a *pass*, *fail*, or *inconclusive* status and functional data is recorded.
- **Evaluation of results:** After the test, the device uploads the test data to the test workbench. You evaluate the test results through the performance and verification point reports that are generated with the uploaded data. You can also design custom reports by manipulating a wide range of counters.

Functional reports provide a comprehensive view of the behavior of the app under test. Reports can be exported and archived for validation.

When the tester is the same person as the developer, this person can develop and test the application in the same Eclipse environment.

When the person who develops the application is different from the person who tests it, the application must be shared between the developer and the tester by using the Application Center. In this case, the testing process includes the following extra stages:

- **Publication:** You can publish Android applications to the Application Center from Worklight Studio by right-clicking an Android project and clicking **IBM Application Center > Publish on IBM Application Center**. For iOS, the application must first be instrumented for testing. You right-click an iOS project, and click **IBM Application Center > Publish Test-Ready Application**. The iOS application is then instrumented and published.
- **Import of the application:** When the application is published in the Application Center, the tester imports this app into the list of managed applications into the test workbench.

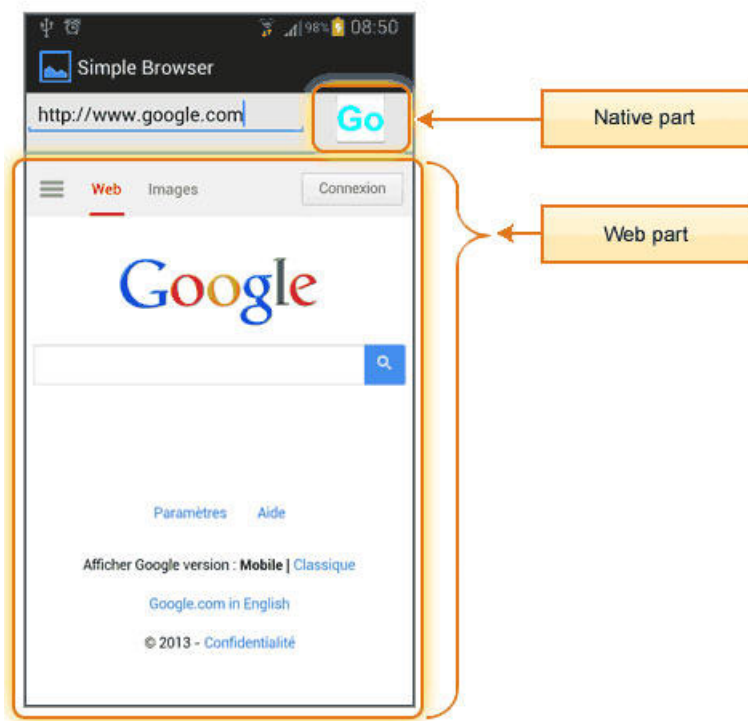
## Support for testing native and hybrid applications

You can use IBM Mobile Test Workbench for Worklight to test both native and hybrid applications that were created with Worklight Studio.

A *native* Android or iOS application is built using a native SDK, whose services are defined according to each platform architecture. Android applications are typically created with Java™ or C++, whereas iOS applications are created with Objective-C.

A *hybrid* application is an application that combines native and web technologies. The web part relies on HTML 5, CSS3, and javascript.

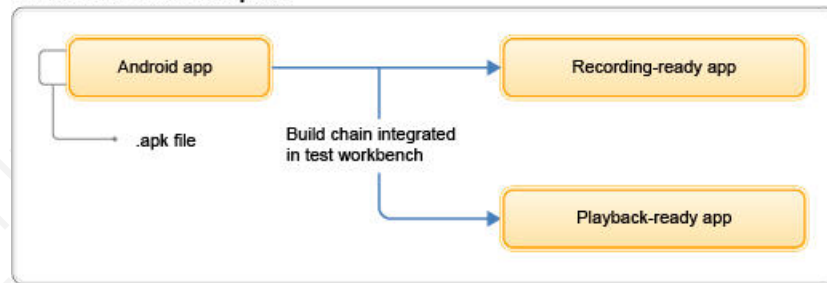
**Note:** To test applications that are not created with IBM Worklight, you must use IBM Rational Test Workbench.



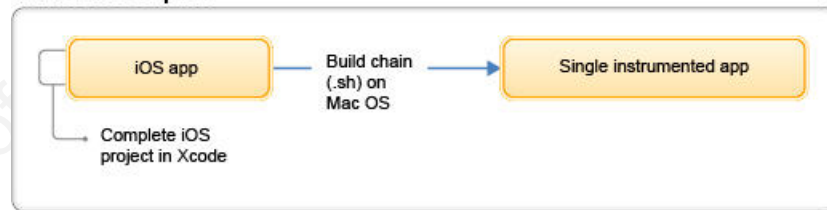
## Differences between Android and iOS testing

The following figure shows the difference between Android and iOS testing.

### Windows or Linux computer



### Macintosh computer



Before you can test an Android or iOS application, the application must first be *instrumented*. An instrumented application contains the application under test augmented with code that allows you to record or play back a test.

When you test an Android application, the Android application (the .apk file) is recompiled into a *recording-ready app* that is heavily instrumented to capture user

actions. In addition, a *playback-ready app* is created. This playback-ready app is a version of the original user application that is signed with a test workbench certificate.

**Note:** There is also a third app: the Tester app. This app contains the runtime code that is needed to replay a test. This app is not noticeable if you run in silent mode. When the application under test is modified, only the recording-ready app and the playback-ready app are generated.

The mobile test client handles the installation and uninstallation of these apps to ensure that you have the correct version of the app for the task.

For Android, the build and compile process takes place in the test workbench on a Windows or Linux computer.

For iOS, the process is different because an iOS application is a complete iOS project in Xcode. The build and compile process to instrument the application takes place entirely in Xcode on a Macintosh computer. One single application is created for both recording and playback. The resulting instrumented application is then added to the Test Workbench.

---

## Installing the mobile test client

This section contains instructions for installing the IBM Rational Test Workbench Mobile Client.

### Software requirements

Before you install the mobile test client, verify that your device meets the software requirements.

#### IBM Mobile Test Workbench for Worklight operating system

- Microsoft Windows version 7, Vista, and 8
- Red Hat Linux version 5 and 6 or SuSE Linux version 11

#### Devices operating system

- Android 2.2 - 4.2
- iOS 6 or later

#### Additional requirements

- Android
  - Android SDK 21 installed
- iOS
  - Mac OS X v10.8 Mountain Lion or later
  - Xcode 4.6
  - iOS Simulator 6
  - Apple Developer or Enterprise License

## Installing the mobile test client on an Android device

To record and run tests from your Android mobile device, you must install the Android Rational Test Workbench Mobile Client on the device. This topic describes

how to download the installer in order to install the client on the device. Refer to other installation topics if you are running Android on a virtual machine or emulator.

## Before you begin

The mobile device must support Android version 2.2 or later.

The mobile device must have a working internet connection (wifi, 3G, or 4G). Also, the mobile device must be on the same network as the computer that is running IBM Mobile Test Workbench for Worklight and be able to ping that computer. You can use an app that tests the connection to a computer.

The device must be allowed to install apps from any sources. To enable this option on the device, depending on the version of Android, tap **Settings > Applications > Unknown sources** or **Settings > Security > Unknown sources**. Tap this option. If the option is not available on your device, or if the following method is not possible, see “Installing the mobile test client for Android with adb.”

## Procedure

To install the mobile test client on the device:

1. In IBM Mobile Test Workbench for Worklight, click **File > New > Device**. A window displays the URL of the workbench and a QR code that contains the URL.
2. Depending on the mobile device, perform one of the following steps:
  - If the mobile device is equipped with a camera and a QR code reading app, then run the QR code app and flash the QR code that is displayed on the workbench screen.
  - If the mobile device cannot flash a QR code, open the web browser and go to the URL that is displayed on the workbench screen.

The mobile device displays a web page with the download link for the mobile test client.

3. In the mobile web browser, tap the link to download the mobile test client installer. After the download completes, tap the complete button.
4. If the mobile test client does not install automatically, use a file manager app to locate the APK installer file (usually in the Downloads folder) and run the installer manually.

## What to do next

After installing the mobile test client, configure and connect the device to IBM Mobile Test Workbench for Worklight. See “Configuring the mobile test client for Android” on page 9.

### Related concepts:

Mobile testing overview

### Related tasks:

Getting started with Android testing

## Installing the mobile test client for Android with adb

On some mobile devices, it might not be possible to install the Android IBM Rational Test Workbench Mobile Client by downloading the installer. This topic

describes an alternative installation method that uses a USB connection and the adb tool that is provided with the Android SDK.

## About this task

For the typical method of installing the mobile test client, see “Installing the mobile test client on an Android device” on page 5.

## Procedure

To install the mobile test client with adb.

1. Enable USB debugging on the mobile device:
  - On Android 3.2 and earlier versions, tap **Settings** > **Applications** > **Development** and select **USB Debugging**.
  - On Android 4.0 and 4.1, tap **Settings** > **Developer options** and select **USB Debugging**.
  - On Android 4.2 and later, tap **Settings** > **About phone** and tap **Build number** seven times. Then, return to the previous screen, tap **Developer options** and select **USB Debugging**.
2. Connect the Android device with a USB cable to the computer that is running the IBM Mobile Test Workbench for Worklight. Some mobile devices might require a specific USB driver. If necessary, go to the web site of the device vendor to download and install the USB driver for the device.
3. In the test workbench, click **File** > **New** > **Device**. The **Add New Device** wizard shows the Workbench URL.
4. Open a web browser, type the Workbench URL, and download the mobile test client.
5. Copy `com.ibm.rational.test.mobile.android.client.ui-release.apk` to a temporary folder. For example, `C:\tmp`.
6. Open a command prompt window and point to the Android SDK directory that typically is `C:\Users\Administrator\AppData\Local\Android\android-sdk\platform-tools`.
7. Type the following commands:

```
adb connect <IP address of workbench computer>
adb install <temporary folder>/
com.ibm.rational.test.mobile.android.client.ui-release.apk
```

## What to do next

After installing the mobile test client, configure and connect the device to IBM Mobile Test Workbench for Worklight. See “Configuring the mobile test client for Android” on page 9.

### Related concepts:

Mobile testing overview

### Related tasks:

Getting started with Android testing

## Installing the mobile test client on an Android emulator

If the Android device is running on a virtual machine or on the emulator provided by the Android SDK, you can install the IBM Rational Test Workbench Mobile Client either by downloading it or using the adb tool.



## About this task

For the typical method of installing the mobile test client, see “Installing the mobile test client on an Android device” on page 5.

## Procedure

To install the mobile test client on an emulator:

1. In IBM Mobile Test Workbench for Worklight, click **File > New > Device**. The **Add New Device** wizard shows the Workbench URL.
2. Complete one of the following steps:
  - a. Download the mobile test client on the emulator.
    - 1) In the emulator, open a web browser and enter the Workbench URL.
    - 2) Click the link to download the mobile test client.
    - 3) On the apps page of the emulator, click **Downloads**, click `com.ibm.rational.test.mobile.android.client.ui-release.apk`, and then click **Install**.
  - b. Use the adb tool to install mobile test client.
    - 1) On the computer, open a web browser and enter the Workbench URL.
    - 2) Click the link to download the mobile test client.
    - 3) Copy `com.ibm.rational.test.mobile.android.client.ui-release.apk` to a temporary folder. For example, copy the file to `C:\tmp`.
    - 4) Open a command line window and point to the Android SDK directory. The default path to the directory is `C:\Users\admin_name\AppData\Local\Android\android-sdk\platform-tools`
    - 5) Type the following command:

```
adb -e install temporary_folder\  
com.ibm.rational.test.mobile.android.client.ui-release.apk
```
3. To install on a virtual machine, open a command line window and type the following commands:

```
cd <temporary folder>  
adb connect <IP address of the virtual machine>  
adb -r install com.ibm.rational.test.mobile.android.client.ui-  
release.apk
```

## What to do next

After installing the mobile test client, configure and connect the device to IBM Mobile Test Workbench for Worklight. See “Configuring the mobile test client for Android” on page 9.

### Related concepts:

Mobile testing overview

### Related tasks:

Getting started with Android testing

## Installing the mobile test client on the iOS Simulator

To record and run tests from your iOS Simulator, you must install the IBM Rational Test Workbench Mobile Client on the simulator.

## Before you begin

You should have a Mac OS X v10.8 Mountain Lion or later with Xcode 4.6. For supported versions, see the Software requirements topic.

**Note:** Before using the shell scripts, ensure that you have all the execution rights required.

## Procedure

1. In Eclipse client IBM Mobile Test Workbench for Worklight, click **File > New > Add Device** and copy the Workbench URL.
2. In the MacOS 10 system, open the web browser, enter the Workbench URL, and click the download link. This task downloads the RTW-iOS-Build-Archive.zip.
3. Before using the shell-scripts, unpack the archive and ensure that all shell scripts in the build-script folder can be executed by the current user. This can be checked with **ls -l** command and changed with **chmod** command. Refer to Mac OS X documentation for more details on how to use these commands.
4. Open the terminal, point to the build-script folder and pass the following command:  

```
./installIPAINSimu.sh /Users/XXX/Downloads/RTW-iOS-Build-Archive/client/iphonesimulator/RTWiOS.ipa
```

 . The .ipa file parameter is required, it is located in the relative path to the shell script ../client/iphonesimulator/RTWiOS.ipa. The iOS mobile test client is installed on the simulator and the mobile test client icon shows up on the apps screen of the simulator.

## What to do next

You must now connect the iOS mobile test client with the test workbench. For information about connecting to the test workbench, see Configuring test workbench topic.

### Related concepts:

Mobile testing overview

### Related tasks:

Getting started with iOS testing

---

## Configuring the mobile test client

This section contains instructions for configuring the IBM Rational Test Workbench Mobile Client.

You must add and configure devices to IBM Mobile Test Workbench for Worklight from the Mobile Devices editor.

## Configuring the mobile test client for Android

To use an Android mobile device for uploading mobile apps and recording or running tests, you must configure the Rational Test Workbench Mobile Client to connect to .

## Before you begin

The mobile test client must be installed and running on the Android device. For details about downloading and installing the mobile test client for Android, see “Installing the mobile test client on an Android device” on page 5.

The mobile device must have a working internet connection (wifi, 3G, or 4G). Also, the mobile device must be on the same network as the computer that is running IBM Mobile Test Workbench for Worklight and be able to ping that computer. You can use an app that tests the connection to a computer.

## Procedure

To add a mobile device to the test workbench:

1. In the Test Navigator view, right-click the Mobile devices node and select **Available Mobile Devices** or click the **Display available mobile devices** icon. The **Mobile Devices** editor opens.
2. In the editor, click the icon to add a device to the list. A window displays the URL of the workbench and a QR code that contains the URL.
3. On the mobile device or on an emulator, start the mobile client.
4. Complete one of the following steps:
  - a. If the mobile device is equipped with a rear camera, tap the **QR code** button and scan the QR code that is displayed on the test workbench. This step is not applicable to emulator.
  - b. Tap **Set Workbench > Address** and type the URL manually. This step is triggered automatically if no rear camera is detected.

On the first client start, you do not need to tap on the QR code button, nor typing the workbench address. These actions are automatically executed.

## Results

The name and properties of the device are displayed in the test workbench, in the **Mobile Devices** editor.

## Configuring the iOS mobile test client

To upload the recording and run tests from the iOS IBM Rational Test Workbench Mobile Client to IBM Mobile Test Workbench for Worklight, you must connect them both.

### Before you begin

You must have installed the iOS mobile test client and test workbench Eclipse client and both must be in the same network.

### About this task

If the test workbench is connected to the iOS mobile test client, after recording the app in the device, the recording is automatically uploaded to the test workbench. You can then generate and edit the test in the test workbench.

## Procedure

1. In the iOS mobile test client, tap **Configure Workbench > Address**.
2. Type the Workbench URL and tap **OK**.

**Note:** To get the workbench URL, in test workbench Eclipse client, click **File > New > Device**.

#### Related tasks:

“Installing the mobile test client on the iOS Simulator” on page 8

“Instrumenting an iOS application” on page 19

“Recording tests from the iOS mobile test client” on page 23

---

## Creating a mobile test project from the IBM Worklight project creation wizard

The test scripts that you create for your mobile applications are located in a test project.

### About this task

You can easily create a test project by selecting **File > New > Test Workbench Project** within Eclipse as described in [Creating a test project](#), or you can create a test project at the same time as you create your IBM Worklight project.

To create a test project while you create your IBM Worklight project, complete the following procedure.

**Note:** The IBM Mobile Test Workbench for Worklight allows testing only the mobile applications that are created with IBM Worklight. To test applications that are not created with IBM Worklight, or if you need more tools for extra testing scenarios, you must use the IBM Rational Test Workbench project.

### Procedure

1. In Worklight Studio, select **File > New > Worklight Project** and follow the steps to create an IBM Worklight project (see [Creating IBM Worklight projects](#)).
2. On the last page of the project creation wizard, click **IBM Mobile Test Workbench**, select **Create a Test Project** and enter the name of the test project.

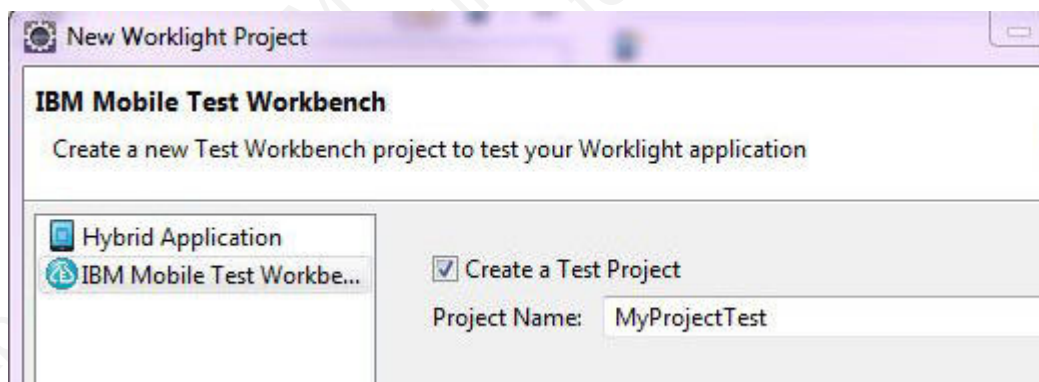


Figure 1. Creating a Test Project from the wizard

3. Click **Finish**.

### What to do next

To start the testing, you must also create your Android or iOS application.

---

## Initiating mobile testing from Android, iPad, and iPhone environments in Worklight Studio

With Worklight Studio, you can easily add iOS or Android applications to IBM Mobile Test Workbench for Worklight, and make them available for the recording and playback of test scripts.

### Before you begin

- For iPad or iPhone applications, you must have an Xcode project created with the IBM Worklight build and deployment action in your environment.
- For Android applications, the APK for your application must already be compiled.

### About this task

If you developed an iOS or an Android hybrid application with Worklight Studio, you can add it to the test workbench in either of these two ways:

- By following the instructions from the section “Adding applications in the workbench” on page 17.
- Or, more easily, by completing the following steps.

**Note:** The following procedure applies only to IBM Worklight hybrid applications. To test native applications that you created with an IBM Worklight native project, you can also use the test workbench, but you must follow the steps that are described in “Adding applications in the workbench” on page 17.

### Procedure

1. Right-click the iPad, iPhone, or Android environment of your IBM Worklight application.
2. Click **Run As > Test with IBM Mobile Test Workbench**.

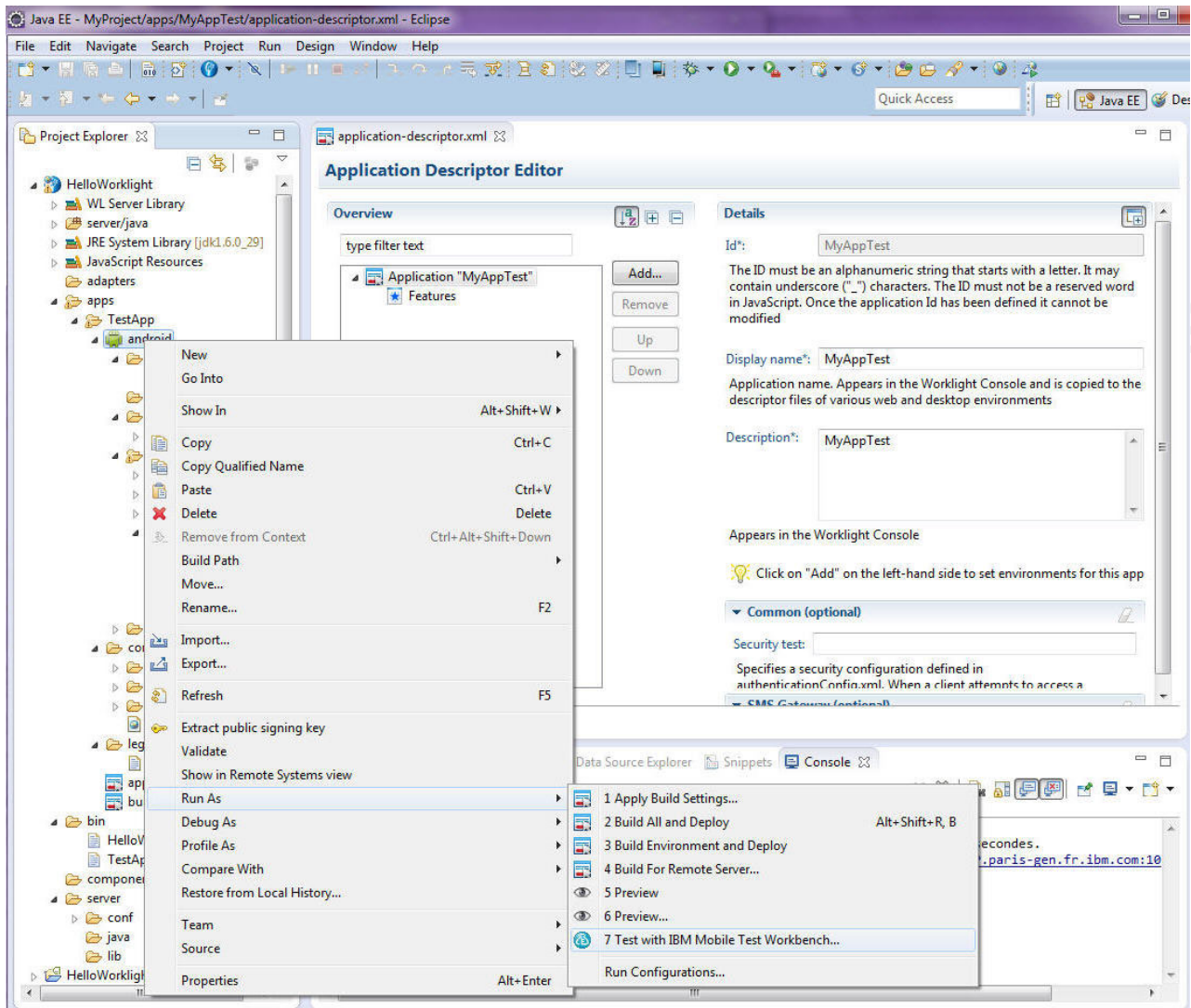


Figure 2. Testing with IBM Mobile Test Workbench from Worklight Studio

- For Android applications, this action places the APK in the test workbench. The application is ready for testing.
- iOS applications must be first instrumented for testing. For iOS environments, the application is first instrumented and the resulting instrumented application is then added to the test workbench. This operation is only applicable on Mac OS.

**Note:** On iOS, this action performs the necessary instrumentation, as an alternative to manually instrumenting your application, by using the provided script as described in “Instrumenting an iOS application” on page 19.

**Note:** You can also use the Application Center to share applications among team members. To know how to share applications between developers and testers, see “Using the Application Center and the Mobile Test Workbench to share applications” on page 14.



---

## Using the Application Center and the Mobile Test Workbench to share applications

Share applications ready for testing with Worklight Studio. Simplify communication between development and test teams by using the Application Center in conjunction with IBM Mobile Test Workbench for Worklight.

In the mobile application development lifecycle, the development and testing of a mobile application can be done by the same person or by different teams. When the person who develops the application is different from the person who tests the application, the application must be shared between the developer and the tester. The Application Center and IBM Mobile Test Workbench for Worklight can simplify the communication between the development team and the test team.

The Application Center is a private application store that can be used to streamline the distribution of applications among an extended development team. The Application Center is similar to public application stores such as Google Play or Apple App Store, but you use it to distribute applications within an enterprise.

With the Application Center, you can create a catalog of mobile applications. Every authorized user can then install mobile applications on their mobile device through the Application Center client.

Worklight Studio and IBM Mobile Test Workbench for Worklight provide an easy way to share applications for test purposes. A developer can upload an Android or iOS application to the Application Center to make this application available to all the members of the test team.

### Sharing Android applications for testing

To share an Android application for mobile testing through the Application Center, you must first prepare your application by building the APK file from the IBM Worklight Android environment in your application. See [Publishing Worklight applications to the Application Center](#) for more information.

To publish the APK file to the Application Center, choose **IBM Application Center > Publish on IBM Application Center**.

No specific instrumentation is required for Android applications. You can use any Android application that is available in the Application Center for testing.

### Sharing iOS applications for testing

Testing iOS applications requires that each application is instrumented before you can record tests on it with IBM Mobile Test Workbench for Worklight. Applications must be instrumented before they are published in the Application Center catalog.

To instrument and publish an iOS application to the Application Center in a single operation, choose **IBM Application Center > Publish test-ready application**. This operation is only available when you run Worklight Studio on MacOS.

The instrumentation of the application uses the Xcode project to do the instrumentation. See [Publishing test-ready iOS applications to the Application Center](#) for more information.



Alternatively, you can use a script to instrument an iOS application. See Command line to launch the `rtwBuildXcode.sh` script for the command line and parameters to use to instrument an iOS application manually. This script produces an archive file that you can manually upload to the Application Center console.

An instrumented iOS application appears with a special test icon in the Application Center catalog. The icon for IBM Mobile Test Workbench for Worklight overlays the application icon.

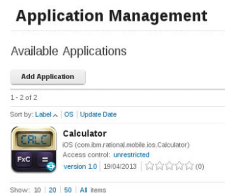


Figure 3. Instrumented application in the Application Center console

## Importing applications into the mobile test workbench

When an application is published for test purposes in the Application Center, a tester can import the application into the list of managed applications in IBM Mobile Test Workbench for Worklight.

In the Perspectives toolbar of IBM Mobile Test Workbench for Worklight, click this icon  to open the editor for mobile applications. In this editor, you can browse the applications available for testing in the Application Center. You can select the applications that you want to import into IBM Mobile Test Workbench for Worklight. See Adding applications in the workbench for more information.

---

## Publishing test-ready iOS applications to the Application Center

Deploy iOS applications that are ready to be tested with the Mobile Test Workbench to the Application Center directly from the Worklight Studio IDE.

### About this task

Worklight Studio provides an easy way to publish test-ready iOS applications to the Application Center. In Worklight Studio, you can instrument a Worklight application for mobile testing and publish it to the Application Center. When an application is available in the Application Center, a member of another team can easily import it into the Mobile Test Workbench for testing.

### Procedure

1. Specify the publication preferences for the Application Center.
  - a. In the main menu, click **Window > Preferences**.
  - b. In the tree on the left, expand IBM Application Center and select **Publish Preferences**.
  - c. Enter the user credentials and server URL for publishing a Worklight application to the Application Center  
See this table for a description of the required publication preferences.

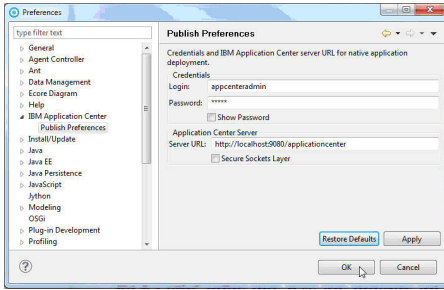


Figure 4. User credentials and server URL for deploying applications to the Application Center

Table 1. Publication preferences for deploying an application to the Application Center

Preference	Description
Credentials	Login name and password for accessing the repository of the Application Center.
Server URL	URL of the Application Center server to use for publishing applications.

2. Publish an iOS application to the Application Center.
  - a. Right-click the iPad or iPhone environment of the IBM Worklight project, or the Xcode project directory, and select **IBM Application Center > Publish Test-Ready Application**. The instrumentation of the project starts.
  - b. When instrumentation is complete, click **Publish** to publish the application with the current preferences or click **Preferences** to change any of the preferences before publishing.

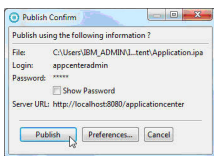


Figure 5. Options to publish the application or change the publication preferences

If the application already exists, publication will fail.

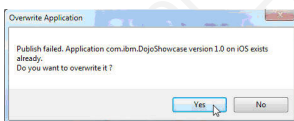


Figure 6. Failed publication of an existing published application

- c. **Option for existing published applications:** Select **Yes** to overwrite the existing version of the application and to publish the new version.

## Managing mobile applications

Before being able to record a test from an Android or iOS application, you must use a mobile device to upload the application that you want to test or import the application to the test workbench. The application is instrumented to make possible the recording and testing processes. You can see the list of the applications available in the workspace and the information that is related to the mobile applications in the **Mobile application** editor.

With the Mobile application editor, you can manage your applications in IBM Mobile Test Workbench for Worklight. There are two main areas in the Mobile application editor view. The area on the left displays the list of Android and iOS apps uploaded in the test workbench or uploaded from a mobile device or an emulator. The area on the right, displays the information on the selected app(s): name, description, version, API level, state, available packages (for Android apps only), resource path, test suites and localized strings. For iOS apps only, you can see the date and time when the application was created. An application with a status of available can be tested.

From the editor, you can add apps in the workbench from different locations or generate managed apps, replace the current application by another one in a set of test suites, or check an option which identifies the application as a web UI enabled app (hybrid app).

## Adding applications in the workbench

To test the mobile application files that are stored on a computer or on a mobile device, you can add them to IBM Mobile Test Workbench for Worklight.

### Before you begin

The Android SDK must be installed on the same computer that Eclipse Client IBM Mobile Test Workbench for Worklight is installed on. You must set the preferences in the mobile application builder path so that they point to the directory where the Android SDK is installed. For more details, see Mobile application builders. To add an application from a mobile device or an emulator, ensure that the device is connected to the test workbench. For more information, see Configuring the mobile client.

### Procedure

1. In the Test navigator view, right-click on the Mobile incoming applications node and then click **Available mobile applications**. Or, in the Test workbench perspective's toolbar, click the **Display available mobile applications** icon . The Mobile applications editor opens.
2. In the editor, click the **Add applications to list** icon to add an application to the test workbench.
3. Complete one of the following tasks in the Add application window:
  - a. To add an application from your local computer, in **from local storage**, click the Android app icon or the iOS app icon to browse for the application.
  - b. To add an application from a workspace, click **from workspace**, click the Android or iOS icons and select the application.
  - c. To add an application from a mobile device, click **from mobile device**, click **Next** and follow the steps in "Uploading Android apps from the mobile test client" on page 18.
  - d. To add an application from a web site, click **from web site** and type the web site URL or click the button to paste a copied URL.
  - e. To add a resource that contains an original mobile application package, click **From existing managed App resource (.ma)**. Click the .ma file to regenerate a managed application. Note that the window displays only application files that are not imported.
4. Optional: In **Application description**, type a brief description of the application that you are adding.
5. Click **Next** and select a project.

6. Click **Finish**. Based on the size of the application, Rational Test Workbench might take some time to prepare the application.

## What to do next

You can now record the application from the mobile device. In the mobile test client, go to **Managed Apps**, tap the application that you added in the test workbench, and tap **Record**.

### Related concepts:

Mobile testing overview

### Related tasks:

Getting started with Android testing

Getting started with iOS testing

## Uploading Android apps from the mobile test client

To test mobile apps, you must import them or upload them to IBM Mobile Test Workbench for Worklight where they are instrumented and recompiled into two new apps: a recording-ready application and a playback-ready application. The recording version contains the application under test, augmented with code and the playback version is the original version with a test workbench certificate. They allow you to either record a test or run a test.

### Before you begin

To upload apps from a mobile device, the mobile device must be running the mobile test client and be connected to test workbench. For more information on configuring the mobile device, see [Configuring the mobile client](#).

The Android SDK must be installed on the workbench computer. You must install the Android SDK mentioned in the **Download For Other Platforms > SDK Tools Only** section of <http://developer.android.com/sdk/index.html>.

**Note:** This web site is not maintained by IBM and the location of SDK might change in the future.

The app must be installed on the mobile device.

### About this task

When you upload your application from a device or simulator and it is being instrumented, depending on the size of the application, this might take a few seconds to several minutes. If you are testing complex applications, in some cases, you might receive an out-of-memory error. The solution could be to increase the memory allocation on the computer where is installed. For more details, see [Increasing memory allocation to upload applications](#)

### Procedure

To upload an app from a mobile device:

1. In the mobile test client, tap **Upload app**.
2. Select an installed app from the list and tap **Upload**. The Mobile application editor opens in Rational Test Workbench and displays the app with a **Preparing** tag until the app is fully uploaded and instrumented for testing.

## What to do next

When the preparation is completed, you can download the instrumented application (recorder app) to a device or simulator to record a test. For more information, see “Recording tests from the Android mobile test client” on page 21

### Related tasks:

Getting started with Android testing

## Instrumenting Android applications in a shell-sharing environment

If you are working in an environment in which IBM Mobile Test Workbench for Worklight & Android Development Toolkit (ADT) are shell-shared, you can automatically import Android applications in the test workbench, launch the instrumentation of the applications, and make them ready for recording and running tests.

### Before you begin

You must modify the configuration file in your Eclipse installation directory and replace your Java virtual machine (JVM) with Oracle Java Development Kit (JDK) to be able to run the Android Development Toolkit. For more information, see the `eclipse.ini` web page. In addition, you must have compiled the Android application package file (APK) for your application before starting the task.

### About this task

You can add Android native or hybrid applications to the test workbench by following the instructions in Adding applications in the workbench, or by completing the following steps.

### Procedure

1. In the Navigator view of the test workbench, right-click on an Android project or in the bin directory and click **Run As > Test with IBM Mobile Test Workbench for Worklight**.
2. In the wizard that opens, enter or select the name of the folder that will embed your application and the name of the APK file. Click **Finish**.
3. The application is instrumented and placed in the test workbench. Now, it is ready for testing. For more information, see Recording tests from the Android mobile test client

## Instrumenting an iOS application

Once the IBM Rational Test Workbench Mobile Client is installed on your simulator, you must run a supplied build script on a MacOS 10 system to be able to instrument the iOS application under test. The instrumented app can be optionally pushed to the Eclipse client IBM Mobile Test Workbench for Worklight and installed on the iOS simulator.

### Before you begin

You must have Xcode installed on your Mac OS 10.8 Mountain Lion, and an Xcode project created. The project must contain the source code of the application to be

tested. Moreover, the `rtwBuildXcode.sh` script must be locally downloaded. It can be found in the build-script directory.

## Procedure

1. In Eclipse client IBM Mobile Test Workbench for Worklight, click **File > New > Add Device** and copy the Workbench URL.
2. Click the Proceed with iOS instructions link.
3. On the Mac OS 10.8 Mountain Lion, open the web browser, enter the Workbench URL, and click the download link. This task downloads the `RTW-iOS-Build-Archive.zip`. Extract the zip file that contains the build-script folder with the `rtwBuildXcode.sh` shell script file.
4. Open the Macintosh terminal, point to the build-script folder, and enter the required command line to launch the `rtwBuildXcode.sh` script with the appropriate parameters. For details, see the Command line to launch the `rtwBuildXcode.sh` script topic. The script is used to automatically run compilations and links in Xcode from an Xcode project file and create an instrumented application.
5. Run the script, this results in an instrumented application, and it is optionally pushed to the test workbench and to the iOS Simulator if you have indicated the required parameters on the command line. Otherwise, an instrumented `.zip` file is generated locally when the script is launched. In that case, you must manually import the instrumented file into the test workbench using the **add application** menu in the Mobile Application editor. Then, you must run another script to install the app into the iOS Simulator. Alternatively, you can push the application to an iOS device. To do so, see Installing an instrumented iOS application on devices.

## Results

The incoming mobile application is displayed in the test workbench Eclipse client, in the navigation view. You must save the incoming application file in a Test Workbench project. Then, when the instrumented application is pushed to a simulator or to a mobile device, you can start recording a test.

## Installing an instrumented iOS application on mobile devices

You can install an instrumented iOS application on mobile devices that run iOS applications, including iPhone, iPad, and iPod Touch.

### Before you begin

To be able to push an instrumented iOS application to an iOS device, you must make the following preparations:

- Have an instrumented iOS application in the test workbench. For more details, see Instrumenting an iOS application.
- iOS mobile test client must be connected to the test workbench. For information about this procedure, see Configuring the iOS IBM Rational Test Workbench Mobile Client.

### Procedure

You are prompted to install the application under test when you launch a recording or playback on your device.

1. In the mobile test client, tap **Manage applications**.



2. Tap the application you want to use for your test.
3. Tap **Record** if you want to record a test or tap **Playback** if you want to run a test, then you will be redirected to the installation steps. A web page opens in the web browser on the device.
4. Click the link to install the instrumented application.
5. At the end of the installation, the instrumented application is installed on the device, ready for recording and testing. You must go back to the page that contains the mobile client icon and tap **IBM RTW** to find the **Record** icon that you will use to record a test. For more information, see Recording tests from the iOS mobile test client.

## Increasing memory allocation to upload applications

Uploading and instrumenting an application from a device or simulator to the test workbench requires memory. If you are testing complex applications, in some cases, the instrumentation might fail and you might receive an out-of-memory error. The solution is to increase the available memory that is allocated to Eclipse Java processes for the computer on which the test workbench is installed.

### Procedure

To increase the memory allocation:

1. Edit the configuration file `eclipse.ini` located in your Eclipse installation directory on which the is installed. Find the line where the available amount of memory is specified. It is defined with the `-Xmxnnnm` parameter, where `-Xmx` is the command name, and `nnn` is the amount of memory, in megabytes.
2. Replace the default value by another one, note that you can change a value that is specified in megabytes by a value in gigabytes: `-Xmx<memory size in Mo>m` or `-Xmx<memory size in Go>g`. On a Windows 32-bit system, the maximum value is 4 gigabytes. Example: `-Xmx4000m` or `-Xmx4g`.
3. Save the `eclipse.ini` file, relaunch the test workbench and try to upload your application again.

---

## Creating mobile tests

You can record a test from an Android or iOS application on a mobile device, a simulator on which the mobile client is installed, or from the test workbench.

## Recording tests from the Android mobile test client

Mobile tests are typically created by recording a session on the mobile device that runs the app being tested. At the end of the recording session, you close the app and the IBM Rational Test Workbench Mobile Client uploads the recorded session to the IBM Mobile Test Workbench for Worklight where it is used to generate a test.

### Before you begin

To record tests on a mobile device, the mobile device must be running the mobile test client and be connected to the test workbench. For more information on configuring the mobile device for Android, see Configuring the IBM Rational Test Workbench Mobile Client.



You must have either added the Android apps to the test workbench directly or uploaded the app from the mobile test client to the test workbench. For information about adding apps to the test workbench, see “Adding applications in the workbench” on page 17. For information about uploading apps to the test workbench, see “Uploading Android apps from the mobile test client” on page 18.

## Procedure

To record a session on the mobile device:

1. In the mobile test client, tap **Managed apps**.
2. Select an app from the list and tap **Record**.

**Note:** If your device or emulator does not have silent mode, the mobile test client uninstalls the installed version of the application under test and installs it again. During this process, tap the **Uninstall**, **OK**, and **Install** buttons accordingly. If your device or emulator has silent mode, this process happens in the background.

To make the silent mode option available on a device, you must connect the device with a computer that has the Android SDK installed. Use an USB cable and enable USB debugging. Ensure that you installed the appropriate USB driver. Next, open the command prompt on the computer and run the following commands:

- `adb devices`: Lists the devices connected to the computer by the USB cable.
- `adb tcpip 5555`: Makes the silent mode option available on the device.

You must follow these steps every time you reboot your device.

Silent mode is not available on devices and emulators with API level 17 (Android 4.2 to 4.2.2), due to a known limitation

3. When the app starts, interact with the device. All your actions on the device and responses from the app are recorded.
4. To end the recording, close the app, switch to another app, or tap the Home button. The recording is uploaded to the test workbench. Depending on the size of the recording, the upload might take a few seconds to several minutes. Recordings are displayed in the test navigator under **Mobile Incoming Recordings** with a name and a timestamp.

**Note:** If your session involves switching between apps, including multiple apps, a new recording is uploaded each time you switch apps. This action produces multiple recording logs in the **Mobile Incoming Recordings** folder. You can combine these multiple recordings to generate a single test.

5. In the test workbench Test Navigator, expand **Mobile Incoming Recordings**, right-click a recording and select **Generate Test**. The New Test from Incoming Recordings window opens.
6. Select a project folder and a name for the new test. If necessary, you can click **New > Test Workbench Project** to create a new project folder.
7. Optional: If you want to generate a test with multiple recordings (for example if your session involves switching between multiple apps), click **Next** and select the recordings that you want to use to generate the test.
8. Click **Finish**. The test editor opens in the test workbench and displays the generated test.

## What to do next

When the test is generated, you can edit the test in the test editor. For more information, see “Editing mobile tests” on page 24.

### Related concepts:

Mobile testing overview

### Related tasks:

Getting started with Android testing

## Recording tests from the iOS mobile test client

Mobile tests are typically created by recording a session on the mobile device that runs the app being tested. At the end of the recording session, you close the app and the IBM Rational Test Workbench Mobile Client uploads the recorded session to the IBM Mobile Test Workbench for Worklight where it is used to generate a test.

### Before you begin

To record tests, the mobile test client must be running on an iOS device or simulator and be connected to IBM Mobile Test Workbench for Worklight. For more information, see “Configuring the iOS mobile test client” on page 10.

You must have an instrumented iOS app installed on your device or simulator, and the latter must be connected to the test workbench. For more information, see “Instrumenting an iOS application” on page 19 and Command line used to instrument an iOS application via a script.

### Procedure

To record a session on the mobile device or simulator from an iOS app:

1. In the IBM Rational Test Workbench Mobile Client, tap **Manage applications**.
2. Select an app from the list and tap **Record**.
3. When the app starts, you can interact with the device using the iOS device or simulator. All your actions on the device and responses from the app are recorded.
4. To end the recording, tap **Home**. The recording is uploaded to the test workbench. Depending on the size of the recording, the upload might take a few seconds to several minutes. Recordings are displayed in the test navigator under **Mobile Incoming Recordings** with a name and a timestamp.

**Note:** If your session involves switching between apps, including multiple apps, a new recording is uploaded each time you switch apps. This action produces multiple recording logs in the **Mobile Incoming Recordings** folder. You can combine these multiple recordings to generate a single test.

5. In the test workbench Test Navigator, expand **Mobile Incoming Recordings**, right-click a recording and select **Generate Test**. The New Test from Incoming Recordings window opens.
6. Select a project folder and a name for the new test. If necessary, you can click **New > Test Workbench Project** to create a new project folder.
7. Optional: If you want to generate a test with multiple recordings (for example if your session involves switching between multiple apps), click **Next** and select the recordings that you want to use to generate the test.

8. Click **Finish**. The test editor opens in Rational Test Workbench and displays the generated test.

## What to do next

When the test is generated, you can edit the test in the test editor. For more information, see “Editing mobile tests.”

### Related concepts:

Mobile testing overview

### Related tasks:

Getting started with iOS testing


## Recording tests from the test workbench

When you initiate the recording from the IBM Mobile Test Workbench for Worklight, the recording does not start until you open an app in the mobile client.

### Before you begin

The workbench must be connected to the mobile client. For information about configuring the connection, see “Configuring the mobile test client for Android” on page 9 for the Android mobile client or “Configuring the iOS mobile test client” on page 10 for the iOS mobile client.

### Procedure

1. In the test workbench, click **File > New > Test From Recording**. Alternatively on the toolbar, click the **Test From Recording** icon .
2. Click **Mobile Test** and click **Next**.
3. Optional: If a test project is not created, click the **Create the parent folder** icon . to create a test project and click **Finish**.
4. Type a name for the test and click **Finish**.
5. On the mobile device, tap **mobile client > Managed Apps**.
6. Tap the app you want to record and tap **Record**. Now, you can interact with the app.
7. To stop the recording, close the apps on the mobile device and click the **Stop client** icon  in the workbench. The test is generated and available in the test project in the Test Navigator view.

---

## Editing mobile tests

With the test editor, you can view or customize a mobile test that you recorded.

The mobile test editor lists the actions that were logged on the device during the recording phase. Actions are represented in natural language, which allows you to document and reproduce the test manually.

There are two main areas in the test editor window. The area on the left, Test Contents, displays the chronological sequence of events in the test. The area on the right, Test Element Details, displays details about the currently selected action in the test script. In this area you can select a graphic object, an action related to the object, and its location. You can also define a time out and user’s think time to execute your test.

**Note:** Note that the maximum think time preference is ignored in mobile tests.

Figure 7. Mobile test editor

When actions are selected in the Test Contents list, the Mobile Data view is automatically synchronized to display a screen capture of the user interface of the app during the recording. You can use the Mobile Data view to select user interface elements and add some verification points, and to create or modify steps in the test with simplified scripts. Or you can create or modify a set of steps manually directly in the test script.

## Creating verification points in a test

Verification points verify that an expected behavior occurred during a run, or verify the state of a control or an object. When you create a verification point, you are capturing information about a control or an object in the application to establish this as baseline information for comparison during playback. When you run a test, the property is compared to see whether any changes have occurred in the application, either intentionally or unintentionally. This is useful for identifying possible defects when an application has been upgraded for example. An error is reported if the expected behavior did not occur.

### Before you begin



You can create verification points in the tests that are created from a native or web-based apps if the **Web UI actions and elements** for hybrid apps are activated. For more information, see “Activating web UI actions” on page 30.

### About this task

You can create verification points for any of the object properties such as label, color, and count, and you can verify that an object property is enabled, that it has focus, whether it is clickable and so on. Verification points can be created while recording a script or you can insert a verification point anytime in the script.

### Procedure

To create verification points:

1. In IBM Mobile Test Workbench for Worklight, open the test script and in the Test Contents area, click an action item for which you want to create a verification point.
2. Click the **insert** button and select **Verification point** for Android or Web UI, depending on the target application. Alternatively, right-click the selection or click **Options** and **insert** in the test editor to select the menu item.
3. In the **Test Element Details** section select a value for the **Graphic object** and **Verify attribute** artifacts identified as required for the action selected. Some artifacts are dependent on others, so when you select an attribute, you must select the values required for the options related to the selected attribute. To combine several attributes for the selected object, select the choice **all of** and select the object’s attribute. Click the **add attribute line button**  to add a new attribute. You can click the **remove attribute line button**  to remove an attribute.

4. Optionally select the **Retry verification point until attribute is verified or time out expires** and enter a value for the **time out**. The values in the graphic object and attributes lists are different for web UI apps and Android apps.
5. Save the test.

## Adding user actions in a test

You can add user actions in a test script from the test editor for Android or iOS apps.

### Before you begin

Create a test from a recording and open the test script in the test editor. You can create user actions in the tests which are created from mobile or Web-based apps if the **Web UI actions and elements** for hybrid apps is activated, see “Activating web UI actions” on page 30.

### Procedure

1. In the IBM Mobile Test Workbench for Worklight, open the test script; and in the Test Contents area, click an action item where you want to add a user action.
2. Click the **insert** button and select **Add user action** for Android or WebUI, depending on the target application. Another way is to right-click the selection or click **Options** and **insert** in the test editor to select the menu item.
3. In the **Test Element Details** section select a value for the artifacts identified as required for the action selected. To create a user action, you must specify an object and an action. The content of the **Graphic object** field is not the same for Android and iOS apps. The choices available in the object’s action list are dependent from the object selected. Other artifacts are optional. The values available in the mandatory fields are different for Web UI apps and Android apps.
4. A user action is added to the test script just before the node selected.
5. Save the test.

## Creating application stubs in tests for Android apps

You can use the test editor to add application stubs manually in your test. An application stub is a program or a piece of code used as a placeholder to simulate the behavior of software components such as a procedure on a remote machine. Use of this application stub will depend on the application being tested. The stub application replaces and simulates the behavior of the real object. The source code is temporarily replaced with a simple statement that returns a specific value to the application under test. You can manually create a stub without using a template, but consider using the stub created automatically in the recording app as a template for your application stub. This action applies only to tests created from Android apps.

### About this task

**Example:** To illustrate the use of stubs in a mobile application: When you tap on a phone number from a mobile device, you call this number, or if you tap an email address, you launch your mailer to send an email at this address. During the test recording, IBM Mobile Test Workbench for Worklight is able to detect this action (call or email) and to replace it by a stub instruction in the script so that there is no need to perform the action during the playback.

## Procedure

To manually create application stubs for Android apps

1. In the IBM Mobile Test Workbench for Worklight, open the test script and in the Test Contents area, click in the launch node where you want a stub to be added.
2. Click the **insert** button and select **Application stubs**. Alternatively, right-click the selection or click **Options** and **insert** in the test editor to select the menu item.
3. In the **Test element details** area, enter the name of the stub application that will simulate a service or a process. The name should contain key and scheme values.
4. In the **Input values** section, click **Add parameters** and enter a name for the operation element that describes the call that the stub expects to receive (scheme, data, and flag, for example), select the format (string, array, or other) of the call in the list items and a value retrieved from the recording app.
5. You can optionally enter values for the **Result code** and **return values**. The return value is the content that is returned by the stub service, simulating the response of the original service. This is the simulated value or canned value. There is one response element associated with each case element. Click **Add parameters** to enter a name for the response element, and then select a format and a value. If you want to delete all parameters, click the **Remove all** button.
6. The stub action is added to the test script with the name of the application stub before the item initially selected.
7. Save the test.

## Defining a variable to run a test with a selected mobile device

To be able to launch subsequent tests in the same logical flow (session) from the same devices, you must define a variable including a reserved variable name and selection criteria related to one or multiple devices.

### Procedure

1. In the Test Navigator, browse to the test and double-click it. The test opens.
2. To create a container for the test variables that you create in a test:
  - a. Open the test, and in the **Test Contents** area, click **Test Variables**, at the top of the test.
  - b. Select **Add > Test Variable Container**. A container named **Test Variables** is created for the user-defined variables.
  - c. Select the container to rename it. The **Test Element Details** area opens for you to type a new name in the **Name** field.
3. To define a variable in a test:
  - a. Open a test and select the test variable node.
  - b. Click **Insert > Variable Declaration**
  - c. Enter the name of the variable, which is a reserved name for this selection variable: `RTW_Mobile_Device_Properties` or `RTW_Mobile_Selected_Device`
  - d. Click **OK**. The variable is added as the last element in the container and the **Test Element Details** area opens.
  - e. In the **Visible in** section, select **This test only** to restrict data to the current test only. Even if another test has a variable with the same name, that variable will not change. Select **All tests for this user** to share the value of



this variable when the test runs in a compound test. For the variable to be shared, both tests must have a variable with the same name and must have this option enabled.

4. Assign a specific value to the variable and initialize the variable:
  - a. Select **Text**
  - b. Enter a selection sentence to assign a variable value to a text string. Enter selection strings including a device's property, followed by an operator value, property's value and a comma separating each string. For more details on the main device properties you can use and on the syntactic rules, see the Variable selection values topic.

## Results

The variable can then be initialized from some external source (a datapool, tests from an IBM Rational Quality Manager test suite, or tests from the same user in compound tests containing one or more mobile tests. It can also be set within a test's execution with a variable assignment action from any data source, including a data correlation reference, custom code, built-in function, datapool, or string constant. As a result, successive tests in the same session will then be sure to run on the same actual devices.

**Note:** When a test launches an application:

- The *RTW\_Mobile\_Selected\_Device* variable content is checked to get the device ID
- The device is reused if it is still applicable to the app that must be launched.  
Conditions:
  - The device operating system must be the same as the operating system of the application to be launched.
  - The tester app is installed or can be installed without user intervention.
- If the conditions are not matched, the content of the *RTW\_Mobile\_Device\_Properties* variable is checked
- If this variable is set, the first device matching all the valid property expressions of the variable is selected
- If the variable is not set, the first applicable device ready for test is used.  
Conditions:
  - The device operating system must be the same as the operating system of the application to be launched.
  - The tester app is installed or can be installed without user intervention.

## Assigning a test variable to an object's property

You can assign a new value to a test variable and set it to a Mobile object's property.

### Before you begin

To be able to create a variable assignment, you must first declare a variable. This task is explained in the Declaring and assigning test variables page. Then you can set a value for the object's property, it will be used when the variable will be executed in the test.

You can create variable assignments in the tests which are created from mobile or Web-based apps if the **Web UI actions and elements** for hybrid apps is activated, see "Activating web UI actions" on page 30.



## Procedure

To create a variable assignment and set it the value to a Mobile object's property:

1. Open the test, and in the **Test Contents** area, select a test element.
2. Select **Insert > Variable Assignment**, which inserts the assignment before the selected element. The Test Editor window opens and lists the variables available to the test.
3. Select the variable that you are assigning a value to, and, in the **Set to** box in the **Test Element Details** area, select **Mobile/Web object's property**, set the value for the variable to Mobile object's property. Select a graphic object and the object's property. The values of the properties are different for Web UI apps and Android apps.
4. Save the test. A set statement is added to the test, with the value you chose.

## Adding hardware actions in a test

You can add hardware actions to your test. It consists in creating an action using a physical widget.

### Before you begin

You must have created a test from a recording and have the test script open in the test editor. You can add hardware actions in the tests which are created from mobile or Web-based apps if the **Web UI actions and elements** for hybrid apps is activated, see Activating Web Ui actions.

### About this task

This action applies only to tests created from Android apps.

### Procedure

1. In the IBM Mobile Test Workbench for Worklight, open a test script and in the Test Contents area, click in the launch app node where you want the action to be added.
2. Click the **insert** button and select **Hardware action**. Another way is to right-click the selection or click **Options** and **insert** in the test editor to select the menu item. The values are different for Web UI apps and Android apps.
3. In the **Test Element Details** section, select an item in the list of object's actions. You can enter a value for the timeout too. The new action is added before the script item you had initially selected in the launch node.
4. Save the test.

## Splitting a test

After you record a test, you can split the test actions into multiple test segments with different nodes. With the test-splitting capability, you can record a relatively long scenario with many functional steps against an application and then, in the editor, modify the target apps. Then you can generate multiple tests from a single recording that you can replay in a different order with a schedule.

### Procedure

To split a mobile test:

1. In the Test Navigator, browse to the test and double-click it. The test opens.

2. In the test editor, select one or more actions in the test script for splitting into one or more application nodes. You can select elements, except for variable containers, that are immediate children of the root node of the test.
3. Right-click the selected elements, and then select **Split Mobile actions**.
4. In the refactoring test dialog box that opens, examine the changes to be performed as a result of the split. You can leave or clear the options if you do not want certain data to be correlated.
5. Click **OK**. One or more app nodes *In application: AppName* are created in the test script from the selected test element.
6. Optionally: you can change the target app to be tested for a selected app node. To do so, select an app node, click the **Change application** button and in the list of mobile apps available, select a new app. Then select the **Starts a new instance of application selected below**. To apply the change of app to the all the test nodes, that is to the whole test suite, click the ... The test nodes turns from *In application: AppName* to *Launch application: AppName*.
7. Save the test.

## Activating web UI actions

You can configure an app under test to be able to perform web UI actions and support web UI elements.

### About this task

### Procedure

To activate web UI actions:

1. Complete one of the following steps:
  - a. Open the application editor, click an app in the list of available apps and, in the right area, select the **Allow web UI actions and elements for this hybrid app** option.
  - a. From the test editor, open a test, click the launch app node and, in the right area, select the option **Allow web UI actions and elements for this hybrid app**.

Then, new menu items are available from the **Add** button. You can add user actions, verification points and create variable assignments for these web UI enabled apps.

To deactivate the web UI actions:

2. Right-click the launch app node and select **Disallow Web UI actions and elements**.

## Actions from the Mobile data view

A mobile data view displays the screen captures that were uploaded from the mobile device during the recording. Use this view to display and select user interface (UI) elements and optionally add verification points to the test script.

### Adding user actions in a test from the Mobile data view

The Mobile Data view offers graphical and hierarchical views of the current step in a test. It also displays a table of properties associated with a selected object. You can also use this view to quickly create user actions, adding steps to the test using the selected graphical elements.

## Before you begin

You must have created a test from a recording and have the test script open in the test editor.

## About this task

You can add actions for any of the widgets in the Mobile Data view.

## Procedure

To add a user action in a test from the Mobile Data view:

1. In the **Test Contents** area of the test editor, click an action item.
2. In the **Screen capture** view, select a graphical object or the corresponding element in the hierarchical list **Elements**, and then right-click and select **Add user action for this element**. In the test editor, a step is added just before the current selected node.
3. In the **Test Element Details** section, select a value for the action of the object.
4. Save the test.

## Modifying a step in a test from the Mobile data view

You can modify the step targets in a test from the Mobile Data view. Two options are available from the menu items. One is used to modify an action in a test script and assign a new object as target of the action. The other option is used to define execution variables for the selected object and give a new value to the property associated with the object.

## Before you begin

You must have created a test from a recording and have the test script open in the test editor. In the IBM Mobile Test Workbench for Worklight, in the Test Contents area, you must have selected the action item for which you want to modify the step.

## About this task

You can modify a step target in a test by either assigning an object as the step target or create a variable assignment from a `propertyName`.

## Procedure

To assign a new object as step target:

1. In the **Screen capture** view, select a graphical object or the corresponding element in the hierarchical list **Elements**, and then right-click and select **Use this element as step target**. In the test editor, the current step target is replaced by the selected graphical object.
2. In the **Test Element Details** section, you can change the action or location initially specified to fit the new target.

To assign a variable and set a new value for the object's property:

3. Select an object in the **Screen capture** or **Elements** view, and then right-click and select **Create variable assignment from propertyName**.
4. In the dialog box, search for a variable that was created in your test. To do so, enter a name to filter the list of available variables and click the one matching the name. Click **OK**.

5. Save the test.

### Creating verification points from a Mobile Data view

You can create some verification points in a test with simplified scripts by using the Mobile Data view. A verification point can be added for an object or created for the properties of the object.

#### Before you begin

To be able to create verification points, you must create variables in the test editor first Declaring and assigning test variables

#### About this task

You can create verification points for any of the widgets or widget properties.

#### Procedure

To create verification points:

1. In the IBM Mobile Test Workbench for Worklight, open the test script, and in the Test Contents area, click an action item for which you want to create a verification point.
2. In the Mobile Data view, select an object in the **Screen capture** view, an item in the hierarchical list of **Elements**, or a property in the table.
3. Right-click and click **Create Verification Point for propertyName**. Note that the properties displayed will be limited to those available for the selected object. A new step is added to the test script for the verification point.
4. Select a value in **Graphic object** and a value for the object's property in **Verify attribute**.
5. Save the test.

---

## Running mobile tests

You can run a mobile test, either from a device, a simulator, or from the test workbench. It is possible to run a test with different localized strings from your device, if there are language differences between the application under test and the mobile device.

### Running tests from an Android mobile test client

You can run a test from an Android mobile device or emulator. After the run, the report is automatically uploaded to the test workbench. You can also view the report on the Rational Test Workbench Mobile Client.

#### Before you begin

- You must have recorded and generated a test as mentioned in Recording Android tests.
- Ensure that you give control to the mobile test client to run the tests by tapping **Switch to active mode**. Then the button changes to **Switch to passive mode**.

#### Procedure

1. In the mobile test client, tap **Managed Apps** and tap the application under test.
2. To view the list of tests available for the app, tap **Test**.
3. Tap the test script, and then tap **Run Test**.

**Note:** If your device or emulator does not have silent mode, the mobile test client uninstalls the installed version of the application under test and installs it again. During this process, tap the **Uninstall**, **OK**, and **Install** buttons accordingly. If your device or emulator has silent mode, this process happens in the background.

To make the silent mode option available on a device, you must connect the device with a computer that has the Android SDK installed. Use an USB cable and enable USB debugging. Ensure that you installed the appropriate USB driver. Next, open the command prompt on the computer and run the following commands:

- `adb devices`: Lists the devices connected to the computer by the USB cable.
- `adb tcpip 5555`: Makes the silent mode option available on the device.

You must follow these steps every time you reboot your device.

Silent mode is not available on devices and emulators with API level 17 (Android 4.2 to 4.2.2), due to a known limitation

The test is played back in the mobile device. Do not interact with the mobile device till the test is complete.

## What to do next

You can now evaluate the test results. See Evaluate results.

### Related concepts:

Mobile testing overview

### Related tasks:

Getting started with Android testing

## Running tests from the iOS mobile test client

You can run a mobile test from an iOS simulator or device, and this will generate a report that is automatically uploaded to IBM Mobile Test Workbench for Worklight. You can also view the report in the IBM Rational Test Workbench Mobile Client.

### Before you begin

- You must have recorded and generated a test as described in Recording tests from the iOS mobile test client .

### Procedure

To run a test from the iOS device or simulator:

1. Tap **Manage Applications** and tap the application under test.
2. To view the list of tests available for the app, tap **Test**.
3. Tap the test script, and then tap **Run Test**.

The test is played back in the iOS simulator. Do not interact with the simulator until the test is complete.

## What to do next

You can now evaluate the test results. For more information, see Evaluate results.

**Related concepts:**

Mobile testing overview

**Related tasks:**

Getting started with iOS testing

## Running tests from the test workbench

After generating the test from a recording, you can edit a test according to your requirements and run it. The result of the test can be viewed from both the test workbench and the IBM Rational Test Workbench Mobile Client.

### Before you begin

- You must have recorded and generated a test as mentioned in “Recording tests from the test workbench” on page 24.
- Ensure that you give control to the test workbench to run tests by tapping **Switch to passive mode**. After you tap it, the button changes to **Switch to active mode** on Android clients. A message tells you to tap **Home** to end this mode or wait until a playback scenario starts on iPhones, or to select any other page on the left on iPads.

### About this task

This procedure applies to Android and iOS applications.

### Procedure

1. From the test workbench, you can initiate the running of a mobile test by using any of the following steps in the Test Workbench perspective:
  - In the Test Navigator view, right-click a test and click **Run As > Test**.
  - From the Test Navigator view, open the test and, in the test editor, click the **Run** button.
  - Add the mobile test to the Compound Test editor.
2. The first time you run a test, you will be prompted to run the test from the Test Execution perspective. Select the **Remember my decision** check box to avoid receiving the message again and click **Yes**. The test is played back in the mobile device. Do not interact with the mobile device till the test is complete.

### What to do next

You can now evaluate the test results. See Evaluate results.

**Related concepts:**

Mobile testing overview

**Related tasks:**

Getting started with Android testing

Getting started with iOS testing

## Running a test with different localized strings

When you record a test on a mobile device, the test is always generated in the default language of the application. However, it is possible that the language defined for the device running the test is different from the default language of the application. This language difference between the mobile device and the application means that to replay the test on the mobile device, you must convert



the mobile strings in your test script into the localized strings of the application. You can do this only if the application has been localized.

## Before you begin

You must have created and recorded a test. To be able to convert the standard strings in your test script into localized strings, you must verify first that the application under test contains translation strings.

## About this task

The words *mobile strings* define the name of graphic objects such as buttons or objects identified by texts in the test script recording. Note that you can convert all the mobile strings into localized strings in your tests at one time, or convert them one by one.

## Procedure

1. Verify that the application has been localized:
  - a. In the mobile application node of the Test Navigator view, double-click your application file or click the **display available mobile applications** icon on the toolbar. In the Mobile Applications editor that opens, select an application from the list.
  - b. In the right pane of the editor, click on the Localized Strings tab. A table displays the translation keys that are found in the application for the mobile strings.
  - c. Click on the **Locale** column heading to see the languages handled by the application. You can apply filters to sort the data items in the table. The filter applies to the key by default but you can filter strings or locales. To do so, enter a value in the filter field and click one of the following icons: **Filter using key** to filter the keys, **Filter using key** to filter the strings, **Filter using locale** to filter the locales.
  - d. Check that you find the appropriate translated strings in the target language of the mobile device that will be used to run the test.
2. Choose how you want to convert the mobile strings in your test script into localized strings of the application.
  - Convert the full set of mobile strings:
    - In the Test Navigator view, double-click on your test file or right-click and click on **Test editor** to edit the test.
    - In the test script, right-click on the root node and click on **Convert mobile strings into localized strings**. The **Localize mobile strings in test** wizard opens:
      - Click on the **Locale** column heading in the table and select a locale for the translation of the strings. This should be the locale used on the device during the test recording. As a result, the table displays the translated strings available in the application. The rows containing translated strings are checked. If several keys are available for a string, you must select a key.
      - In the next cell, click **select key** and choose the appropriate key in the list. Click **Finish**.
      - Now, in the test script, you can see that the localized strings are underlined. If you click on a localized string in the test script that



- corresponds to a graphic object identified by text, you can see in the right pane that the **Text** field contains multiple choices for the current string.
- Convert a single mobile string in your test script into a localized string of the application:
    - In the test script, select the launch application node. In the right pane, click **Used locale for localized strings** and select a language the locale used to record the test script. If your test contains instances of other applications or several nodes, click the **Apply selected locale to** icon and select one of the choices **Apply locale to the same application node** or **Apply locale to all application nodes**.
    - Select the node containing the mobile strings converted to localized strings and right-click on the text edit in the right pane, then choose "Convert string to localized string". In the test script, now you can see that the localized strings are underlined. If you click on a localized string in the test script that corresponds to a graphic object identified by text, you can see in the right pane that the Text field contains multiple choices for the current string.
3. Convert the localized strings in your test into standard strings. If you want to have the localized strings or the localization keys as standard values in your test script, you must convert the mobile strings into standard strings in the test script.
    - a. Click a mobile element in the test containing a localized string. In the right pane, right-click on the **Text** field. A list containing multiple choices for the selected string is displayed. You can filter the list.
    - b. Double-click on the string of your choice in the list and click **Convert into standard string using localized string as value** to have the selected localized string in the test or **Convert into standard string using localization key as value** to have the associated key in the test.
  4. Save and replay your test. You can run the test in different language environments.
  5. In the test report, you can see that the object names and the text are displayed in the new target language.

**Related concepts:**

Mobile testing overview

**Related tasks:**

"Recording tests from the Android mobile test client" on page 21

"Recording tests from the iOS mobile test client" on page 23

"Recording tests from the test workbench" on page 24

"Running tests from an Android mobile test client" on page 32

"Running tests from the iOS mobile test client" on page 33

"Running tests from the test workbench" on page 34

"Viewing mobile reports" on page 37

Getting started with Android testing

Getting started with iOS testing

---

## Evaluating results

To check whether or not the mobile test ran successfully, you can open the test report. You can also view each recorded functional action in the report.

## About this task

When you run a test from the IBM Mobile Test Workbench for Worklight, you can view both the mobile web report and the statistical report in the test workbench. By default, the mobile web report is displayed after the run. You can also view this report on the mobile device.

To open the mobile web report and statistical report, from the Test Navigator view double-click a result from the Results folder.

When you run a test from the mobile device or emulator, you can view the mobile web report in the device or emulator. After the run, the report is uploaded to the test workbench automatically. There is no statistical report for the test that is run from the device or emulator.

The report is in a tabular format and displays the application that was tested, its execution status, and duration of the test. Each action is displayed in a row with the screen capture of the action highlighted and the time taken for that action from the beginning of the test.

If you added verification points to the test, you can also view the verification points entries in the report. The Execution Status of the report displays Failure, if the verification points fail.

## Viewing mobile reports

You can choose to view the test reports on the mobile device, on an emulator, or on the test workbench.

### About this task

This procedure applies to tests generated from Android or iOS applications.

### Procedure

Choose one of the following steps:

1. To view the test reports from the IBM Rational Test Workbench Mobile Client device, emulator, or simulator:
  - a. Open the mobile test client and tap **Managed Apps**.
  - b. Select the app for which you want to view the results.
  - c. Tap **Test** and tap a test for which you want to view the reports.
  - d. Tap **Reports**.
2. To view the test reports from the test workbench:
  - a. From the Test Navigator, expand the project folder and double-click the test reports. Each report begins with the name of the test, and ends with the timestamp of the run in brackets. If the Test Navigator is logically arranged, expand the project folder, expand the **Results** folder, and open reports. If you have initiated the test run from the device, the mobile reports are stored in the **Mobile Results** folder.

## Managing logs for Android mobile test client

To debug issues that occur in the IBM Rational Test Workbench Mobile Client, the logs store all type of messages in the mobile test client. By default, the error and warning messages are stored. You can also set the log level to store the information messages.

### About this task

The logs are stored in the mobile device or on an emulator. By default, the log data is stored until the mobile test client is running at a particular session. You can manage the logs in the following ways:

- Set the number of messages to be logged
- Set the type of messages to be logged
- Store all the logs
- Capture the recording actions

You can also upload the logs to the test workbench. For more information about uploading logs, see [Uploading logs to workbench](#).

### Procedure

To manage logs:

1. In the Rational Test Workbench Mobile Client, tap the dropdown arrow menu and tap **Settings**.
2. In the workbench settings, complete any of the following steps:
  - To set the number of messages to be logged, in **Log Size**, drag the slider.
  - To select the type of messages to be logged, in **Filter log messages**, drag the slider. By default, Fatal, Error, and Warning messages are logged. You can log Information, Debug, and Trace messages as well.
  - To store the logs, tap the **Persist log** check box. If this check box is selected, logs of the mobile test client are stored in the disk of the mobile device.

**Note:** Consumption of the large amount of disk space might reduce the speed of the device.

- To capture the recording trace, tap the **Trace recording** check box. If this check box is selected, the mobile test client logs all the recording events. Typically, you use this option only when requested to do so by IBM® Software Support.

### Uploading logs to test workbench

To share the device logs with other users or Support, you can upload the logs from IBM Rational Test Workbench Mobile Client to the test workbench, if you are using Android mobile test client only. After the upload, you can export the logs to a text file and share the file.

### Before you begin

The mobile test client must be connected to the test workbench.

## About this task

When you upload the logs to the test workbench, the logs remain in the device until the mobile test client is running. In the test workbench, the logs show up in the Error Log view.

### Procedure

1. In the mobile test client, open the menu and tap **Logs**.

**Note:** On some devices, the menu is in the form of a dropdown icon placed in the upper-right corner of the screen.

2. Tap **Upload**. In the test workbench, click **Window > Show View > Error Log** to view the uploaded logs.
3. Optional: To clear the logs, tap **Clear**. This action removes the logs from the display, but the logs remain available in the test workbench.
4. Optional: To delete the logs, tap **Delete** and tap **Yes**.

---

## Compound tests

You can create compound tests to help you organize smaller tests into scenarios that can then be run end-to-end. Each of the smaller tests in a compound test can run on a different domain if required, such as a mobile device, or a web browser, and so on.

If you need to combine various tests into a single workflow or end-to-end scenario, you can organize the tests into a compound test. Each of the tests may perform parts of the scenario. Each of the tests may also run in a different domain if required, for example, a web browser or a mobile device, or others. A typical example of a compound test is an online buying workflow. You may have built smaller tests for each part of an online purchase transaction, such as "log on", "log out", "view item", "add to cart", and "check out". You can combine these tests into a single flow in a compound test. When the compound test is run, its individual tests are run in sequence.

The types of tests you can combine into a compound test depend on the testing capabilities you have purchased. If you have purchased only mobile testing capabilities, you can combine tests on mobile applications into a compound test. If you have purchased additional testing capabilities along with mobile testing, you can also combine tests built using Selenium, HTTP tests, Socket tests, Citrix tests or SAP tests into a compound test.

To build the scenario you require in a compound test, you can also add the following annotations:

- Comments
- Synchronization points
- Loops
- Delays
- Transaction folders
- Tests that are mandatory, using the **Finally** blocks
- Tests to be run in random order, using the **Random Selector**

## Creating a compound test

You can create compound tests to help you organize smaller tests into scenarios that can then be run end-to-end. Each of the smaller tests in a compound test can run on a different domain if required, such as a mobile device, or a web browser, and so on.

### Procedure

1. Create a project.
2. In the Test Workbench perspective, in the Test Navigator, right-click the project and click **New**, and then click **Compound Test**.
3. In the New Compound Test dialog box, specify the name of the compound test and the location where it must be stored. By default, the test is stored in the workspace of the project you selected. You can select a different project location if desired. The file extension `testsuite` is added to the file name, and the new compound test is added to the Compound Tests folder of the project, visible in the Logical View. The new test is also visible in the Resource View, under the project. The contents and test element details are displayed in the compound test editor in the right panel.
4. In the compound test editor, add the components of the compound test. The types of tests you can combine into a compound test depend on the testing capabilities you have purchased. If you have purchased only mobile testing capabilities, you can combine tests on mobile applications into a compound test. If you have purchased additional testing capabilities along with mobile testing, you can also combine tests built using Selenium, HTTP tests, Socket tests, Citrix tests or SAP tests into a compound test.
5. To build the scenario you require in a compound test, you can also add the following annotations by clicking **Add** and selecting the appropriate option:
  - Comments
  - Synchronization points
  - Loops
  - Delays
  - Transaction folders
  - Tests that are mandatory, using the **Finally** blocks
  - Tests to be run in random order, using the **Random Selector**
6. Save your changes.

## Viewing compound tests

You can view a compound test in the Compound Test Editor.

### About this task

When you open a workspace, the tests and projects that reside in the workspace are listed in the Test Navigator.

You can view compound tests in the Logical and Resource Views in the Test Navigator. From any of these views, you can open the test in the Compound Test Editor.

### Procedure

- In the Logical View of the Test Navigator, compound tests are listed in the Compound Tests folder under the project into which they were imported.

Double-click the compound test under the Compound Tests folder to open it in the Compound Test Editor. In the Resource View, all tests under a project are shown in the project folder. Double click the compound test under the project folder to open it in the Compound Test Editor.

- In the Java perspective, compound tests under a project are shown under the root project folder. Double click the compound test under the project folder to open it in the Compound Test Editor.
- The Compound Test Editor contains two panels - the **Compound Test Elements** panel, where the elements of the workflow are listed. Click one of the elements, and its details are displayed in the far right portion of the right panel, which is the **Compound Test Element Details** panel. Double-click any of the test or the test elements to view its details. The name of the test, test path, source type and execution mode are displayed.

## Adding tests into a compound test

After creating a compound test, you can add the smaller test pieces that contribute to the larger workflow you are constructing with the compound test. When you run a compound test, each of the tests added to it are invoked in the sequence defined.

You can add many tests of the same type, or different types, to a compound test, depending on the testing capabilities you have purchased. If you have purchased only mobile testing capabilities, you can combine tests on mobile applications into a compound test. If you have purchased additional testing capabilities along with mobile testing, you can also combine tests built using Selenium, HTTP tests, Socket tests, Citrix tests or SAP tests into a compound test.

To add tests to a compound test, complete these steps:

1. In the Test Navigator, double-click the compound test to which you want to add a test. The contents of the compound test are shown in the **Compound Test Contents** panel in the Compound Test editor.
2. Do one of the following:
  - Click **Add** to add a test as the first element in the compound test.
  - To insert a test before a specific element in the compound test, select the element and click **Insert**.

The Select Tests dialog box is opened, and the tests found in the Eclipse Client workspace are displayed.

3. Select the test you want to add to the Compound test, and click **OK**. The test is added to the compound test, and is displayed as part of the elements of the compound test in the **Compound Test Contents** panel. When you click the test you added, its details are displayed in the **Compound Test Element Details** panel in the Compound Test editor.
4. Save your changes.

In addition to the tests that you can add to a compound test, you can also add the following elements to construct the workflow you need:

- Comments to document the test
- Delays in the test
- Synchronization points
- Loops
- Transaction folders



- Parts of the test that are mandatory
- Tests to be run in random order

## Modifying a compound test

You can modify a compound test in the Compound Test Editor.

### About this task

A compound test is a testing workflow comprising smaller tests and other test elements in a certain sequence. You might want to order the tests and test elements to suit your workflow requirement, or add further tests and elements.

### Procedure

1. In the Test Navigator, double-click the compound test that you want to modify. Its elements are shown in the **Compound Test Contents** right panel in the Eclipse Client.
2. To add a test or test element at the beginning of the compound test elements list, select the compound test in the **Compound Test Contents** panel, click **Add**, and then click **Test**. To insert a test or test element into the test, select the test element before which the insertion must be made, and click **Insert**.
3. Add or insert the test or test element you need, and click **OK**. The modified compound test displays its updated elements in the **Compound Test Contents** right panel.
4. Save your changes.

## Running compound tests

When you run a compound test, its test elements are run in the order defined in the compound test.

### About this task

When you run a compound test, you are prompted to open the Test Execution perspective, in which details of the test run are displayed. When the test run is complete, the Test Log displays the run results.

### Procedure

1. In the Test Navigator, select the compound test you want to run.
2. Click the Run As icon on the toolbar. The test runs. To run a launch configuration option, click the arrow beside the Run As icon and select Run Configuration. Select the desired configuration option and run the test. The Confirm Perspective Switch dialog box is opened, prompting you to switch to the Test Execution perspective. Click **Yes**.
3. Select the desired option to run the test. The Test Execution perspective is opened and the test runs. On completion, the test log is displayed.

### Results

You can work with the test log by exporting it into a flat file.



## Generating compound test result reports

When a compound test run is completed, a Test Log is shown in the Test Execution perspective. You can work with the information in the test log and also generate test result reports.

### Exporting the Test Log

When a compound test run is completed, a Test Log is displayed in the Test Execution perspective.

#### About this task

The Test Log displays the following details:

- The General Information tab displays the name of the compound test and its description. The location of the test log file is also shown.
- The Common Properties tab shows the verdict of the test results.
- The Verdict Summary and Verdict List tabs provide a pie chart of verdicts for different components of the test, and a list of the first 20 verdicts. You can view details about the verdicts by clicking the links in the Verdict List tab.

You can export the contents of the test log to a full-text file.

#### Procedure

1. To export the contents of the test log to a full-text file, right-click the test run result under the Results folder of the compound test, and click **Export Test Log**.
2. In the Export Test Log dialog box, specify where the test log should be exported to, in the **Location** field.
3. Select the format in which the log must be exported, from the list in the **Export Format** field. You can select either Flat Text - Default Encoding or Flat Text - Unicode Encoding.
4. Click **Finish**. The test log is exported as a full-text file, with the test results run name, to the location you specified.

### Generating a functional test report

You can generate a functional test report from the test run results as a HTML file.

#### About this task

When you generate a functional test report as a HTML file, the following details are displayed in the report:

- A global summary, which lists the number of tests run, verification points, defects
- A test summary which displays the name of each test, the start and end times and the verdicts.

#### Procedure

1. In the Test Workbench perspective, test run results are displayed under the Results folder of a project. Right-click the test run result you want to view and click **Generate Functional Test Report**. The Generate Functional Test Report dialog box is opened.
2. Select the parent folder in which the report must be stored.
3. By default, the name of the compound test and the date and time stamp is displayed as the name of the report in the **Name** field. You can change the name.

4. Click **Next**.
5. Select the report template to be used. If you select the Common Functional Test Report (XSL) format, the report is generated as a HTML file. If you select the Common Test Functional Report format, you can select either the HTML or PDF output format.
6. Click **Finish**. The report is generated and displayed. The report is listed under the Functional Reports folder under the compound test in the Test Navigator.

### Creating an executive summary

You can create an executive summary or test statistics report from the test run results. Executive summaries are generated according to the type of test.

#### About this task

An executive summary displays the tests and methods that were run, and their success or failure information. This information is shown in summary charts as well as in bar graphs.

#### Procedure

1. Under the Results folder of the project, right-click the test run result you want to view and click **Create Executive Summary**. The Generate Functional Test Report dialog box is opened.
2. Select the type of test report you want to generate.
3. Click **Finish**. The report is generated and displayed. The report is listed under the Functional Reports folder under the compound test in the Test Navigator.

## Adding a compound test to a Test Workbench project

You can create a compound test in a project. If you have an existing compound test, you can import the test to a project.

### Creating a compound test in a project

You can create a compound test in a project.

#### Procedure

1. Create a project.
2. In the Test Workbench perspective, in the Test Navigator, right-click the project and click **New**, and then click **Compound Test**.
3. In the New Compound Test dialog box, specify the name of the compound test and the location where it must be stored. By default, the test is stored in the workspace of the project you selected. You can select a different project location if desired. The file extension `testsuite` is added to the file name, and the new compound test is added to the Compound Tests folder of the project, visible in the Logical View. The new test is also visible in the Resource View, under the project. The contents and test element details are displayed in the compound test editor in the right panel.
4. In the compound test editor, add the components of the compound test. The types of tests you can combine into a compound test depend on the testing capabilities you have purchased. If you have purchased only mobile testing capabilities, you can combine tests on mobile applications into a compound test. If you have purchased additional testing capabilities along with mobile testing, you can also combine tests built using Selenium, HTTP tests, Socket tests, Citrix tests or SAP tests into a compound test.

5. To build the scenario you require in a compound test, you can also add the following annotations by clicking **Add** and selecting the appropriate option:
  - Comments
  - Synchronization points
  - Loops
  - Delays
  - Transaction folders
  - Tests that are mandatory, using the **Finally** blocks
  - Tests to be run in random order, using the **Random Selector**
6. Save your changes.

### Importing a compound test into a Test Workbench project:

You can import a compound test into a project.

#### Procedure

1. In the Test Workbench perspective, in the Test Navigator, right-click the project into which you want to import the compound test and click **Import**.
2. In the Import dialog box, expand **General** in the source list, select **Import test assets with dependencies** and then click **Next**.
3. Specify the directory in which the compound test resides. Click **Browse**. By default, the compound test is imported into the project folder.
4. The compound test assets in the folder you selected are displayed. Select the components you want to import.
5. Click **Finish**. The imported compound test is displayed in the **Compound Test Elements** panel in the Compound Test editor.

---

## Extending Rational Test Workbench Eclipse Client

### Extending test execution with custom code

You can extend how you run your tests by writing custom Java code and calling the code from the test. You can also specify that results from the tests that are affected by your custom code be included in reports.

#### Creating custom Java code

Custom code uses references in the test as input and returns modified values to the test. Use the `ICustomCode2` interface to create custom code and the `ITestExecutionServices` interface to extend test execution. These interfaces are contained in the `com.ibm.rational.test.lt.kernel.services` package.

#### About this task

**Note:** When you use the `ITestExecutionServices` interface in your custom code to report test results, the results for the custom code are displayed in the test log. If you log custom verification point verdicts, these are reflected in the overall schedule verdict.

Custom code input values can be located in references or field references. You can also pass a text string as an argument to custom code. References that are used as input to custom code must be included in the same test as the custom code. In the test, the reference must precede the code that it affects. Verify that the test contains

the references that are required for customized inputs to your code. For details about creating references and field references, see [Creating a reference or field reference](#).

If your custom code uses external JAR files, you might need to change the Java build path. In some cases, you can avoid changing the build path manually by running the test before adding your custom code to it. The first time a test runs, classes and libraries that are required for compilation are added to the build path. For example, you can import Test and Performance Tools Platform (TPTP) classes that are required to create custom events in the test log if the test, to which you have added your custom code, has run previously. However, if the test has never been run, import errors occur because the classes are not named in the build path for the project until the test has run.

If your code uses external resources, for example, an SQL database or a product that manages customer relationships, you must configure the custom code to work on every computer on which your test runs.

Custom code is saved in the `src` folder of the project that contains the test that calls the code. By default, custom code is located in a package named `test` in the `src` folder.

You can reuse a custom code package for tests that are located in multiple projects. The projects must be in one workspace. To reuse custom code across projects, use the project name before the custom code package. For example, `..`

The following example shows the standard Navigator view of two custom code classes. (The Test Navigator does not display Java source files.)

When you add the `ReplaceCC.java` and `VerifyUserID.java` custom code classes to the test and return a value to the test, **Substitute** lists these two classes.

The test package also contains the generated Java code for tests in the project.

You can put custom code in a different package (for example, `custom`). Separate custom code from generated code, especially if you use a source-control system.

## Procedure

To add custom code:

1. Open the test, and select a test element.
2. Click **Add** or **Insert**, and select **Custom Code**. **Add** appends the custom code to the bottom of the selected test element. **Insert** adds the custom code above the selected test element.

**Note:** After you add or insert custom code, the Problems view displays an error stating that the new custom code element has no Java file. This error message remains until you click **View Code** or **Generate Code**, to remind you that the custom code test element is not yet associated with any Java code.

3. Inspect the **Class name** field, and complete one of these steps:
  - If the code to call already exists, change the class name to match its name. Click **View Code** to open the code in the Java editor.

- If the code does not exist, change the class name to describe the purpose of the code. Click **Generate Code** to generate a template class for logging results and to open it in the Java editor. If a class with this name exists, you are warned that it will be overwritten.
4. In the **Arguments** field, click **Add**.
  5. In the Custom Code window, select all inputs that your code requires. The Custom Code window lists all values in the test that can be used as inputs to your code (references or field references in the test that precede the code).
  6. Click **OK**. The window closes, the selected references are added to the **Arguments** field.
  7. Optional: To add text strings as inputs to your custom code, click **Text**, and then type the text string to use.
  8. In the test, after your custom code, locate a value that your code returns to the test.
  9. Highlight the value.
  10. Right-click the highlighted value, click **Substitute**, and select the class name of your custom code. The custom code classes that you have added are listed. After you have made your selection, the value to be returned to the test is highlighted in orange, and the **Used by** table is updated with this information.

### What to do next

Custom code is not displayed in the Test Navigator view. To view custom code, open the Package Explorer view and use the Java tools to identify the custom code that you added.

### Test execution services interfaces and classes

You use the test execution services interfaces and classes to customize how you run tests. These interfaces and classes are located in the `com.ibm.rational.test.lt.kernel` package. Each interface and class is described briefly in this topic and in detail in the Javadoc information.

The custom code does not run on the mobile device, but from the generated Java code that is available in the `.`. So, if you initiate the test run from the mobile device and the test script includes custom code, the custom code is not executed. To execute the custom code that is available in a mobile test scrip, you must initiate the run from `.`. If you want to integrate custom code between two mobile instructions, you must split the test script. See [Splitting a test](#).

The Javadoc for the test execution services interfaces and classes are in this [reference topic](#).

#### Test execution services interfaces

Interface	Description
ICustomCode2	Defines customized Java code for test execution services. Use this interface to create all custom code.

## Test execution services interfaces

Interface	Description
ITestExecutionServices	Provides information for adding custom test execution features to tests. Replaces the ILog interface. All the methods that were available in ILog are contained in ITestExecutionServices, along with several newly exposed objects and interfaces. This interface is the primary interface for execution services. ITestExecutionServices contains the following interfaces: IDataArea, IARM, ILoopControl, IPDLogManager, IStatisticsManager, ITestLogManager, ITime, and ITransaction.
IDataArea	Defines methods for storing and accessing objects in data areas. A data area is a container that holds objects. The elements of a data area are similar to program variables and are scoped to the owning container. To use objects specific to a protocol, you should use objects provided by that protocol that are stored in the protocol-specific data area.
IARM	Provides information about defining ARM (Application Response Measurement) specifications. You use this interface if your virtual users are being sampled for ARM processing.
ILoopControl	Provides control over loops in a test or schedule. For example, you can use this interface to break loops at specific points in a test. The loop that is affected is the nearest containing loop found in either the test or the schedule.
IPDLogManager	Provides logging information such as problem severity, location levels, and error messages.
IStatisticsManager	Provides access to performance counters in the ICustomCode2 interface (used for defining custom code). Performance counters are stored in a hierarchy of counters. Periodically, all the counter values in the hierarchy are reported to the testing workbench and collected into test run results, where they are available for use in reports and graphs. Each counter in the hierarchy has a type (defined in class StatType). The operations that are available on a counter depend on the counter's type.

## Test execution services interfaces

Interface	Description
ITestLogManager	Logs messages and verification points to the test log. Use this interface for handling error conditions, anomalies in expected data or other abstract conditions that need to be reported to users, or for comparisons or verifications whose outcome is reported to the test log. ITestLogManager can also convey informational or status messages after the completion of a test.
ITime	Defines basic time services, such as the current system time in milliseconds (adjusted so that all systems are synchronized with the schedule controller), the time the test begins, and the elapsed time from the beginning of the test.
ITransaction	Provides support for transactions. A collection of named transactions is maintained for each virtual user. Transactions created in custom code can be started and stopped wherever custom code can be used. These transactions can span several tests. Performance counters are kept for custom code transactions and appear in reports. An example of how you could use ITransaction is to create transactions for one virtual user but not another, to help verify responses from tests.
IEngineInfo	Provides information about the testing execution engine; for example, the number of virtual users running in this engine, the number of virtual users that have completed, the local directory in which test assets are deployed, and the host name of the computer on which the engine runs.
ITestInfo	Provides information about the test that is running; for example, the test name and information about the current problem determination log level for this test.
IVirtualUserInfo	Provides information about virtual users; for example, the virtual user's name, problem determination log level, TestLog level, globally unique ID, and user group name.
IScalar	Provides methods for simple integer performance counters. It is used for counters of SCALAR and STATIC types. Use this interface to decrement and increment counters.
IStat	Defines observational performance counters. It defines the method for submitting a data point to performance counters of type RATE, AVERAGE, and RANGE.



## Test execution services interfaces

Interface	Description
IStatistics	Retrieves the performance counter tree associated with the current statistics processor. Stops the delivery of performance counters. Changes the priority of the statistics delivery thread.
IStatTree	Provides methods that can retrieve child counters, create the XML fragments that define counters, and set the description field of counters.
IText	Contains text-based performance counters. Performance counters that do not fit any of the other counter types can be created as type TEXT. TEXT counters are not assigned definitions, but they are collected in the test results.

## Test execution services classes

Class	Description
DataAreaLockException	Throws an exception whenever an attempt is made to modify a locked DataArea key.
OutOfScopeException	Indicates that an object created by ITestExecutionServices has been referenced outside of its intended scope.
TransactionException	Throws an exception when a transaction is misused. The following conditions lead to a TransactionException exception: attempting to start a transaction that has already been started, attempting to stop a transaction that has not been started, and getting the start time or the elapsed time of a transaction that has not been started. Any operation (except abort()) on a transaction that has been aborted will throw a TransactionException exception.
StatType	Provides a list of valid performance counter types. The performance counter types are: AVERAGE, iAVERAGE, iRANGE, iRATE, iSCALAR, iSTATIC, iSTRUCTURE, iTEXT, RANGE, RATE, SCALAR, STATIC, STRUCTURE, and TEXT.

### Related information:

API references

### Reducing the performance impact of custom code

If custom code runs inside a page, it can affect that page's response time.

HTTP pages are containers of HTTP requests. On a given HTTP page, requests run in parallel across all of the connections between the agent computer and the system under test.

*Page response time* is the interval between *page start* and *page end*, which are defined as follows: Page start is the first timestamp associated with the client-server interaction. This interaction is either the first byte sent or the first connect of the first HTTP request. Page end is the last timestamp associated with the client-server interaction. This interaction is the last byte received of the last HTTP request to complete. Because of parallelism, the last HTTP request to complete might not be the last one listed for the page.

Typically, you should not insert custom code inside a page. While custom code that runs for only a few milliseconds should have little effect on page response time, the best practice is to place custom code outside a page. Custom code placed outside a page has no effect on page response time, and its execution time can overlap with think time delays.

Do not use custom code for data correlation if you can instead use the data correlation features built into the product. The built-in data correlation code takes advantage of requests running in parallel, whereas custom code actions do not begin until all earlier actions are completed.

You might need to place custom code inside a page to correlate a string from the response of a request inside that page to another request inside the same page. Even in this case, if you split the page into two pages, you can use the built-in data correlation features instead of custom code.

If you still want to run tests with custom code inside HTTP pages, use the Page Element report to evaluate performance. The Page Element report shows the response time and throughput for individual HTTP requests. Custom code does not affect the response time measurement of individual HTTP requests.

**Related concepts:**

Performance testing tips

**Custom code examples**

Custom code enables you to perform such tasks as managing loops, retrieving virtual user information, running external programs from tests, and customizing data correlation.

**Controlling loops:**

This example demonstrates extending test execution by using custom code to control loops. It provides sample code that shows how you can manipulate the behavior of loops within a test to better analyze and verify test results.

This example uses a recording of a stock purchase transaction using IBM's Trade application. The concepts shown here can be used in tests of other applications.

The test begins with a recording of a stock purchase transaction, using datapool substitution for the login IDs.

The pages are wrapped in a five-iteration loop, as shown in the following figure:

Notice that among the various pages of the test, three items of custom code exist (indicated by the green circles with "C"s in them). This example explores these items of custom code.

The first piece of custom code, `InitializeBuyTest`, is mentioned here:

```
package customcode;

import java.util.Random;

import com.ibm.rational.test.lt.kernel.IDataArea;
import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;
import com.ibm.rational.test.lt.kernel.services.IVirtualUserInfo;

/**
 * @author unknown
 */
public class InitializeBuyTest implements
    com.ibm.rational.test.lt.kernel.custom.ICustomCode2 {

    /**
     * Instances of this will be created using the no-arg constructor.
     */
    public InitializeBuyTest() {
    }

    /**
     * For description of ICustomCode2 and ITestExecutionServices interfaces,
     * see the Javadoc information. */
    public String exec(ITestExecutionServices tes, String[] args) {
        // Get the test's data area and set a flag indicating that nothing
        // has failed yet. This flag will be used later to break out
        // of the schedule loop as soon as a failure is encountered.
        IDataArea dataArea = tes.findDataArea(IDataArea.TEST);
        dataArea.put("failedYet", "false");

        // Get the virtual users's data area
        IDataArea vda = tes.findDataArea(IDataArea.VIRTUALUSER);

        // Randomly select a stock to purchase from the set of s:0 to s:499.
        IVirtualUserInfo vuInfo = (IVirtualUserInfo) vda.get(IVirtualUserInfo.KEY);
        Random rand = vuInfo.getRandom();
        String stock = "s:" + Integer.toString(rand.nextInt(499));

        // Persist the name of the stock in the virtual user's data area.
        vda.put("myStock", stock);

        return stock;
    }
}
```

This custom code is located in the method `exec()`.

First, the data area for the test is acquired to store a flag value, in this case a string of text, to be used later to stop the test loop when an error is discovered. Data stored in this way can be persisted across tests.

Then a randomly generated stock string is created. The value is stored as the variable *stock*, and is passed back as the return value for the method. This return value is used as a substitute in a request later, as shown in the following figure:

The highlighted item uses a substitution (s%3A716), which is the value returned by the InitializeBuyTest custom code item. We are using custom code to drive the direction of our test.

The next lines of code in InitializeBuyTest use the Virtual User data area to store the name of the stock for later reference. Again, data stored in this way can persist across tests.

The second piece of custom code is called CheckStock. Its contents are as follows (listing only the exec() method this time):

```
public String exec(ITestExecutionServices tes, String[] args) {  
  
    // Get the actual and requested stock purchased.  
    String actualStock = args[0].replaceAll("<B>", "");  
    actualStock = actualStock.substring(0, actualStock.indexOf("<"));  
    String requestedStock = args[1];  
  
    // Set the log level to ALL.  
    IDataArea dataArea = tes.findDataArea(IDataArea.TEST);  
    ITestInfo testInfo = (ITestInfo)dataArea.get(ITestInfo.KEY);  
    testInfo.setTestLogLevel(ITestLogManager.ALL);  
  
    // If the log level is set to ALL, report the actual and requested stock  
    // purchased.  
    ITestLogManager testLogManager = tes.getTestLogManager();  
    if (testLogManager.wouldReport(ITestLogManager.ALL)) {  
        testLogManager.reportMessage("Actual stock purchased: "  
            + actualStock + ". Requested stock: " + requestedStock  
            + ".");  
    }  
  
    // If the actual and requested stock don't match, submit a FAIL verdict.  
    if (testLogManager.wouldReport(ITestLogManager.ALL)) {  
        if (!actualStock.equalsIgnoreCase(requestedStock)) {  
            testLogManager.reportVerdict(  
                "Actual and requested purchase stock do not match.",  
                VerdictEvent.VERDICT_FAIL);  
  
            // Use the test's data area to record the fact that an error has  
            // occurred.  
            dataArea.put("failedYet", "true");  
        }  
    }  
    return null;  
}
```

This code begins by extracting two arguments that have been passed to the code. A part of the response in the original recording is highlighted and used as a reference, as shown in the following figure.

Some string manipulation is needed to acquire the text of interest; in this case, the name of the stock that was actually purchased. This newly created reference is then passed into CheckStock as an argument, as shown in the following figure:

Note that the return value of InitializeBuyTest is passed in as an argument as well.

The CheckStock custom code item uses these values to verify that the randomly chosen stock generated by InitializeBuyTest is actually purchased during the execution of the test.

CheckStock then sets the test log level, reports the actual and requested stock purchase, and raises a FAIL verdict if they do not match. CheckStock also stores a true value associated with the tag failedYet in the test's data area.

The third piece of custom code (exec() method only) is mentioned here:

```
public String exec(ITestExecutionServices tes, String[] args) {  
  
    // Get the test log manager.  
    ITestLogManager testLogManager = tes.getTestLogManager();  
  
    // Get the test's data area and get a flag indicating to  
    // see if anything has failed yet. If so, stop the loop.  
    IDataArea dataArea = tes.findDataArea(IDataArea.TEST);  
    String failedYet = (String) dataArea.get("failedYet");  
  
    // Break out of the loop if an error has been encountered.  
    if (failedYet.equalsIgnoreCase("true")) {  
        tes.getLoopControl().breakLoop();  
  
        if (testLogManager.wouldReport(ITestLogManager.ALL)) {  
            testLogManager.reportMessage("Loop stopped.");  
        }  
    }  
  
    return null;  
}
```

This code uses the test's data area to determine the user-defined value associated with the tag failedYet. If failedYet is true, StopLoopCheck breaks out of the test loop.

### Retrieving the IP address of a virtual user:

This example shows how to retrieve the local IP address of a virtual user. Retrieving IP addresses is particularly useful when virtual users are using IP aliases.

The following custom code retrieves the IP address that was assigned to a virtual user:

```
import java.net.InetAddress;  
import com.ibm.rational.test.lt.kernel.IDataArea;  
import com.ibm.rational.test.lt.kernel.services.ITestLogManager;  
import com.ibm.rational.test.lt.kernel.services.IVirtualUserInfo;  
  
public String exec(ITestExecutionServices tes, String[] args) {  
    IVirtualUserInfo vui = (IVirtualUserInfo) tes.findDataArea(IDataArea.VIRTUALUSER).get(IVirtualUserInfo);  
    ITestLogManager tlm = tes.getTestLogManager();  
  
    if (vui != null) {  
        String localAddr = null;  
        InetAddress ipAddr = vui.getIPAddress();  
        if (ipAddr != null)  
            localAddr = ipAddr.toString();  
        tlm.reportMessage("IPAlias address is " + (localAddr != null ? localAddr : "not set"));  
        return localAddr;  
    }  
}
```

```

    }
else
    return ("Virtual User Info not found");
}

```

### Printing input arguments to a file:

The PrintArgs class prints its input arguments to the file C:\arguments.out. This class could be used, for example, to print a response returned by the server.

```

package customcode;

import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;

import java.io.*;

/**
 * The PrintArgs class prints its input arguments to the file
 * C:\arguments.out. This example could be used to print a response
 * returned by the server.
 */

/**
 * @author IBM Custom Code Samples
 */

public class PrintArgs implements
    com.ibm.rational.test.lt.kernel.custom.ICustomCode2 {

    /**
     * Instances of this will be created using the no-arg constructor.
     */
    public PrintArgs() {
    }

    public String exec(ITestExecutionServices tes, String[] args) {
        try {
            FileWriter outFile = new FileWriter("C:\\arguments.out");
            for (int i = 0; i < args.length; i++)
                outFile.write("Argument " + i + " is: " + args[i] + "\n");
            outFile.close();
        } catch (IOException e) {
            tes.getTestLogManager().reportMessage(
                "Unable to write to C:\\arguments.out");
        }
        return null;
    }
}

```

### Counting the number of times that code is executed:

The CountAllIterations class counts the number of times code is executed by all virtual users. The CountUserIterations class counts the number of times code is executed by an individual virtual user.

The CountAllIterations class counts the number of times it is executed by all virtual users running in a particular JVM and returns this count as a string.

```

package customcode;

import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;

/**
 * The CountAllIterations class counts the number of times it is executed
 * by all virtual users running in a particular JVM and returns this count
 * as a string. If all virtual users on an agent are running in the same

```

```

* JVM (as would typically be the case), this class will count the number of
* times it is run on the agent.
*/

/**
 * @author IBM Custom Code Samples
 */

public class CountAllIterations implements
    com.ibm.rational.test.lt.kernel.custom.ICustomCode2 {
    private static int numJVMLoops = 0;

    /**
     * Instances of this will be created using the no-arg constructor.
     */
    public CountAllIterations() {
    }

    public String exec(ITestExecutionServices tes, String[] args) {
        return Integer.toString(++numJVMLoops);
    }
}

```

The CountUserIterations class counts the number of times code is executed by an individual virtual user.

```

package customcode;

import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;
import com.ibm.rational.test.lt.kernel.IDataArea;

/**
 * The CountUserIterations class counts the number of times it is executed
 * by an individual virtual user and returns this count as a string.
 */

/**
 * @author IBM Custom Code Samples
 */

public class CountUserIterations implements
    com.ibm.rational.test.lt.kernel.custom.ICustomCode2 {

    /**
     * Instances of this will be created using the no-arg constructor.
     */
    public CountUserIterations() {
    }

    public String exec(ITestExecutionServices tes, String[] args) {
        IDataArea userDataArea = tes.findDataArea(IDataArea.VIRTUALUSER);
        final String KEY = "NumberIterationsPerUser";

        Number numPerUser = (Number)userDataArea.get(KEY);
        if (numPerUser == null) {
            numPerUser = new Number();
            userDataArea.put(KEY, numPerUser);
        }

        numPerUser.value++;
        return Integer.toString(numPerUser.value);
    }

    private class Number {
        public int value = 0;
    }
}

```



## Setting and clearing cookies for a virtual user:

The SetCookieFixedValue class sets a Cookie for a virtual user, and the ClearCookies class clears all cookies for a virtual user.

The SetCookieFixedValue class sets a Cookie, defined in the newCookie variable, for a virtual user just as if the server had returned a Set-Cookie.

```
package customcode;

import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;
import com.ibm.rational.test.lt.execution.http.cookie.IHTTPVirtualUserInfo;
import com.ibm.rational.test.lt.kernel.IDataArea;

import java.text.ParseException;

/**
 * The SetCookieFixedValue class sets a Cookie, defined in the newCookie
 * variable, for a virtual user just as if the server had returned a Set-Cookie.
 */

/**
 * @author IBM Custom Code Samples
 */

public class SetCookieFixedValue implements
    com.ibm.rational.test.lt.kernel.custom.ICustomCode2 {

    /**
     * Instances of this will be created using the no-arg constructor.
     */
    public SetCookieFixedValue() {
    }

    public String exec(ITestExecutionServices tes, String[] args) {
        String newCookie = "MyCookie=CookieValue;path=/;domain=.ibm.com";
        IDataArea dataArea = tes.findDataArea(IDataArea.VIRTUALUSER);
        IHTTPVirtualUserInfo httpInfo =
            (IHTTPVirtualUserInfo)dataArea.get(IHTTPVirtualUserInfo.KEY);

        try {
            httpInfo.getCookieCache().setCookie(newCookie);
        } catch (ParseException e) {
            tes.getTestLogManager().reportMessage("Unable to parse Cookie " +
                newCookie);
        }

        return null;
    }
}
```

The ClearCookies class clears all Cookies for a virtual user. For information on how cookies are treated in tests and schedules, see\*\*\*\* MISSING FILE \*\*\*\*.

```
package customcode;

import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;
import com.ibm.rational.test.lt.execution.http.util.CookieCacheUtil;

/**
 * The ClearCookies class clears all Cookies for a virtual user.
 */

/**
 * @author IBM Custom Code Samples
 */
```

```

public class ClearCookies implements
    com.ibm.rational.test.lt.kernel.custom.ICustomCode2 {

    /**
     * Instances of this will be created using the no-arg constructor.
     */
    public ClearCookies() {
    }

    public String exec(ITestExecutionServices tes, String[] args) {
        CookieCacheUtil.clearCookieCache(tes);
        return null;
    }
}

```

### Determining where a test is running:

The ComputerSpecific class determines where a test is running  
package customcode;

```

import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;

import java.net.InetAddress;
import java.net.UnknownHostException;

/**
 * The ComputerSpecific class determined the hostname on which the test is
 * running, prints the hostname and IP address as a message in the test log,
 * and returns different strings based on the hostname.
 */

/**
 * @author IBM Custom Code Samples
 */

public class ComputerSpecific implements
    com.ibm.rational.test.lt.kernel.custom.ICustomCode2 {

    /**
     * Instances of this will be created using the no-arg constructor.
     */
    public ComputerSpecific() {
    }

    public String exec(ITestExecutionServices tes, String[] args) {
        String hostName = "Unknown";
        String hostAddress = "Unknown";

        try {
            hostName = InetAddress.getLocalHost().getHostName();
            hostAddress = InetAddress.getLocalHost().getHostAddress();
        } catch (UnknownHostException e) {
            tes.getTestLogManager().reportMessage(
                "Not able to obtain host information");
            return null;
        }
        tes.getTestLogManager().reportMessage("The hostname is " + hostName +
            "; IP address is " + hostAddress);

        if (hostName.equals("host-1234"))
            return "Special";
        else
            return "Normal";
    }
}

```

### Storing and retrieving variable values:

You can use the `getValue()` and `setValue()` methods to store and retrieve values in variables. Depending on the storage location that you specify, variables can be shared among tests, or stored locally in the current test.

You can use the `getValue()` and `setValue()` methods to store multiple values in variables in one custom code call. You can then create substitutions from variables instead of from multiple custom code elements.

For example, assume that a response contains three values: `id`, book title, and price. You can read all three values from the response, and then use custom code to set the variables `id`, `title`, and `price`. You can then substitute the values from the three variables as needed in the test, instead of having to write custom code for each variable.

**Note:** The storage location passed to the method must match the storage location used when declaring the variable.

```
package customcode;

import com.ibm.rational.test.lt.kernel.IDataArea;
import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;

/**
 * For Javadoc information on the ICustomCode2 and ITestExecutionServices interfaces,
 * see the 'Extending test execution with custom code' help topic.
 */

/**
 * @author IBM Custom Code Samples
 */

public String exec(ITestExecutionServices tes, String[] args) {

    tes.getValue("myVar", tes.STORAGE_USER); // This retrieves a value from a test for the va
    tes.getValue("myLocalVar", tes.STORAGE_LOCAL); // This variable is stored locally, per te

    tes.setValue("myVar", tes.STORAGE_USER, "myNewValue"); // Change the value of the variabl
    tes.setValue("myLocalVar", tes.STORAGE_LOCAL, "myLocalNewVar"); // Change the value of th
    return null;
}
```

### Extracting a string or token from its input argument:

The `ParseResponse` class extracts a string from its input argument. The `ExtractToken` class extracts a particular token (string) from its input argument. Both classes can be useful for handling certain types of dynamic data correlation.

The `ParseResponse` class extracts a string from its input argument, using a regular expression for pattern matching.

```
package customcode;

import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;

import java.util.regex.*;

/**
 * The ParseResponse class demonstrates using Custom Code to extract a
 * string from its input argument using a regular expression for pattern
 * matching.
 */
```

```

    * In this sample, the args[0] input string is assumed to be the full
    response from a previous request. This response contains the day's
    headlines in a format such as:
    *
    * <a class=f href=r/d2>In the News</a><small class=m>&nbsp;<span id=nw>
    * </span></small></h2>
    * <div class=ct>
    * &#149;&nbsp;<a href=s/213231>Cooler weather moving into eastern
    U.S.</a> * <br>&#149;&nbsp;<a href=s/262502>Digital camera shipments
    up</a><br> *
    * Given the above response, the extracted string would be:
    * Cooler weather moving into eastern U.S.
    */

/**
 * @author IBM Custom Code Samples
 */

public class ParseResponse implements
    com.ibm.rational.test.lt.kernel.custom.ICustomCode2 {

    /**
     * Instances of this will be created using the no-arg constructor.
     */
    public ParseResponse() {}

    public String exec(ITestExecutionServices tes, String[] args) {
        String HeadlineStr = "No Headline Available";
        String RegExpStr = ".*In the News[^;]*;[^;]*;[^;]*;<a
href=([>]*)>([<]*)<";
        Pattern pattern =
Pattern.compile(RegExpStr, Pattern.DOTALL);
        Matcher matcher =
pattern.matcher(args[0]);
        if (matcher.lookingAt())
            HeadlineStr = matcher.group(2);
        else
            tes.getTestLogManager().reportMessage("Input does not match
pattern.");
        return HeadlineStr;
    }
}

```

The ExtractToken class extracts a particular string from its input argument.

```

package customcode;

import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;

/**
 * The ExtractToken class demonstrates using Custom Code to extract a particular
 * token (string) from its input argument. This can be useful for handling
 * certain types of dynamic data correlation.
 *
 * In this sample, the args[0] input string is assumed to be comma-delimited
 * and the token of interest is the next-to-last token. For example, if
 * args[0] is:
 * javascript:parent.selectItem('1010','[Negative]1010','1010','', 'IBM',
 * '30181','Rational','1','null','1','1','6fd8e261','RPT')
 * the class will return the string 6fd8e261.
 */

/**
 * @author IBM Custom Code Samples
 */

public class ExtractToken implements
    com.ibm.rational.test.lt.kernel.custom.ICustomCode2 {

    public ExtractToken() {

```

```

    }

    public String exec(ITestExecutionServices tes, String[] args) {
        String ArgStr;
        String NextToLastStr;
        String[] Tokens = args[0].split(",");

        if (Tokens.length > 2) {
            ArgStr = Tokens[Tokens.length - 2];           // Extract next-to-last token

            // Remove enclosing ''
            NextToLastStr = ArgStr.substring(1, ArgStr.length() - 1);
        } else {
            tes.getTestLogManager().reportMessage("Could not extract value");
            NextToLastStr = null;
        }
        return NextToLastStr;
    }
}

```

### Retrieving the maximum JVM heap size:

The `JVM_Info` class retrieves the maximum heap size of the JVM.

```

package customcode;

import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;

import java.net.*;

/**
 * The JVM_Info class retrieves the maximum heap size of the JVM.
 * It writes a message in the test log with the hostname where the
 * JVM is running and the JVM's maximum heap size in megabytes.
 */

/**
 * @author IBM Custom Code Samples
 */

public class JVM_Info implements
    com.ibm.rational.test.lt.kernel.custom.ICustomCode2 {

    public JVM_Info() {
    }

    public String exec(ITestExecutionServices tes, String[] args) {
        Runtime rt = Runtime.getRuntime();
        long maxMB = rt.maxMemory()/(1024*1024); // maxMemory() size is in bytes
        String hostName = "Unknown";

        try {
            hostName = InetAddress.getLocalHost().getHostName();
        } catch (UnknownHostException e1) {
            tes.getTestLogManager().reportMessage("Can't get hostname");
            return null;
        }

        tes.getTestLogManager().reportMessage("JVM maximum heap size for host "
            + hostName + " is " + maxMB + " MB");

        return null;
    }
}

```

## Running an external program from a test:

The ExecTest class runs a program, defined in the execName variable, on the system where the test is running.

```
package customcode;

import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;
import com.ibm.rational.test.lt.kernel.services.ITestLogManager;
import org.eclipse.hyades.test.common.event.VerdictEvent;

import java.io.IOException;

/**
 * The ExecTest class runs a program, defined in the execName variable,
 * on the system where the test is running.
 * The test verdict is set to PASS if the program return code is 0.
 * The test verdict is set to FAIL if the program doesn't execute or
 * if the program return code is non-zero
 * In this sample, the program is perl.exe.
 */

/**
 * @author IBM Custom Code Samples
 */

public class ExecTest implements
    com.ibm.rational.test.lt.kernel.custom.ICustomCode2 {

    /**
     * Instances of this will be created using the no-arg constructor.
     */
    public ExecTest() {
    }

    public String exec(ITestExecutionServices tes, String[] args) {
        ITestLogManager logger = tes.getTestLogManager();
        int rtnval = 1;
        Process p = null;
        String execName = "C:/Windows/System32/perl.exe C:/Perl/true.pl";

        Runtime rt = Runtime.getRuntime();
        // Execute test
        try {
            p = rt.exec(execName);
        } catch (IOException e) {
            logger.reportMessage("Unable to run = " + execName);
            logger.reportVerdict("Execution of " + execName + " failed",
                VerdictEvent.VERDICT_FAIL);
            return null;
        }

        // Wait for the test to complete
        try {
            rtnval = p.waitFor();
            logger.reportMessage("Process return value is " +
                String.valueOf(rtnval));
        } catch (InterruptedException e1) {
            logger.reportMessage("Unable to wait for " + execName);
            logger.reportVerdict("WaitFor on " + execName + " failed",
                VerdictEvent.VERDICT_FAIL);
            return null;
        }

        // Check the test return code and set the test verdict appropriately
        if (rtnval != 0)
        {

```

```

        logger.reportVerdict("Execution failed", VerdictEvent.VERDICT_FAIL);
    } else {
        logger.reportVerdict("Execution passed", VerdictEvent.VERDICT_PASS);
    }

    return null;
}
}

```

### Adding custom counters to reports:

You can add custom counters to performance reports by using custom code. After running tests, the results from the custom counters are automatically aggregated in the same way that the default performance testing counters are (for example, byte and page counters). The aggregate for the custom counters is combined from all agent computers.

**Note:** Unless you place the custom counters under Run, Pages, or another root element, the Add/Remove Run Statistics Counters window will not contain information for the custom counters.

With the following code, you can add a custom counter. After running tests, you can display the custom counter on the report by dragging the custom counter from the results onto the report or by using the Add/Remove wizard.

```

package CustomCounter;

import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;

/**
 * @author unknown
 */
public class Class implements
    com.ibm.rational.test.lt.kernel.custom.ICustomCode2 {

    /**
     * Instances of this will be created using the no-arg constructor.
     */
    public Class() {
    }

    /**
     * For javadoc of ICustomCode2 and ITestExecutionServices interfaces, select 'Help Contents' in t
     * Help menu and select 'Extending Rational Performance Tester functionality' -> 'Extending test
     */
    public String exec(ITestExecutionServices tes, String[] args) {tes.getStatisticsManager().getStat
        .submitDataPoint(Double.valueOf(Math.random()*100.).longValue());

        return null;
    }
}
}

```

### Related tasks:

“Creating custom Java code” on page 45

### Using transactions and statistics:

You can use custom code to start transactions, gather additional statistics during a transaction, and stop a transaction.

The following code shows how to start a transaction. Transactions that are generated by test execution services automatically create and manage statistics.



```

package customcode;

import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;
import com.ibm.rational.test.lt.kernel.services.ITransaction;

/**
 * @author IBM Custom Code Samples
 */
public class BeginTransaction implements
    com.ibm.rational.test.lt.kernel.custom.ICustomCode2 {

    /**
     * Instances of this will be created using the no-arg constructor.
     */
    public BeginTransaction() {
    }

    /**
     * For Javadoc information on the ICustomCode2 and ITestExecutionServices interfaces,
     * see the 'Test execution services interfaces and classes' help topic.
     */
    public String exec(ITestExecutionServices tes, String[] args) {
        // the name of the transaction could have been passed in via data correlation mechanism.
        ITransaction foo = tes.getTransaction("foo");
        foo.start();
        return null;
    }
}

```

The following code shows how to gather additional statistics during a transaction.

```

package customcode;

import com.ibm.rational.test.lt.kernel.ITime;
import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;
import com.ibm.rational.test.lt.kernel.statistics.IScalar;
import com.ibm.rational.test.lt.kernel.statistics.IStat;
import com.ibm.rational.test.lt.kernel.statistics.IStatTree;
import com.ibm.rational.test.lt.kernel.statistics.impl.StatType;

/**
 * @author IBM Custom Code Samples
 */
public class BodyTransaction implements
    com.ibm.rational.test.lt.kernel.custom.ICustomCode2 {

    /**
     * Instances of this will be created using the no-arg constructor.
     */
    public BodyTransaction() {
    }

    /**
     * For Javadoc information on the ICustomCode2 and ITestExecutionServices interfaces,
     * see the 'Test execution services interfaces and classes' help topic.
     */
    public String exec(ITestExecutionServices tes, String[] args) {
        IStatTree tranStat;
        IStatTree timeStat;
        IStatTree countStat;

        IStat timeDataStat = null; // counter for the time RANGE
        IScalar countDataStat = null; // counter for the count SCALAR

        ITime timer = tes.getTime();

        IStatTree rootStat = tes.getStatisticsManager().getStatTree();
    }
}

```

```

    if (rootStat != null) {
        // these counters set up the hierarchy
        tranStat = rootStat.getStat("Transactions", StatType.STRUCTURE);
        timeStat = tranStat.getStat("Body Time", StatType.STRUCTURE);
        countStat = tranStat.getStat("Bocy Count", StatType.STRUCTURE);

        // the name of the counters could have been passed in via data correlation mechanism
        timeDataStat = (IStat) timeStat.getStat("foo", StatType.RANGE);
        countDataStat = (IScalar) countStat.getStat("foo", StatType.SCALAR);
    }

    // get the start time
    long startTime = timer.timeInTest();

    // do the work
    // whatever that work might be

    // get the end time
    long endTime = timer.timeInTest();

    // update timeDataStat with the elapsed time
    if (timeDataStat != null)
        timeDataStat.submitDataPoint(endTime - startTime);

    // update the countDataStat
    if (countDataStat != null)
        countDataStat.increment();

    return null;
}
}

```

The following code shows how to stop a transaction.

```

package customcode;

import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;
import com.ibm.rational.test.lt.kernel.services.ITransaction;

/**
 * @author IBM Custom Code Samples
 */
public class EndTransaction implements
    com.ibm.rational.test.lt.kernel.custom.ICustomCode2 {

    /**
     * Instances of this will be created using the no-arg constructor.
     */
    public EndTransaction() {
    }

    /**
     * For Javadoc information on the ICustomCode2 and ITestExecutionServices interfaces,
     * see the 'Test execution services interfaces and classes' help topic.
     */
    public String exec(ITestExecutionServices tes, String[] args) {
        // the name of the transaction could have been passed in via data correlation mechanism.
        ITransaction foo = tes.getTransaction("foo");
        foo.stop();
        return null;
    }
}
}

```

## Reporting custom verification point failures:

You can use custom code to report a custom verification point failure.

The following code shows how to report a custom verification point failure.

```
package customcode;

import org.eclipse.hyades.test.common.event.VerdictEvent;
import org.eclipse.hyades.test.common.runner.model.util.Verdict;

import com.ibm.rational.test.lt.execution.core.IVerificationPoint;
import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;

/**
 * @author IBM Custom Code Samples
 */
public class Class implements
    com.ibm.rational.test.lt.kernel.custom.ICustomCode2 {

    /**
     * Instances of this will be created using the no-arg constructor.
     */
    public Class() {
    }

    /**
     * For javadoc of ICustomCode2 and ITestExecutionServices interfaces, select 'Help Contents' in the
     * Help menu and select 'Extending Rational Performance Tester functionality' -> 'Extending test ex
     */
    public String exec(ITestExecutionServices tes, String[] args) {
        tes.getTestLogManager().reportVerificationPoint("CustomVP", VerdictEvent.VERDICT_FAIL);
        return null;
    }
}
```

## Debugging custom code:

This example demonstrates debugging custom code by adding a breakpoint. It provides sample code to add a breakpoint. This way of debugging custom code is applicable only for a schedule.

### Procedure

1. Start IBM Rational Performance Tester and create a performance test project MyProject.
2. Create an HTTP test, **MyTest**, by recording a visit to `http://<hostname>:7080/`.

**Note:** Before accessing the URL, ensure that Rational Performance Tester is running. The URL returns an HTTP 404 error, which is expected.

3. Expand the first request and click the response element.
4. In the Test Element Details section, right-click in the **Content** field and click **Create Field Reference**.
5. Type the reference name and click **OK**.
6. Click the first page, and then click **Add > Custom Code**.
7. In the **Arguments** section of Test Element Details, click **Add**.
8. Expand the data source for the search results page, select the reference name that you created in step 5, and click **Select**.

9. Click **Generate Code**. A new tab with the generated code is displayed.
10. Insert the following the code into the `exec()` method:

```
ITestLogManager history = tes.getTestLogManager();
if (args.length > 0) {
    if (args[0].indexOf("Investor Relations") != -1) {
        history.reportMessage("First page failed. Bail loop!");
        tes.getLoopControl().continueLoop();
    }
}
```

**Important:**

- Fix the double quotation marks, if any, so they are straight and the compiler no longer gives warning.
- To resolve compiler warnings related to importing a class, press **Ctrl + Shift + O**.

The code will look like this:

11. To set a breakpoint, click anywhere on the `args[0].indexOf` line. Move the pointer to the left-most portion of the text editor window and double-click with the pointer horizontally on the same line. A blue button is displayed in this left-most portion of the window indicating the breakpoint is set.

12. Save the custom code and then the test.

13. Create a new schedule, `Schttest`.

- a. In `Schttest`, set the number of users to run to 1.
- b. Click **User Group 1** and click **Add > Test**. Select the `MyTest` test and click **OK**.
- c. Click **User Group 1** and click the **Run this group on the following locations** button.
- d. Click **Add > Add New**.
- e. In the `New Location` window, type the following information:
  - 1) In **Host name**, type `localhost`.
  - 2) In **Name**, type `debuglocation`.
  - 3) In **Deployment directory**, type `C:\mydeploy`.
  - 4) Click **Finish**.
- f. Save the schedule.

14. In the `Test Navigator`, right-click `debuglocation` and click **Open**.

15. Click the **General Properties** tab and click **Add**.

16. In the **Property name** field, type `RPT_VMARGS` and in the **Property value** field, add the following values each separated by a space.

```
-Xdebug
-Xnoagent
-Djava.compiler=NONE
-Xrunjdpw:transport=dt_socket,server=y,suspend=y,address=8000
```

17. Save the location.

18. Attach the debugger to the schedule execution process.

- a. Run the schedule. Because the schedule is using **debuglocation**, it will pause at the beginning to let you attach the debugger to the execute process.

- b. Click **Window > Open Perspective > Other > Debug**.
- c. Click **Run > Debug Configurations**.
- d. In the Debug Configurations window, right-click **Remote Java Application** and click **New**.
- e. Click **Debug**. A list of running threads are displayed in the Debug window and the schedule execution pauses at the debug breakpoint.
- f. If you are doing it for the first time, you might need to provide the source location to see the custom Java code. You do this by taking the following steps:
  - 1) Click **Edit Source Lookup Path** and click **Add**.
  - 2) Click **Workspace Folder > OK**.
  - 3) Now, expand MyProject, select the src folder, and click **OK**. The schedule run stops at the specified breakpoint.

### **Migrating custom code from previous versions**

You can run scripts that contain custom code from previous releases and edit tests to make new calls to old or new custom code classes.

#### **About this task**

You can perform the following tasks without any additional steps:

- Run a script that contains custom code that was created in a previous release.
- Edit a test to make a new call to an old custom code class.
- Add new custom code to a test that contains old custom code.

To edit a class in existing custom code so that it can call new `TestExecutionServices` methods, type cast the `IKlog` argument in the old custom code to the `ITestExecutionServices` interface.

---

# Index

## A

- Add user action for this element 31
- Add user action in the test editor 26
- add user cation 26
- Android
  - configuring the mobile client 9
- android application 19
- application
  - preparing
    - Mobile 17

## C

- ClearCookies class 57
- Compound tests
  - adding tests 41
  - adding to Test Workbench
    - projects 44
  - creating 40
  - modifying 42
  - overview 39
  - running 42
  - viewing 40
- ComputerSpecific class 58
- configuring
  - Android mobile client 9
- cookies
  - setting and clearing for virtual users 57
- CountAllIterations class 55
- CountUserIterations class 55
- Custom code
  - debug 66
- custom counters
  - test execution services 63
- custom Java code
  - code execution counts 55
  - controlling loops 51
  - creating 45
  - custom counters 63
  - determining where a test is running 58
  - extracting strings 59
  - interfaces and classes 47
  - migrating 68
  - overview 45
  - performance 51
  - printing input arguments to a file 55
  - retrieving the maximum JVM heap size 61
  - retrieving virtual user IP address 54
  - running a program with a test 62
  - setting and clearing cookies for virtual users 57
  - statistics 63
  - transactions 63
  - using strings 59
  - verification points 66

## D

- data correlation
  - custom code example 59
- DataAreaLockException (test execution services) 47
- Debug custom code 66

## E

- ExecTest class 62

## F

- files
  - printing input arguments to 55

## I

- IARM (test execution services) 47
- ICustomCode2 (test execution services) 47
- IDataArea (test execution services) 47
- IEngineInfo (test execution services) 47
- ILoopControl (test execution services) 47
- instrument 19
- IP addresses
  - retrieving from virtual user 54
- IPDLogManager (test execution services) 47
- IScalar (test execution services) 47
- IStat (test execution services) 47
- IStatistics (test execution services) 47
- IStatisticsManager (test execution services) 47
- IStatTree (test execution services) 47
- ITestExecutionServices (test execution services) 47
- ITestInfo (test execution services) 47
- ITestLogManager (test execution services) 47
- IText (test execution services) 47
- ITime (test execution services) 47
- ITransaction (test execution services) 47
- IVirtualUserInfo (test execution services) 47

## J

- Java
  - test execution services 45
- JVM heap size
  - retrieving maximum 61
- JVM\_Info class 61

## L

- loops
  - controlling 51

## M

- migration
  - custom Java code 68
- mobile
  - testing 1
- Mobile
  - application editor overview 17
  - test editor overview 24
- Mobile client
  - Android configuration 9
  - mobile data 31
  - modify step target 31

## O

- OutOfScopeException (test execution services) 47

## P

- ParseResponse class 59
- PrintArgs class 55

## S

- SetCookieFixedValue class 57
- StatType (test execution services) 47
- strings
  - extracting from input arguments 59
  - managing 59

## T

- test execution services
  - code execution counts 55
  - custom counters 63
  - determining where a test is running 58
  - extracting strings 59
  - interfaces and classes 47
  - migrating Java code 68
  - overview 45
  - printing input arguments to a file 55
  - retrieving the maximum JVM heap size 61
  - retrieving virtual user IP address 54
  - running a program with a test 62
  - setting and clearing cookies for virtual users 57
  - statistics 63
  - transactions 63
  - using strings 59
  - verification points 66
- testing
  - mobile 19
    - overview 1
  - shell-sharing 19
- tests
  - adding custom Java code 45

- tests (*continued*)
  - customizing 47
  - declaring variables 28
  - editing
    - Mobile 24
  - extending
    - controlling loops 51
    - custom Java code 45
  - migrating custom Java code 68
  - test execution services
    - code execution counts 55
    - custom counters 63
    - determining where a test is running 58
    - extracting strings 59

- tests (*continued*)
  - test execution services (*continued*)
    - printing input arguments to a file 55
    - retrieving the maximum JVM heap size 61
    - retrieving virtual user IP address 54
    - running a program with a test 62
    - setting and clearing cookies for virtual users 57
    - statistics 63
    - transactions 63
    - using strings 59
  - tokens
    - extracting from input arguments 59

- TransactionException
  - test execution services 47

## V

- variables
  - assigning 28
  - Mobile/web object property 28
- virtual users
  - counting code runs 55
  - retrieving IP addresses 54
  - setting and clearing cookies 57



---

## Chapter 7. 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.

#### WLC1ient JavaScript client library

This collection of topics lists the public methods of the IBM Worklight runtime client API for mobile apps, desktop, and web.

WLC1ient is a JavaScript client library that provides access to IBM Worklight capabilities. You can use WLC1ient 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 (deprecated in IBM Worklight V6.0.0), 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:

WLC1ient 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.

WLC1ient 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

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 509, “WL.Client.reloadApp” on page 518
  - Connectivity: “WL.Client.setHeartBeatInterval” on page 519, “WL.Client.connect” on page 503, Connectivity-related JavaScript Events
  - Session management methods: “WL.Client.getUserName” on page 507, “WL.Client.getLoginName” on page 506, “WL.Client.login” on page 517, “WL.Client.logout” on page 517, “WL.Client.isUserAuthenticated” on page 516, “WL.Client.getUserInfo” on page 507, “WL.Client.updateUserInfo” on page 520
  - Data access methods: “WL.Client.invokeProcedure” on page 513
  - Activity logging methods: “WL.Client.logActivity” on page 516
  - User preference methods: “WL.Client.setUserPref” on page 519, “WL.Client.getUserPref(key)” on page 508
  - Application properties methods: “WL.Client.getEnvironment” on page 505, “WL.Client.getAppProperty” on page 504
  - Error handling: “WL.App.getErrorMessage” on page 499
  - Debugging: “The WL.Logger object” on page 573
- Mobile functionality and UI
  - Push notification API: “WL.Client.Push.isPushSupported” on page 592, “WL.Client.Push.isPushSMSSupported” on page 592, “WL.Client.Push.onReadyToSubscribe” on page 593, “WL.Client.Push.registerEventSourceCallback” on page 594, “WL.Client.Push.subscribe” on page 595, “WL.Client.Push.subscribeSMS” on page 595, “WL.Client.Push.unsubscribe” on page 596, “WL.Client.Push.unsubscribeSMS” on page 597
  - Network details: “WL.Device.getNetworkInfo” on page 521
  - Opening a URL: “WL.App.openURL” on page 499
  - Options menu: WL.OptionsMenu
  - Tab bar: Tab Bar API
  - Badge: “WL.Badge.setNumber” on page 501
  - Toast: “WL.Toast.show” on page 524

- Globalization: “WL.App.getDeviceLocale” on page 498, “WL.App.getDeviceLanguage” on page 498
- Back button: “WL.App.overrideBackButton” on page 500, “WL.App.resetBackButton” on page 501
- Dialog box: “WL.SimpleDialog” on page 607
- Busy indicator: “WL.BusyIndicator (constructor)” on page 524
- Closing an app: “WL.App.close” on page 497: Deprecated
- Accessing native pages on mobile apps: “WL.NativePage.show” on page 523
- Switching between HTML Pages: “WL.Fragment.load” on page 610, “Class WL.Page” on page 611: Both deprecated
- Encrypted offline cache: WL.EncryptedCache
- Clipboard: “WL.App.copyToClipboard” on page 498
- Web and desktop widget methods
  - Desktop window state: “WL.Client.close” on page 502, “WL.Client.minimize” on page 517
  - Globalization: “WL.Client.getLanguage” on page 531
- Mechanisms used by the WLClient methods
  - “The options object” on page 597, Timeout

## WL.Analytics

The IBM Worklight Analytics API provides the ability to enable, disable, reset, and log analytics data during IBM Worklight Application runtime.

You can enable or disable WL.Analytics by default on the `wlInitOptions` object, by creating the `analytics` object in it. The `analytics` object has the boolean variable `enabled`, and optionally, the string variable `url`. The event `WL/ANALYTICS/READY` is triggered when WL.Analytics is enabled. WL.Analytics cannot be used before this event is triggered, or before using `WL.Analytics.enable([options])`.

**Note:** No data is sent to the IBM Worklight server until the application is successfully connected to the server. You can make this connection by setting `connectOnStartup : true` in the `wlInitOptions` object in `initOptions.js`, which is found in `[app folder]/common/js`, otherwise this is done automatically on the first successful call to an adapter in the Worklight server.

### WL.Analytics.disable:

Disables the capture of analytics data. Any data logged while WL.Analytics is disabled will be lost.

#### Syntax

```
Promise WL.Analytics.disable()
```

#### Returns

Promise: Resolved with no parameters, rejected with an error object.

#### Example

```
WL.Analytics.disable()

.then(function () {
  //Capture of Analytics data is fully disabled.
})
```

```
.fail(function (errObj) {
  //errObj.src = function that failed
  //errObj.msg = error message
});
```

### **WL.Analytics.enable:**

Turns on the capture of analytics data. The Promise returned by enable must be resolved prior to any WL.Analytics API call.

The options object can contain the url key with a string value pointing to the server to which Tealeaf will POST its collected analytics data. If no URL is specified (meaning url is not defined, or url is an empty string), it defaults to the Worklight server.

If enable is called with a different URL than the one currently set, analytics data collected for the previous URL is discarded and not sent to the new server.

When enabled, any crash of the application is logged and sent to the server when the app runs again. The information that is logged for the crash is the stack trace together with the IBM Worklight app name and version, and the environment in which the app was running.

### **Syntax**

Promise WL.Analytics.enable([options])

### **Parameters**

Parameter	Description
options	Object (Optional)

### **Returns**

Promise: Resolved with no parameters, rejected with an error object.

### **Example**

```
WL.Analytics.enable({url: 'http://example.org'})

.then(function () {
  //Capture of Analytics data is fully enabled.
  //Data will be sent to 'http://example.org'
})

.fail(function (errObj) {
  //errObj.src = function that failed
  //errObj.res = error message
});
```

### **WL.Analytics.log:**

Logs a message with IBM Worklight contextual data, then is added to a Tealeaf queue.

This queue will be flushed every ten WL.Analytics.log calls, and every time the app goes to the background. If flush queue fails to contact the server, an error object is passed as the second parameter of the success callback. The first parameter is always the same, see return values.

The first parameter is the JSON object that you want to log as the message, and the second parameter is an optional name for this particular event being logged. The contextual data provided by IBM Worklight is the application name and version, the environment in which the application is running, and the GPS and/or network information, if it is available, and if all required permissions are previously set for the application.

### Syntax

```
Promise WL.Analytics.log(msg, [name])
```

### Parameters

Parameter	Description
<code>msg</code>	Object (Message)
<code>[name]</code>	String (Optional, name of the message)

### Returns

Promise: Resolved with true if the queue was flushed and false if the queue was not flushed. Rejected with an error object.

### Example

```
WL.Analytics.log({data: [1,2,3]});  
//or  
WL.Analytics.log({data: [1,2,3]}, 'MyData');  
  
//Checking if the queue was flushed  
WL.Analytics.log({hello: 'world'})  
  
.always(function (wasQueueFlushed, errObj) {  
    if (wasQueueFlushed) {  
        WL.Logger.debug('The queue was flushed');  
    } else {  
        WL.Logger.debug('The queue was NOT flushed');  
    }  
    if (typeof errObj === 'object') {  
        WL.Logger.debug('Error trying to flush the queue', errObj);  
    }  
});
```

### WL.Analytics.restart:

Disables analytics capture, resets the Tealeaf configuration settings to their default value, and re-enables the capture of analytics data.

`WL.Analytics.restart` can be called with an options object that has the same format as the one passed to `WL.Analytics.enable`. Any data in the queue is discarded. In Android, it resets the configuration to what is set in the `TLFConfigurableItems.properties` file. In iOS, it resets it to the values in `TLFConfigurableItems.plist`. The URL is set to whatever is specified in `options.url`, or if none is specified, the URL points to Worklight Server, like `WL.Analytics.enable` does.

## Syntax

Promise WL.Analytics.restart([options])

## Parameters

Parameter	Description
<b>options</b>	Object. Has the same properties as the one passed to WL.Analytics.enable

## Returns

Promise: Resolved with no parameters, rejected with an error object.

## Example

```
WL.Analytics.restart({url: 'http://example.org'})

.then(function () {
  //Tealeaf's configuration is reset to its default values
  //Capture of Analytics data is fully enabled.
  //Data will be sent to 'http://example.org'
})

.fail(function (errObj) {
  //errObj.src = function that failed
  //errObj.msg = error message
})
```

## WL.Analytics.state:

Contains keys defining the analytics state.

The state object is kept by WL.Analytics and contains the following keys:

### **enabled (boolean)**

Value is true if Tealeaf is enabled, false otherwise.

### **url (string)**

Value is the URL Tealeaf sends messages queued with WL.Analytics.log

### **currentQueueSize (integer)**

Value is the number of messages in the queue.

### **resumeEventAttached (boolean)**

Value is true if the queue is flushed on resume.

### **lastUpdate (date)**

Last time the server was accessible and log data was sent. Null if the server has never been reached.

Changing the state object that is returned does not affect the state object that is kept internally.

## Syntax

Object WL.Analytics.state()

## Returns

Object: Copy of the state object

### Example

```
WL.Analytics.state();  
//{enabled: true, url: 'http://example.org', currentQueueSize: 1, resumeEventAttached: true}
```

### **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 75. *WL.App.BackgroundHandler.setOnAppEnteringBackground* parameters

Parameter	Description
<b>handler</b>	<p>Function. The function that is called upon receiving the event from iOS that the application is about to enter background. Values:</p> <p><b>WL.App.BackgroundHandler.defaultIOSBehavior</b> Use the default behavior of iOS (which is equivalent to not doing anything).</p> <p><b>WL.App.BackgroundHandler.hideView</b> 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:</p> <ul style="list-style-type: none"> <li>• Do not call <code>setOnAppEnteringForeground</code>: this option automatically shows the application after it returns from the background.</li> <li>• Call <code>setOnAppEnteringForeground</code>, but customize what the application does when it returns from the background. Return <code>WL.App.BackgroundHandler.hideViewToForeground()</code> at the end of the function. For an example, see “<code>WL.App.BackgroundHandler.setOnAppEnteringForeground</code>” on page 497.</li> </ul> <p><b>WL.App.BackgroundHandler.hideElements (deprecated)</b> Hide all HTML elements that have the style <code>WLHideOnEnteringBackground</code>.  This function is deprecated since IBM Worklight V6.0.0. Instead, use <b>WL.App.BackgroundHandler.hideView</b> to hide the current view.</p> <p><b>Custom function</b></p>

## Examples

Example: Use **hideView**

```
// CSS
<span id="moneyInTheBank" class="WLHideOnEnteringBackground">
  ...
</span>
```

```
// JavaScript
WL.App.BackgroundHandler.setOnAppEnteringBackground(
    WL.App.BackgroundHandler.hideView
);
```

Example: Use custom function myFunc

```
// 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 76. 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 495.)

```
WL.App.BackgroundHandler.setOnAppEnteringForeground( function () {
    alert("This is my custom code");
    return WL.App.BackgroundHandler.hideViewToForeground();
});
```

### WL.App.close

Deprecated. 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.

## 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 77. WL.App.copyToClipboard parameters

Parameter	Description
<b>string</b>	Mandatory. String. The text to be copied to the clipboard
<b>callback</b>	Optional. For Android environments only. The callback function that is called after the data is copied to the clipboard.

## 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 78. WL.App.getErrorMessage parameters

Parameter	Description
<b>exception</b>	Mandatory. The exception object from which the error string is extracted.

## WL.App.openURL

Open a URL. The behavior depends on the application platform.

### 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 79. WL.App.openURL application environments and behavior

Environment	Description
Adobe AIR	Opens a new browser window at the specified URL. The <b>target</b> and <b>options</b> parameters are ignored.
Android	Replaces the application with a new default browser window at the specified URL. The <b>target</b> and <b>options</b> parameters are ignored. The application is not closed; pressing Back on the phone brings the user back to the application.
BlackBerry 6 and 7	Replaces the application with a new default browser window at the specified URL. The <b>target</b> and <b>options</b> parameters are ignored.
iPhone, iPad	Replaces the application with a new Safari window at the specified URL. The <b>target</b> and <b>options</b> parameters are ignored.

Table 79. *WL.App.openURL* application environments and behavior (continued)

Environment	Description
Mobile web apps	Opens a new browser window at the specified URL. Whether the <b>target</b> and <b>options</b> parameters are ignored or not depends on the specific mobile browser.
Windows Phone 8	Replaces the application with a new Internet Explorer window at the specified URL. The <b>target</b> and <b>options</b> parameters are ignored.
Windows Phone 7.5 (deprecated in IBM Worklight V6.0.0)	Replaces the application with a new Internet Explorer window at the specified URL. The <b>target</b> and <b>options</b> parameters are ignored.
Windows 8	Replaces the application with a new Internet Explorer window at the specified URL. The <b>target</b> and <b>options</b> parameters are ignored.
Other environments	If the value of the <b>target</b> parameter is <code>_self</code> or unspecified, replaces the application iframe with the specified URL. Otherwise, opens a new browser window with the specified URL. The <b>target</b> and <b>options</b> parameters are NOT ignored.

## Parameters

Table 80. *WL.App.openURL* parameters

Parameter	Description
<b>url</b>	Mandatory. The URL of the web page to be opened.
<b>target</b>	Optional. The value to be used as the target (or name) parameter of the JavaScript <code>window.open</code> method. If no value is specified, <code>_self</code> is used.
<b>options</b>	Optional. The value to be used as the options (or specs) parameter of the JavaScript <code>window.open</code> method.  If no value is specified, the following options are used:  <code>status=1, toolbar=1, location=1, menubar=1, directories=1, resizable=1, scrollbars=1</code>

## 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 (deprecated in IBM Worklight V6.0.0), 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 81. *WL.App.overrideBackButton* parameters

Parameter	Description
<code>callback</code>	Mandatory. Function. The function that is called when <b>Back</b> is pressed.

## Return Value

None

```
WL.App.overrideBackButton(backFunc);
function backFunc(){
  alert('you hit the back key!');
}
```

## WL.App.resetBackButton

Resets the original Back button behavior.

**Note:** This method applies to Android, Windows Phone 7.5 (deprecated in IBM Worklight V6.0.0), 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 82. *WL.Badge.setNumber* parameters

Parameter	Description
<b>number</b>	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)
```

## 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
<b>headerName</b>	Mandatory. The name of the header to be added..
<b>headerValue</b>	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.

## Parameters

Parameter	Description
<b>options</b>	<p>Optional. A JSON block with the following additional properties:</p> <p><b>onSuccess</b> A callback function invoked when the connection to the Worklight Server is established</p> <p><b>onFailure</b> A callback function invoked when the WL.Client.connect method fails to establish connection with the Worklight server. The callback receives one parameter of type WL.FailureResponse, which might be null if connect is called while a previous call to connect has not yet returned.</p> <p><b>timeout</b> Integer. Number of milliseconds to wait for the server response before failing with a request timeout.</p>

## 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 83. Parameters for user preference method WL.Client.deleteUserPref*

Parameter	Description
<b>key</b>	Mandatory. The user preference key. Can be up to 128 characters long.
<b>options</b>	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.

## Parameters

Table 84. Values returned for a specified property

Property	Description
<b>property</b>	<p>Mandatory. One of the following values:</p> <p><b>WL.AppProperty.AIR_ICON_16x16_PATH</b> For AIR widgets only; the relative path to the AIR icon.</p> <p><b>WL.AppProperty.AIR_ICON_128x128_PATH</b> For AIR widgets only; the relative path to the AIR icon.</p> <p><b>WL.AppProperty.DOWNLOAD_APP_LINK</b> For desktop widgets only; the URL for downloading an updated version of the application.</p> <p><b>WL.AppProperty.APP_DISPLAY_NAME</b> The application display name, as defined in the application descriptor.</p> <p><b>WL.AppProperty.APP_LOGIN_TYPE</b> The application login type, as defined in the application descriptor: never, onstartup, or ondemand</p> <p><b>WL.AppProperty.APP_VERSION</b> The application version, as defined in the application descriptor (a newer version might be available on the Worklight Server)</p> <p><b>WL.AppProperty.LATEST_VERSION</b> The latest application version available on the Worklight Server.</p> <p><b>WL.AppProperty.MAIN_FILE_PATH</b> For web environments only; the absolute URL to the main application file.</p> <p><b>WL.AppProperty.SHOW_IN_TASKBAR</b> For AIR widgets only; a Boolean stating whether the Air application shows in the taskbar, as defined in the descriptor</p> <p><b>WL.AppProperty.THUMBNAIL_IMAGE_URL</b> An absolute URL for the thumbnail image for the application</p>

### 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, Android, or Windows.

## 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.IPAD`
- `WL.Environment.IPHONE`
- `WL.Environment.MOBILE_WEB`
- `WL.Environment.PREVIEW` (when the application runs in Preview mode)
- `WL.Environment.WINDOWS_PHONE_8`
- `WL.Environment.WINDOWS_PHONE` (Windows Phone 7.5 - deprecated in IBM Worklight V6.0.0)
- `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.

### **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 85. Parameters for session management method `WL.Client.getLoginName`

Parameter	Description
<code>realm</code>	Optional. The name of a realm that is defined in the <code>authenticationConfig.xml</code> file.  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
<code>realm</code>	Mandatory. The name of a realm that is defined in the <code>authenticationConfig.xml</code> file.
<code>key</code>	Mandatory. The name of the key that is present in the specified realm.

## `WL.Client.getUserName`

This method returns the user name of the user who is currently logged in.

### 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
<code>realm</code>	Optional. The name of a realm defined in the <code>authenticationConfig.xml</code> file.  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 86. Parameters for user preference method `WL.Client.getUserPref(key)`

Parameter	Description
<code>key</code>	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.

### **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 87. Parameters for user preference method `WL.Client.hasUserPref`

Parameter	Description
<code>key</code>	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 597, augmented with the following optional properties:

Property	Description
<b>Timeout</b>	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.
<b>enableLogger</b>	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. <b>Note:</b> 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 573.
<b>messages</b>	A dictionary object for localizing texts, in the messages.js file. If not specified, the default object Messages (in the same file) is used.
<b>authenticator</b>	An object that implements the Authenticator API. If not specified, Authenticator is used.



Property	Description
<b>heartBeatIntervalInSecs</b>	An integer value that denotes the interval in seconds between heartbeat messages that are automatically sent by WLCClient to the Worklight Server. The default value is 420 (7 minutes).
<b>connectOnStartup</b>	<p>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 <code>initOptions.js</code> file is false.</p> <p>The value <code>false</code> 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.</p> <p>The value <code>true</code> is appropriate if your app must receive data from the server when it starts. However, the app might start more slowly.</p>
<b>onConnectionFailure</b>	A failure handling function that is invoked when connection to the Worklight Server, performed on initialization by default, or if the <code>connectOnStartup</code> flag is true, fails.
<b>onUnsupportedVersion</b>	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 597.
<b>onRequestTimeout</b>	A failure handling function that is invoked when the <code>init()</code> request times out. For more information about the signature of failure handling functions, see "The options object" on page 597.
<b>onUnsupportedBrowser</b>	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 597.
<b>onDisabledCookies</b>	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 597.
<b>onUserInstanceAccessViolation</b>	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 597.

Property	Description
<b>onErrorRemoteDisableDenial</b>	<p>A failure-handling function 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 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).</p> <p>You can add the following parameters:</p> <p><b>message</b> 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.</p> <p><b>downloadLink</b> 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.</p> <p>Example:</p> <pre>var wlInitOptions = {   connectOnStartup : true,   onErrorRemoteDisableDenial : function (message, downloadLink) {     WL.SimpleDialog.show(       "Application Disabled",       message,       [{text: "Close application", handler: function() {         // Close application       }},       {text: "Download new version", handler: function() {         // Download new version       }}     );   } };</pre>
<b>onErrorAppVersionAccessDenial</b>	<p>A failure-handling function 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 defined in the IBM Worklight Console.</p> <p><b>Note:</b> <code>onErrorAppVersionAccessDenial</code> is deprecated since V5.0.6. Instead, use <code>onErrorRemoteDisableDenial</code>.</p>
<b>validateArguments</b>	<p>A Boolean value, indicating whether the IBM Worklight Client runtime library validates the number and type of method parameters. Default is true.</p>

Property	Description
<b>updateSilently</b>	A Boolean value, indicating whether Direct Update is performed without notifying the user before downloading new application resources. Default is false.
<b>onGetCustomDeviceProvisioningProperties</b>	<p>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.</p> <p>The function receives a <b>resumeDeviceProvisioningProcess</b> argument, which must be called to resume the provisioning process, and transfers the custom provisioning data as a JSON hash map.</p> <p>Example:</p> <pre data-bbox="933 756 1425 934"> In initOptions.js: var w1InitOptions = {   ...   ...   onGetCustomDeviceProvisioningProperties: collectCustomPr   ... } </pre> <pre data-bbox="933 966 1425 1197"> In application JavaScript file: function collectCustomProvisioningProperties (   resumeDeviceProvisioningProcess) {   // Collect provisioning token from user resumeDevicePr   {     token: token   } }; } </pre>
<b>showCloseOnDirectUpdateFailure</b>	A Boolean value, indicating whether the <b>Close</b> 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 <b>Close</b> 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.

Property	Description
<b>showCloseOnRemoteDisableDenial</b>	<p>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 <b>Get new version</b> button and a <b>Close</b> button. Clicking <b>Close</b> closes the dialog, but allows the user to continue working offline, with no connection to the Worklight Server.</p> <p>If you set the value to false, the behavior is as follows:</p> <ul style="list-style-type: none"> <li>• If you disable the application on the Worklight Console and specify a link to the new version, the dialog displays only the <b>Get new version</b> button. The <b>Close</b> button is not shown. The user has no choice but to update the application, and the user is forced to exit the application.</li> <li>• If you disable the application and do not specify a link to a new version, the dialog displays only the <b>Close</b> button.</li> </ul> <p>The default is true.</p>

**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
<b>invocationData</b>	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 514.

Parameter	Description
<b>options</b>	<p>Optional. A standard options object, as defined in “The options object” on page 597, augmented with the following property:</p> <ul style="list-style-type: none"> <li>• <b>timeout</b>: Integer. Number of milliseconds to wait for the server response before failing with a request timeout.</li> </ul> <p>The success handler of this call receives an augmented response that is described in “The WL.Client.invokeProcedure Success Handler Response Object” on page 515</p> <p>The failure handler of this call is called in two cases:</p> <ul style="list-style-type: none"> <li>• The procedure was called but failed. In this case, the <code>invocationResult</code> property is added to the response received by the failure handler. This property has the same structure as the <code>invocationResult</code> property returned to the success handler, but the value of the <code>isSuccessful</code> attribute is false. For the structure of the <code>invocationResult</code> property, see <code>invocationResult</code>.</li> <li>• A technical failure resulted in the procedure not being called. In this case, the failure handler receives a standard response object.</li> </ul>

### 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: [],
  compressResponse: true/false
}
```

The JSON block contains the following properties:

Property	Description
<b>adapter</b>	Mandatory. A string that contains the name of the adapter as specified when the adapter was defined.
<b>procedure</b>	Mandatory. A string that contains the name of the procedure as specified when the adapter was defined.
<b>parameters</b>	Optional. An array of parameters that is passed to the back-end procedure.

Property	Description
<b>compressResponse</b>	Optional. A string that requests the response from the server to be sent in a compressed format to reduce the amount of data that is transferred between Worklight Server and the device. The default value, if compressResponse is not specified, is false. <b>Note:</b> This option is applicable for Android, iOS, Windows Phone 8, BlackBerry 10, Mobile Web, and Adobe AIR. For Mobile Web applications, compression is supported only when the device browser can decompress GZIP data.

### The WL.Client.invokeProcedure Success Handler Response Object

The success handler response object can contain the following properties:

Property	Description
<b>invocationContext</b>	The invocationContext object that was originally passed to the Worklight Server in the callback object.
<b>invocationResult</b>	An object that contains the data that is returned by the invoked procedure, and the invocation status. Its format is as follows: <pre>invocationResult = {   isSuccessful: Boolean,   errors: "Error Message"   // Procedure results go here }</pre> <p>Where:</p> <ul style="list-style-type: none"> <li>• isSuccessful – Contains true if the procedure invocation succeeded, false otherwise. If the invocation failed, the failure handler for the request is called.</li> <li>• errors – An optional string array that contains the error messages.</li> </ul>

### Return Value

None.

### WL.Client.isConnected (Deprecated)

This method is deprecated.

### Syntax

```
WL.Client.isConnected()
```

### Description

**Note:** This method is deprecated since IBM Worklight V4.1.3. 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.

#### Parameters

Parameter	Description
<b>realm</b>	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

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 861.

**Note:** To ensure that the activity is stored in the database, set **reports.exportRawData** to true in the worklight.properties file. For information about how to specify IBM Worklight configuration properties, see "Configuration of IBM Worklight applications on the server" on page 714.

#### Parameters

Parameter	Description
<b>activityType</b>	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.

## Parameters

Parameter	Description
<code>realm</code>	Mandatory. A realm that defines how the login process is performed. The realm is the one defined in the application descriptor.
<code>options</code>	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
<code>realm</code>	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.
<code>options</code>	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.

## Parameters

None.

## 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.

## 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
<code>headerName</code>	Mandatory. The name of the header to be removed..

## Return Value

None.

## Example

```
WL.Client.removeGlobalHeader("MyCustomHeader");
```

## 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 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
<b>interval</b>	Mandatory. An integer value, denoting the interval in seconds between heartbeat messages automatically sent by <code>WLClient</code> to the Worklight Server.  An interval value of -1 disables the heartbeat: <code>WL.Client.setHeartBeatInterval(-1)</code>

## 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
<b>key</b>	Mandatory. The user preference key. Can be up to 128 characters long.
<b>value</b>	Mandatory. The value of the user preference. Can be up to 3072 characters long.
<b>options</b>	Optional. A standard options object.

## 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 action 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
<b>{key1:value1, key2:value2, ...}</b>	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.
<b>options</b>	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)`
- `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
<code>options</code>	Optional. A standard <code>options</code> object.

## Return Value

None.

## 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.

## Parameters

Table 88. *WL.Device.getNetworkInfo* parameters

Parameter	Description
<code>callback</code>	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 89. 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 WiFi 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 WiFi 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
ipAddress	<p>Mandatory. IP address of the device</p> <p><b>Note:</b> The value <b>ipAddress</b> is set to the first non-null value of the possible four IP addresses in this order:</p> <ul style="list-style-type: none"> <li>• IPv4 WiFi address</li> <li>• IPv4 3G address</li> <li>• IPv6 WiFi address</li> <li>• IPv6 3G address</li> </ul> <p>For example, if both IPv4 addresses are not present, the value <b>ipAddress</b> takes the value of the IPv6 WiFi address.</p>	Yes	Yes

Ipv4Addresses	Optional array that contains key value pairs: <ul style="list-style-type: none"> <li>wifiAddress - The IPv4 WiFi address if it is present</li> <li>3GAddress - The IPv4 3G address if it is present</li> </ul>	Yes	Yes
Ipv6Addresses	Optional array that contains key value pairs: <ul style="list-style-type: none"> <li>wifiAddress - The IPv6 WiFi address if it is present (only on iOS)</li> <li>3GAddress - The IPv6 3G address if it is present</li> </ul>	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);
```

### Parameters

Table 90. WL.NativePage.show parameters

Parameter	Description
<b>className</b>	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).
<b>callback</b>	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.
<b>data</b>	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 : 'val
```



```
// 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 91. 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 (deprecated in IBM Worklight V6.0.0), 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 92. WL.BusyIndicator parameters

Parameter	Description
<b>containerId</b>	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 (deprecated in IBM Worklight V6.0.0), Windows Phone 8, and Windows 8.
<b>options</b>	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.

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 Windows Phone 7.5 and Windows Phone 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, the JSONStore allows 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 as EOC (PBKDF2 for key derivation from user password and AES 256). EOC continues to be supported as a cross-platform on-device data store mechanism for various mobile client OS platforms, as listed in “System requirements for using IBM Worklight” on page 8, but no major technical updates will be made to the EOC feature set.

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 about JSONStore, see “JSONStore overview” on page 393.

## 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)
```

### Description

Closes the cache. The cache must be reopened with the user’s credentials to be used again.

### Parameters

Parameter	Description
<code>onCompleteHandler</code>	Mandatory. Function. A callback method that is invoked when the encrypted cache is ready for use.  Signature: <code>successCallback(status)</code> , where <code>status</code> can be <code>WL.EncryptedCache.OK</code> .
<code>onErrorHandler</code>	Mandatory. Function. A callback method that is invoked when the action fails.  Signature: <code>failureCallback(status)</code> , where <code>status</code> can be <code>WL.EncryptedCache.ERROR_KEY_CREATION_IN_PROGRESS</code> .

### 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
<b>successCallback</b>	Mandatory. Function. A callback method that is invoked when the action succeeds.  Signature: successCallback ( <i>status</i> ), where <i>status</i> can be WL.EncryptedCache.OK.
<b>failureCallback</b>	Mandatory. Function. A callback method that is invoked when the action fails.  Signature: successCallback ( <i>status</i> ), where <i>status</i> can be WL.EncryptedCache.ERROR_KEY_CREATION_IN_PROGRESS.

#### Return Value

##### WL.EncryptedCache.OK

The encryption data was successfully removed from memory.

### 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 Worklight Server. Hence, the action of creating a cache requires that the app is connected to the Worklight Server. After a cache is created, it can then be opened without a connection.

## Parameters

Table 93. *WL.EncryptedCache.open* parameters

Parameter	Description
<b>credentials</b>	Mandatory. String. The credentials that are used to encrypt the stored data.
<b>create_if_none</b>	Mandatory. Boolean. Whether to create an encrypted cache if one does not exist.
<b>onCompleteHandler</b>	Mandatory. Function. A callback method that is invoked when the encrypted cache is ready for use.  The signature of this method is <code>onCompleteHandler(status)</code> . The possible value for <i>status</i> is <code>WL.EncryptedCache.OK</code>
<b>onErrorHandler</b>	Mandatory. Function. A callback method that is invoked when the action fails.  The signature of this method is <code>onErrorHandler(status)</code> . Possible values for <i>status</i> are: <code>WL.EncryptedCache.ERROR_KEY_CREATION_IN_PROGRESS</code> , <code>WL.EncryptedCache.ERROR_LOCAL_STORAGE_NOT_SUPPORTED</code> , <code>WL.EncryptedCache.ERROR_NO_EOC</code> , <code>WL.EncryptedCache.ERROR_COULD_NOT_GENERATE_KEY</code> , <code>WL.EncryptedCache.ERROR_CREDENTIALS_MISMATCH</code>

## Return Value

None

## **WL.EncryptedCache.read:**

Decrypts the value that is associated with the specified key.

### Syntax

```
WL.EncryptedCache.read(key, successCallback, failureCallback)
```

## Parameters

Table 94. *WL.EncryptedCache.read* parameters

Parameter	Description
<b>Key</b>	Mandatory. String. The key whose value needs to be decrypted.
<b>successCallback</b>	Mandatory. Function. A callback method that is invoked when the action succeeded.  Signature: <code>successCallback(value)</code> , where <i>value</i> is the result of the read action.
<b>failureCallback</b>	Mandatory. Function. A callback method that is invoked when the action fails.  Signature: <code>failureCallback(status)</code> , where <i>status</i> can be <code>WL.EncryptedCache.ERROR_EOC_CLOSED</code> .

## 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 95. WL.EncryptedCache.remove parameters

Parameter	Description
<b>key</b>	Mandatory. String. The key to remove.
<b>successCallback</b>	Mandatory. Function. A callback method that is invoked when the action succeeded.  Signature: <code>successCallback (status)</code> , where <i>status</i> can be <code>WL.EncryptedCache.OK</code> .
<b>failureCallback</b>	Mandatory. Function. A callback method that is invoked when the action fails.  Signature: <code>failureCallback(status)</code> , where <i>status</i> can be <code>WL.EncryptedCache.ERROR_EOC_CLOSED</code> .

## Return Value

None

## 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 96. WL.EncryptedCache.write parameters

Parameter	Description
<b>key</b>	Mandatory. String. The key to associate the data ( <b>value</b> ) with.



Table 96. *WL.EncryptedCache.write* parameters (continued)

Parameter	Description
<b>value</b>	Mandatory. String. The data to encrypt. When set to null, the key is removed.
<b>successCallback</b>	Mandatory. Function. A callback method that is invoked when the action succeeds.  Signature: <code>successCallback(status)</code> , where <code>status</code> can be <code>WL.EncryptedCache.OK</code> .
<b>failureCallback</b>	Mandatory. Function. A callback method that is invoked when the action fails.  Signature: <code>failureCallback(status)</code> , where <code>status</code> can be <code>WL.EncryptedCache.ERROR_EOC_CLOSED</code> .

### 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.

### 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 ISO 3166-1-alpha-2 country or region code.

### WL.JSONStore API

Descriptions and reference information for the JSONStore API functions.

For an overview and more descriptive information about JSONStore, see “JSONStore overview” on page 393.

### WL.JSONStore.add:

Stores data as documents inside a collection.

You can use the “WL.JSONStore.add” on page 531 function to store documents inside a collection. You can also pass an array of objects (for example [{name: 'carlos'}, {name: 'tim'}]) instead of a single object. Every object in the array is stored as a new document inside the collection. The documents require “WL.JSONStore.push” on page 543, unless {push: false} is specified

### API

@method add

@param data {Object or Array of Objects} Data to be added the collection.

@param [options] {Object}  
 additionalSearchFields (object, default: none)  
 push (boolean, default: true)  
 onSuccess (function, default: none, deprecated)  
 onFailure (function, default: none, deprecated)

@return {Promise} Resolved when the operation succeeds, returns number of docs added.  
 Rejected when there is a failure.

### Example

```
var data = {name: 'carlos', age: 99};  
var collectionName = 'people';  
var options = {}; //default
```

```
WL.JSONStore.get(collectionName)  
  
  .add(data, options)  
  
  .then(function (numberOfDocumentsAdded) {  
    //handle success  
  })  
  
  .fail(function (errorObject) {  
    //handle failure  
  });
```

### WL.JSONStore.changePassword:

Changes the password for the internal storage.

You must have a collection initialized before calling changePassword.

### API

@method changePassword

@static

@param oldPassword {string}  
 Must be alphanumeric ([a-z, A-Z, 0-9]) with at least 1 character.

@param newPassword {string} The new password  
 Must be alphanumeric ([a-z, A-Z, 0-9]) with at least 1 character.

@param username {string} (default: jsonstore)  
 Must be an alphanumeric string ([a-z, A-Z, 0-9]) with length greater than 0.

@param [options] {Object}  
 onSuccess (function, default: none, deprecated)  
 onFailure (function, default: none, deprecated)

@return {Promise} Resolved when the operation succeeds.  
 Rejected when there is a failure.

### Example

```
var oldPassword = '123';
var newPassword = '456';
var username = 'carlos';
```

```
WL.JSONStore.changePassword(oldPassword, newPassword, username)
```

```
.then(function () {
  oldPassword = null;
  newPassword = null;
  //handle success
})

.fail(function (errorObject) {
  //handle failure
});
```

### WL.JSONStore.closeAll:

Locks access to all the collections until `init` is called.

If “`WL.JSONStore.init`” on page 540 can be thought of as similar to login, then “`WL.JSONStore.closeAll`” is similar to logout. You can change the password using “`WL.JSONStore.changePassword`” on page 532. If the collections in the persistent store are password-protected, the password must be specified during the “`WL.JSONStore.init`” on page 540 call.

### API

@method closeAll

@static

@param [options] {Object}

onSuccess (function, default: none, deprecated)

onFailure (function, default: none, deprecated)

@return {Promise} Resolved when the operation succeeds.

Rejected when there is a failure.

### Example

```
WL.JSONStore.closeAll()
```

```
.then(function () {
  //handle success
})

.fail(function (errorObject) {
  //handle failure
});
```

### WL.JSONStore.count:

Returns the number of documents inside a collection.

There is no count function to get the number of documents that match a certain query (for example, to count how many documents match `{name: 'carlos'}`). If you must count documents that are based on a query, use `find` and get the length of the array that is returned.

## API

@method count

```
@param [options] {Object}
  onSuccess (function, default: none, deprecated)
  onFailure (function, default: none, deprecated)
```

```
@return {Promise} Resolved when the operation succeeds, returns an integer.
  Rejected when there is a failure.
```

### Example

```
var collectionName = 'people';

WL.JSONStore.get(collectionName).count()

.then(function (numberOfDocuments) {
  //handle success
})

.fail(function (errorObject) {
  //handle failure
});
```

### WL.JSONStore.destroy:

Completely wipes data for all users, destroys the internal storage, and clears security artifacts.

destroy removes the following:

- All documents
- All collections
- All stores (see “JSONStore multiple user support” on page 406)
- All JSONStore metadata and security artifacts (see “JSONStore security” on page 407)

## API

@method destroy

@static

```
@param [options] {Object}
  onSuccess (function, default: none, deprecated)
  onFailure (function, default: none, deprecated)
```

```
@return {Promise} Resolved when the operation succeeds.
  Rejected when there is a failure.
```

### Example

```
WL.JSONStore.destroy()

.then(function () {
  //handle success
})

.fail(function (errorObject) {
  //handle failure
});
```

### WL.JSONStore.documentify:

Takes an `_id` and a JSON object and creates a JSONStore-style document.

The use case for this function is getting the `_id` and the JSON object from the DOM, calling `documentify`, and then calling one of the JSONStore core API functions that takes a document as the first parameter (for example, “WL.JSONStore.replace” on page 546 or “WL.JSONStore.remove” on page 545).

#### API

@method documentify

@static

@param id {integer} `_id` for the Document

@param data {object} JSON data for the Document

@return {Object} JSONStore-style Document

#### Example

```
var d = WL.JSONStore.documentify(1, {fn: 'carlos', age: 99});  
//d = { _id: 1, json: {fn: 'carlos', age: 99} }
```

```
//Typical usage is calling a function like replace  
var collectionName = 'people';
```

```
WL.JSONStore.get(collectionName).replace(d)
```

```
.then(function () {  
  //handle success  
})  
  
.fail(function (errorObject) {  
  //handle failure  
});
```

### WL.JSONStore.enhance:

The `enhance` function allows developers to extend the core API to better fit their needs.

It provides a way to add functions to the prototype of a collection. You cannot use the same names that the core JSONStore core API uses (for example, “WL.JSONStore.find” on page 536, “WL.JSONStore.push” on page 543, “WL.JSONStore.add” on page 531).

#### API

@method enhance

@param name {string} - Function name

@param func {function} - Function to add to the prototype

@return {Integer} 0 return code for success or an error code for failure

#### Example

```
var collectionName = 'people';
```

```
//Definition
```

```
WL.JSONStore.get(collectionName).enhance('findByName', function (name, limit) {  
  var collectionAccessor = this;
```

```

        return collectionAccessor.find({name: name}, {exact: true, limit: limit});
    });

    //Usage
    WL.JSONStore.get(collectionName).findByName('carlos', 10)

    .then(function () {
        //res => Up to 10 documents that have a name of exactly 'carlos'
        //handle success
    })

    .fail(function (errorObject) {
        //handle failure
    });

```

### WL.JSONStore.find:

Locates a document inside a collection using a query.

To find all documents, use an empty object for the query (for example {}). There is a “WL.JSONStore.findAll” on page 537 method that returns all the documents inside the collection. It is possible to search by the document's unique identifier using “WL.JSONStore.findById” on page 538.

### Limit and Offset

Passing a limit to the options object will restrict the amount of results by the number specified. It is also possible to pass an offset, this will skip results by the number specified. In order to pass an offset a limit must also be passed. This is useful for implementing pagination. Document that get returned from find are stored in memory as strings at the JavaScript layer. In order to conserve memory when a significant number of results is expected it is a good idea to specify a limit.

### Fuzzy vs Exact

The default behavior is fuzzy searching, meaning that queries return partial results. For example, the query {name: 'carl'} finds 'carlos' and 'carl' (for example name LIKE '%carl%'). When {exact: true} is passed matches are exact but not case-sensitive, for example 'hello' matches 'Hello' (for example name.toLowerCase() = 'hello'). Integer matching is not type-sensitive, for example "1" matches both "1" and "1.0"). Numbers are stored as their decimal representation, for example "1" is stored as "1.0". Boolean values are indexed as 1 (true) and 0 (false).

- Exact Find: 0 or '0' or false
  - Returns: '0', 0, false
  - Android on IBM Worklight® <6.0 also returns 'false'
- Exact Find: 1 or '1' or true
  - Returns: 1 or '1', true
  - Android Worklight <6.0 also returns 'true'

The strings true and false are not converted to Boolean values.

- Exact Find: 'false'
  - Returns: 'false'
  - Android on IBM Worklight <6.0 also returns Boolean value 'false'
- Exact Find: 'true'
  - Returns: 'true'

- Android Worklight <6.0 also returns the Boolean value 'true'

### API

@method find

@param query {Query Object}

@param [options] {Object}  
  exact (boolean, default: true)  
  limit (integer, default: none)  
  offset (integer, default: none, depends on limit)  
  onSuccess (function, default: none, deprecated)  
  onFailure (function, default: none, deprecated)

@return {Promise} Resolved when the operation succeeds, returns an array of results.  
  Rejected when there is a failure.

### Example

```
var query = {name: 'carlos'};  
var collectionName = 'people';  
var options = {  
  exact: false,  
  limit: 10 //returns a maximum of 10 documents  
};  
  
WL.JSONStore.get(collectionName)  
  
  .find(query, options)  
  
  .then(function (arrayResults) {  
    //arrayResults = [{_id: 1, json: {name: 'carlos', age: 99}}]  
  })  
  
  .fail(function (errorObject) {  
    //handle failure  
  });
```

### WL.JSONStore.findAll:

Returns all of the documents stored in a collection.

findAll is the same as calling “WL.JSONStore.find” on page 536 with an empty object (for example {}) as the query. Limit and offset are explained in the “WL.JSONStore.find” on page 536 documentation page.

### API

@method findAll

@param [options] {Object}  
  limit (integer, default: none)  
  offset (integer, default: none, depends on limit)  
  onSuccess (function, default: none, deprecated)  
  onFailure (function, default: none, deprecated)

@return {Promise} Resolved when the operation succeeds, returns an array of results.  
  Rejected when there is a failure.

### Example

```
var collectionName = 'people';  
var options = {  
  limit: 10 //returns a maximum of 10 documents  
};
```



```

WL.JSONStore.get(collectionName)

.findAll(options)

.then(function (arrayResults) {
    //arrayResults = [{_id: 1, json: {name: 'carlos', age: 99}}]
})

.fail(function (errorObject) {
    //handle failure
});

```

### **WL.JSONStore.findById:**

Returns one or more documents that match the `_id` supplied to the function.

It is also possible to pass an array containing many `_id` values.

#### **API**

@method findById

@param id `{_id or Array of _id values}` `_id` values must be greater than 0

@param [options] `{Object}`  
 onSuccess (function, default: none, deprecated)  
 onFailure (function, default: none, deprecated)

@return `{Promise}` Resolved when the operation succeeds, returns an array of results.  
 Rejected when there is a failure.

#### **Example**

```

var id = 1;
var collectionName = 'people';

WL.JSONStore.get(collectionName)

.findById(id)

.then(function (arrayResults) {
    //arrayResults = [{_id: 1, json: {name: 'carlos', age: 99}}]
})

.fail(function (errorObject) {
    //handle failure
});

```

### **WL.JSONStore.get:**

Provides an accessor to the collection if the collection exists or undefined.

The `get` function depends on “`WL.JSONStore.init`” on page 540 being called first, with the collection name requested. The following methods clear the instances that are stored: “`WL.JSONStore.init`” on page 540 with `{clear: true}`, “`WL.JSONStore.destroy`” on page 534 and “`WL.JSONStore.closeAll`” on page 533. Accessors, also known as `JSONStoreInstance` Objects, must not be altered. To update values call “`WL.JSONStore.init`” on page 540 again.

#### **API**

@method get

@static

```
@param collection {String} Name of the collection
@return {Accessor} Allows access to the collection by name
```

### Example

```
var collectionName = 'people';
var people = WL.JSONStore.get(collectionName);
```

The variable `people` can now be used to perform operations such as: add, find and replace on the `people` collection.

### WL.JSONStore.getErrorMessage:

Returns the message that is associated with a JSONStore error code.

This function is only relevant to the JSONStore feature. It is not applicable to other error codes in the rest of IBM Worklight.

### API

```
@method getErrorMessage
```

```
@static
```

```
@param errorCode {Integer}
```

```
@return {String} The Error Message associated with the status code or 'Not Found'
                  if you pass an invalid value (non-integer) or a nonexistent status code.
```

### Example

```
WL.JSONStore.getErrorMessage(-50);
// "PERSISTENT_STORE_NOT_OPEN"
```

### WL.JSONStore.getPushRequired:

Returns all documents that are marked dirty.

`getPushRequired` returns every document that has local-only changes. The document that is returned has special metadata, such as the last operation that was performed on it.

### API

```
@method getPushRequired
```

```
@param [options] {Object}
```

```
  onSuccess (function, default: none, deprecated)
```

```
  onFailure (function, default: none, deprecated)
```

```
@return {Promise} Resolved when the operation succeeds, returns array of documents.
                  Rejected when there is a failure.
```

### Example

```
var collectionName = 'people';

WL.JSONStore.get(collectionName).getPushRequired()
  .then(function (arrayOfDocumentsThatWillBePushed) {
    //handle success
  })
```

```
.fail(function (errorObject) {  
    //handle failure  
});
```

### **WL.JSONStore.init:**

Starts one or more collections.

Starting or provisioning a JSONStore collection means that the persistent storage used to contain collections and documents is created, if it does not exist. If the store is encrypted and a correct password is passed, the necessary security procedures to make the data accessible are run. There is minimal overhead in initializing all the collections when an application starts.

### **Getting an accessor after init**

See the “WL.JSONStore.get” on page 538 function to retrieve an accessor to a specific collection. The accessor is commonly referred to as a JSONStoreInstance Object. It allows developers to call functions such as “WL.JSONStore.find” on page 536, “WL.JSONStore.add” on page 531, and many others, on collections that are properly initialized.

### **Calling init multiple times**

It is possible to call “WL.JSONStore.init” multiple times with different collections. New collections are initialized without affecting collections that are already initialized. Passing {clear: true} clears accessors without removing its contents from the store.

### **Search fields and additional search fields**

Refer to the “JSONStore search fields” on page 397 search field section for information about Search Fields and Additional Search Fields. These parameters are passed to init to determine what is indexed when data is added to a collection.

### **Security**

When a password is passed, the contents of the collection are encrypted. See “JSONStore security” on page 407 for information.

The first time that JSONStore initializes a collection with a password, meaning that the developer wants to encrypt data inside the store, it needs a random token. That random token can be obtained from the client, or from the server.

Passing `{localKeyGen: true}` tells JSONStore to initialize itself, and to encrypt its contents without contacting the Worklight Server for a secure token.

The token is generated locally, using native cryptographic functions that are part of the Android or iOS API. The default behavior (`{localKeyGen: false}`) is to contact the Worklight server for the random token. That token is generated on the server, and sent to the client with an AJAX call, for example: "response [/apps/services/random] success: e9097576c8663f4d9946c9389570ff34bf81975c"  
The tradeoff is between being able to initialize a JSONStore collection offline and trusting the client to generate that random token (less secure), or doing the initialization call with access to the Worklight Server (requires connectivity) and trusting the server (more secure).

For encrypted collection sets, the password is only required the first time that `init` is called. See “`WL.JSONStore.closeAll`” on page 533 to disable access to the accessors.

### Multiple user support

When a user name is passed it is used to determine the file name of the store.

See “`JSONStore multiple user support`” on page 406 for information.

### Worklight adapter integration

You can pass adapter meta data (for example adapter name, add procedure name, load procedure) to the collection object passed to `init`. The goal is to get and send data from and to IBM Worklight adapters. The `accept` function that is part of the adapter object gets called with the response from the adapter, if it returns true, the document is marked as clean (not dirty, no local-only modifications). Refer to the adapter integration section for information.

### Remove content from a collection

See “`WL.JSONStore.removeCollection`” on page 546 to remove the contents of a specific collection from disk. See “`WL.JSONStore.destroy`” on page 534 for a full data wipe of all `JSONStore` related data.

### API

@method `init`

@static

@param `collections` {Object}

`collectionName` (string [a-z, A-Z, 0-9])

`searchFields` (object, default: {})

`additionalSearchFields` (object, default: {})

`adapter` (object, default: {})

`adapter.name` (string) - Name of the Adapter

`adapter.add` (string) - Name of the add procedure

`adapter.remove` (string) - Name of remove procedure

`adapter.load` (object)

`adapter.load.procedure` (string) - Name of the load procedure

`adapter.load.params` (array) - Parameters sent to the load procedure

`adapter.load.key` (string) - Key in the response containing objects to add

`adapter.accept` (function, returns: boolean) - Called after push with the response from the adapter

`adapter.timeout` (integer) - Timeout for the adapter call

@param `[options]` {Object}

`username` (string [a-z, A-Z, 0-9], default: 'jsonstore')

`password` (string, default: none),

`clear` (boolean, default: false)

`localKeyGen` (boolean, default: false)

@return {Promise} Resolved when all collections have been initialized.

  Rejected when there is a failure (no accessors created).

### Example

```
var collectionName = 'people';
```

```
//Object that defines all the collections
```

```
var collections = {};
```

```
//Object that defines the 'people' collection
```

```

collections[collectionName] = {};

//Object that defines the Search Fields for the 'people' collection
collections[collectionName].searchFields = {name: 'string', age: 'integer'};

//Optional Worklight Adapter integration
collections[collectionName].adapter = {
  name: 'people',
  add: 'addPerson',
  remove: 'removePerson',
  replace: 'replacePerson',
  load: {
    procedure: 'getPeople',
    params: [],
    key: 'peopleList'
  },
  accept: function (adapterResponse) {
    return (adapterResponse.status === 200);
  },
  timeout: 3000
};

//Optional options object
var options = {};

//Optional username
options.username = 'carlos';

//Optional password
options.password = '123';

//Optional local key generation flag
options.localKeyGen = false;

//Optional clear flag
options.clear = false;

WL.JSONStore.init(collections, options)

.then(function () {
  //handle success
})

.fail(function (errorObject) {
  //handle failure
});

```

### **WL.JSONStore.isPushRequired:**

Returns a boolean that is true if the document has local-only changes and is pushed when push is called, and is false otherwise.

#### **API**

@method isPushRequired

@param doc {Document or \_id}

@param [options] {Object}

onSuccess (function, default: none, deprecated)

onFailure (function, default: none, deprecated)

@return {Promise} Resolved when the operation succeeds, return a boolean.  
Rejected when there is a failure.

### Example

```
var collectionName = 'people';
var doc = {_id: 1, json: {name: 'carlitos', age: 99}};

WL.JSONStore.get(collectionName).isPushRequired(doc)

.then(function (isDocumentDirty) {
  //isDocumentDirty - true or false
  //handle success
})

.fail(function (errorObject) {
  //handle failure
});
```

### WL.JSONStore.load:

Gets data that is defined in the load portion of the adapter.

This process is analogous to invoking an adapter with `WL.Client.invokeProcedure` and calling the `add` function in `JSONStore` with the `{push: false}` flag and the data that is returned by the adapter. This function always stores whatever it gets back from the adapter. If the data exists, it is duplicated in the collection.

### API

@method load

```
@param [options] {Object}
  onSuccess (function, default: none, deprecated)
  onFailure (function, default: none, deprecated)
```

```
@return {Promise} Resolved when the operation succeeds, returns number of documents loaded.
  Rejected when there is a failure.
```

### Example

```
var collectionName = 'people';

WL.JSONStore.get(collectionName).load()

.then(function (numberOfDocumentsLoaded) {
  //handle success
})

.fail(function (errorObject) {
  //handle failure
});
```

### WL.JSONStore.push:

Pushes documents inside the collection that have local-only changes to an IBM Worklight adapter linked during the `init` function.

For every document marked requiring push, call the corresponding adapter procedure that is linked to the collection. The documents are processed on the client in order of its last modification date. To check the number of documents that are pushed, see “`WL.JSONStore.getPushRequired`” on page 539. `push` can take a document, an array of documents or an `_id`, and if what is passed has local-only changes, it is sent to the adapter.

## Errors

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 go to the promise's fail function. This class of error implies that 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 go to the promise's then or done function. An overview of failure conditions includes:

- Wrong parameters are passed.
- Something goes wrong while native code is running to get the documents or verify which documents have local changes.
- When the Worklight adapter is called, with the appropriate option (add, "WL.JSONStore.replace" on page 546, or "WL.JSONStore.remove" on page 545) it can fail if the adapter or procedure name does not exist (if it is misspelled, for example) or if the Worklight Server or back-end server cannot be reached.
- If the server is successfully contacted, checking the status code from the adapter procedure's response to determine whether to mark that document as not dirty. This step uses native code and can fail.

## Callbacks for onSuccess and onFailure

The deprecated callbacks for success and failure are called once per document. If you try to push 10 documents, your success callback may get called 9 times, and the failure callback once. The promise-based code waits until every document has tried to reach the adapter before returning.

## API

@method push

@param [options] {Options or Array of Documents or Document or \_id}  
onSuccess (function, default: none, deprecated)  
onFailure (function, default: none, deprecated)

@return {Promise} Resolved when the operation succeeds, returns an array.  
Array returned is either empty (everything worked) or full of error responses.  
Rejected when there is a failure.

## Example

```
var collectionName = 'people';

WL.JSONStore.get(collectionName).push()

.then(function (res) {
  //handle success
  //res is an empty array if all documents reached the server
  //res is an array of error responses if some documents failed to reach the server
})

.fail(function (errorObject) {
  //handle failure
});
```

## WL.JSONStore.pushRequiredCount:

Returns the number of documents with local-only changes (that is, dirty documents).



This number includes documents that are marked for removal with `remove`. It is the same as calling `getPushRequired` and getting the length of the array that is returned.

### API

@method `pushRequiredCount`

@param [options] {Object}  
  onSuccess (function, default: none, deprecated)  
  onFailure (function, default: none, deprecated)

@return {Promise} Resolved when the operation succeeds, returns an integer.  
  Rejected when there is a failure.

### Example

```
var collectionName = 'people';

WL.JSONStore.get(collectionName).pushRequiredCount()

.then(function (numberOfDirtyDocuments) {
  //handle success
})

.fail(function (errorObject) {
  //handle failure
});
```

### WL.JSONStore.remove:

Marks a document deleted inside a collection.

Removed documents are not returned by “`WL.JSONStore.find`” on page 536, “`WL.JSONStore.findAll`” on page 537 “`WL.JSONStore.findById`” on page 538, and they do not affect “`WL.JSONStore.count`” on page 533. Documents are not really erased from the collection until “`WL.JSONStore.push`” on page 543 is successfully called. This is because `JSONStore` keeps track of local-only changes. Passing `{push: false}` forces documents to be physically deleted without having to call “`WL.JSONStore.push`” on page 543, changes are not tracked.

### API

@method `remove`

@param doc {Document or Array of Documents or Query or \_id}  
@param [options] {Object}  
  push (boolean, default: true)  
  onSuccess (function, default: none, deprecated)  
  onFailure (function, default: none, deprecated)

@return {Promise} Resolved when the operation succeeds, returns number of docs removed.  
  Rejected when there is a failure.

### Example

```
var query = { _id: 1 };
var collectionName = 'people';
var options = {
  push: true
};

WL.JSONStore.get(collectionName)

.remove(query, options)
```

```

.then(function (numberOfDocumentsRemoved) {
    //handle success
})

.fail(function (errorObject) {
    //handle failure
});

```

### **WL.JSONStore.removeCollection:**

Deletes all the documents stored inside a collection.

The “WL.JSONStore.removeCollection” function is similar to dropping a table, in database terms. This only removes the contents of the collection locally. To use a collection with the same name you must call “WL.JSONStore.init” on page 540. This action does not call push before the operation. In order to remove specific documents, see the “WL.JSONStore.remove” on page 545 function. To destroy everything related to JSONStore, use “WL.JSONStore.destroy” on page 534.

### **API**

@method removeCollection

@param [options] {Object}  
 onSuccess (function, default: none, deprecated)  
 onFailure (function, default: none, deprecated)

@return {Promise} Resolved when the operation succeeds.  
 Rejected when there is a failure.

### **Example**

```

var collectionName = 'people';

WL.JSONStore.get(collectionName)

.removeCollection()

.then(function () {
    //handle success
})

.fail(function (errorObject) {
    //handle failure
});

```

### **WL.JSONStore.replace:**

Overwrites a document with a given document.

The “WL.JSONStore.replace” function is used to modify documents inside a collection. The field used to perform the replacement is the document unique identifier (`_id`). By default, documents are marked for replacement when push is called, unless `{push: false}` is specified.

### **API**

@param doc {Document or Array of Documents}

@param [options] {Object}  
 push (boolean, default: true)  
 onSuccess (function, default: none, deprecated)  
 onFailure (function, default: none, deprecated)

@return {Promise} Resolved when the operation succeeds, returns number of docs replaced.  
Rejected when there is a failure.

The following example assumes that the document `{_id: 1, json: {name: 'carlos', age: 99}}` is in the collection, and demonstrates how to replace the name carlos in the document with the name carlitos.

### Example

```
var doc = {_id: 1, json: {name: 'carlitos', age: 99}};
var collectionName = 'people';
var options = {
  push: true
};

WL.JSONStore.get(collectionName)

.replace(doc, options)

.then(function (numberOfDocumentsReplaced) {
  //handle success
})

.fail(function (errorObject) {
  //handle failure
});
```

### WL.JSONStore.toString:

Prints the contents of the collection by using `WL.Logger.debug` asynchronously.

If no parameters are passed, `toString` prints the first 100 documents. If the first parameter is a valid integer, it prints up to that number of documents (`limit`). It is also possible to pass a second valid integer as the second parameter to skip some number of documents (`offset`). If a 0 is passed as the first parameter it prints only the collection metadata (for example, `name`, `searchFields`, `additionalSearchFields`, `adapter`).

### 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

//Equivalent to:
collection.findAll().done(function(data){WL.Logger.debug(JSON.stringify(data))});
```

### WL.JSONStore deprecated:

These functions are deprecated as of IBM Worklight 6.0

*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” on page 540

## Parameters

Parameter	Description
<b>Boolean</b>	Returns 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.erase:
```

Removes Documents from a collection.

## Syntax

```
erase(doc, [options]) onSuccess deprecated
```

## Description

**Note:** Deprecated, use “WL.JSONStore.remove” on page 545

Removes the Document from internal storage immediately, rather than just marking document for removal to be removed later (when you call **push** or **pushSelected** with that specific document) as “WL.JSONStore.remove” on page 545 does.

## Parameters

Parameter	Description
<b>doc</b>	Document, Array of Documents, Query, or Integer  The Integer is an <code>_id</code> .
<b>[options]</b>	Returns Options (not required)
<b>onSuccess</b>	Returns the number of documents removed.
<b>onFailure</b>	Returns 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.initCollection:*

Creates a new object to interact with a single collection.

### Syntax

`initCollection (name, searchfields, [options])` JSONStoreInstance deprecated static

### Description

**initCollection** creates a new object to interact with a single collection and must be called sequentially, meaning that 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 user name and a password, both are optional. If no user name is passed, it uses the default one. You cannot use the following default *usernames*: *jsonstore*, *JSONStoreKey*, *dpk*.

Deprecated, use “WL.JSONStore.init” on page 540

### Parameters

Parameter	Description
<b>Name</b>	A string defining the collection name.
<b>searchField</b>	The search fields.
<b>[options]</b>	Returns Options (not required)  You can also link a collection to an Adapter. You can also pass <code>load:true</code> and it will check if the collection is empty and load data using the adapter you defined to get data. You may pass a user name (alphanumeric strings only: [a-zA-Z, 0-9]) and a password. <code>localKeyGen:true</code> rather than contacting the IBM Worklight server to generate the data protection key, it will be done on the local device.
<b>onSuccess</b>	A JSONStoreInstance The collection will not be usable until the promise is resolved or the successful callback is called.
<b>onFailure</b>	Returns 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 you 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 (error) {
    WL.Logger.debug(error.toString());
});

//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.pushSelected:*

Push the selected Documents

### Syntax

```
pushSelected(doc, [options])
```

### Description

Pushes only the selected Documents. See “WL.JSONStore.push” on page 543. The Document passed will not be sent to the Adapter (pushed) if it is not marked unpushed.

### Deprecated

Use `push(doc)`

### Parameters

Parameter	Description
<b>doc</b>	Document or Array of Documents
<b>[options]</b>	Options (not required)
<b>Promise</b>	Returns a dynamically-generated Promise.
<b>onSuccess</b>	See “WL.JSONStore.push” on page 543
<b>onFailure</b>	Returns an error code.

### 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.

### Deprecated

Use “WL.JSONStore.replace” on page 546

### Parameters

Parameter	Description
<b>doc</b>	Document or Array of Documents
<b>[options]</b>	Options (not required)



Parameter	Description
<b>onSuccess</b>	Returns an Integer with the number of Documents replaced.
<b>onFailure</b>	Returns 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);
```

*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 531 to store Documents as **unpushed**.

### Deprecated

Use “WL.JSONStore.add” on page 531

### Parameters

Parameter	Description
<b>data</b>	Returns an Object or Array of Objects. Data to be added to the collection.
<b>[options]</b>	Returns Options (not required)  For additional options: additionalSearchFields : {}
<b>Promise</b>	Returns a dynamically-generated Promise.
<b>onSuccess</b>	Returns an integer with the amount of data stored
<b>onFailure</b>	Returns an error code.

### 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 Multiple Objects without the Array
var data1 = dataArray[0];
var data2 = dataArray[1];
collection.store(data1, {onSuccess: function(){
    collection.store(data2, {onSuccess: win});
}});
```

*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 data that is stored locally on the device. This function is deprecated.

### Deprecated

Use “WL.JSONStore.init” on page 540

### Parameters

Parameter	Description
<b>pwd</b>	String containing the password
<b>onSuccess</b>	Returns a boolean: true if the password is a valid string, false otherwise. Example: var

### Example

```
var pwd = prompt('What is your password?');
WL.JSONStore.usePassword(pwd);
```

## Location services API

IBM Worklight provides a number of functions for location services. Location services enable you to use Geo and WiFi positions to perform various actions.

Location services are supported for hybrid applications on Android and iOS.

For Android, the following permissions are required.

For Geo acquisition:

- ACCESS\_COARSE\_LOCATION
- ACCESS\_FINE\_LOCATION (when **enableHighAccuracy=true**)

For WiFi acquisition:

- ACCESS\_WIFI\_STATE
- CHANGE\_WIFI\_STATE

For iOS, you must update info.plist with the following information.

Geo:  
UIRequiredDeviceCapabilities:  
  location-services  
  gps (when **enableHighAccuracy=true**)  
Wifi:  
UIRequiredDeviceCapabilities: wifi

When location services are running in the background on iOS:  
UIBackgroundModes key: location (when **enableHighAccuracy=true**)

### **WL.App.setKeepAliveInBackground:**

Depending on the returned value, either unbinds from the foreground service, or binds to a foreground service. This API function is available only for Android.

#### **Syntax**

WL.App.setKeepAliveInBackground(enabled, options)

#### **Description**

If the **enabled** parameter has a value of true, the device binds to a foreground service and presents a notification. The notification is instantiated according to the **options** settings. When the notification is pressed, the activity that made the call is started. You can use SingleTop or SingleTask launch modes to reuse the same instance of the activity,

If the **enabled** parameter has a value of false, the device unbinds from the foreground service, if it was previously bound.

#### **Parameters**

Parameter	Description
<b>enabled</b>	A Boolean value that determines whether to enable or disable the binding.

Parameter	Description
<b>options</b>	<p>Options that are used to configure the notification. The <b>options</b> parameter has the following properties:</p> <ul style="list-style-type: none"> <li>• <b>tickerText</b>. Text to be displayed in the status bar. By default this is the application display name.</li> <li>• <b>contentTitle</b>. A title in the expanded entry on the notifications screen. By default this is the application display name.</li> <li>• <b>contentText</b>. Text in the expanded entry on the notifications screen. By default the text is "App keeps running in background".</li> <li>• <b>icon</b>. The name of the icon to be shown. By default the name is "push", which matches the icon that is displayed when push notifications are received.</li> <li>• <b>notificationID</b>. An ID number to be used for this notification. By default the ID number is -111.</li> <li>• <b>className</b>. A string that contains the compiled name of the native Activity class (including package). For example, com.myApp.MyActivity. By default, the current activity is used.</li> </ul>

#### **WL.Client.purgeEventTransmissionBuffer:**

Purges the internal event transmission buffer.

#### **Syntax**

```
WL.Client.purgeEventTransmissionBuffer
```

#### **Description**

The internal event transmission buffer is purged, and all events awaiting transmission are permanently lost.

#### **Parameters**

None.

#### **WL.Client.setEventTransmissionPolicy:**

Configures the transmission of events from the client to the server, according to the provided transmission policy.

#### **Syntax**

```
WL.Client.setEventTransmissionPolicy(policy)
```

## Parameters

Parameter	Description
<b>policy</b>	<p>The <b>policy</b> object has the following properties. Both of these properties are optional.</p> <ul style="list-style-type: none"><li>• <b>eventStorageEnabled</b>. A Boolean value that determines where events are stored. If the value is true, events may be stored in HTML5 session storage. If the value is false, events that are waiting for transmission are stored in memory. The default value is false.</li></ul> <p>The value true should be used only if the developer of the policy is not concerned about privacy issues involved in storing the events, which include the user's device context.</p> <ul style="list-style-type: none"><li>• <b>interval</b>. The transmission interval, in milliseconds. The default value is 60000 (one minute). Events are accumulated before being transmitted, and if <b>eventStorageEnabled</b> is true, they are copied into system storage.</li></ul>

### WL.Client.transmitEvent:

Transmits a provided event object to the server.

#### Syntax

```
WL.Client.transmitEvent(event, immediate)
```

#### Description

An event object is added to the transmission buffer. The event object is either transmitted immediately, if the **immediate** parameter is set to true, otherwise it is transmitted according to the transmission policy. For more information, see “WL.Client.setEventTransmissionPolicy” on page 555. One of the properties for the received event object might be **deviceContext**, which comprises geolocation and WiFi data. If no **deviceContext** property is transmitted as part of the event, the current device context, as returned by WL.Device.getContext, is added automatically to the event during the transmission process.

#### Parameters

Parameter	Description
<b>event</b>	Mandatory. The event object that is being transmitted. The event object is either a literal object, or a reference to an object.
<b>immediate</b>	Optional. A Boolean flag that indicates whether the transmission should be immediate (true), or should be based on the transmission policy's interval (false). If <b>immediate</b> is true, previously buffered events are transmitted, as well as the current event. The default value is false.

### WL.Device.Geo.acquirePosition:

Acquires a geographical position.

#### Syntax

```
WL.Device.Geo.acquirePosition(onSuccess, onFailure, options)
```

#### Description

The device attempts to acquire a geographical position. This attempt could be based on geolocation data acquired by the device, or it could involve the use of WiFi. If the attempt is successful, the following actions take place:

- The device context might be updated. This action is dependent on the freshness of the data in the context, and the new position data being at least as accurate as the existing position data.
- The **onSuccess** function is invoked.
- If the device context was updated, triggers might be activated.

**Note:** Because `acquirePosition` might activate triggers, you should not call `acquirePosition` from a trigger callback. Potentially, this could cause an endless loop of trigger evaluations leading to callbacks leading to `acquirePosition` calls.

For details of the permissions required for Android and iOS, see “Location services API” on page 553.

#### Parameters

Parameter	Description
<b>onSuccess</b>	A callback function that is invoked when a position is acquired successfully. The position is passed as a parameter to the callback function and is described in the W3C Geolocation specification. Note that the <b>timestamp</b> property is represented as the number of milliseconds that has elapsed since 1 January 1970.
<b>onFailure</b>	A callback function that is invoked if the acquisition fails.  The callback function receives a <code>PositionError</code> parameter, as described in the W3C Geolocation specification.

Parameter	Description
<b>options</b>	<p>Options that are used during the acquisition configuration process. <b>Options</b> has the following properties:</p> <ul style="list-style-type: none"> <li>• <b>timeout</b>. An optional property. The number of milliseconds spent waiting for each reading. If this amount of time elapses before a reading is obtained, the <b>onFailure</b> function is called. By default the timeout interval is infinite.</li> <li>• <b>enableHighAccuracy</b>. A mandatory property that has a Boolean value indicating whether it is possible to obtain high-accuracy measurements, for example by using GPS.</li> <li>• <b>highAccuracyOptions</b>. An optional property. When <b>enableHighAccuracy</b> is true, this property may specify additional parameters: <ul style="list-style-type: none"> <li>– <b>desiredAccuracy</b>. An optional parameter. An integer value that gives the required accuracy in meters. A higher <b>desiredAccuracy</b> value could result in power savings. For example, the platform might choose to use a WiFi or Network-based approach to determining the position, instead of GPS.</li> <li>– <b>iOSBestAccuracy</b>. An optional parameter. The value can be <code>WL.Device.Geo.IOS_BEST_ACCURACY</code> or <code>WL.Device.Geo.IOS_BEST_ACCURACY_FOR_NAVIGATION</code>. If <b>iOSBestAccuracy</b> is enabled, it overrides the <b>desiredAccuracy</b> setting. For more information about iOS Accuracy constants, search for “location constants” on the Apple Mac Developer Library at <a href="https://developer.apple.com/library/mac/">https://developer.apple.com/library/mac/</a> navigation.</li> </ul> </li> <li>• <b>maximumAge</b>. A time, measured in milliseconds. A cached position is returned if the age of that position is less than the specified <b>maximumAge</b> value. A higher <b>maximumAge</b> value can result in previously acquired positional data being used. This saves power, but the trade-off is that the data is not as fresh.</li> </ul> <p>Note that the <b>options</b> parameter is different from the <b>policy</b> parameter that is passed to the <code>WL.Device.startAcquisition</code> function. If <code>aPolicy</code> is the <b>policy</b> parameter for <code>WL.Device.startAcquisition</code>, then <code>aPolicy.Geo</code> would be the options object that could be passed to this API.</p>



## WL.Device.Geo.Profiles:

Returns a geo policy.

### Syntax

WL.Device.Geo.Profiles.*function()*

### Description

This API returns a geo policy, which is dependent on the function specified. You can modify the returned object before using it in the startAcquisition or acquirePosition APIs.

### Functions

Function	Description
PowerSaving	<p>Used to save power. If PowerSaving is specified, accurate location information is not provided.</p> <p>The PowerSaving function has the following properties, which have preset values as follows:</p> <ul style="list-style-type: none"><li>• <b>enableHighAccuracy=false</b></li><li>• <b>minChangeTime=300000</b> (5 minutes)</li><li>• <b>minChangeDistance=1000</b> (1 kilometer)</li><li>• <b>maximumAge=300000</b> (5 minutes)</li></ul>
RoughTracking	<p>Used to track devices, but at a rough granularity.</p> <p>The RoughTracking function has the following properties, which have preset values as follows:</p> <ul style="list-style-type: none"><li>• <b>enableHighAccuracy=true</b></li><li>• <b>highAccuracyOptions=true</b><ul style="list-style-type: none"><li>– <b>desiredAccuracy=200</b> (200 meters)</li></ul></li><li>• <b>minChangeTime=30000</b> (30 seconds)</li><li>• <b>minChangeDistance=50</b> (50 meters)</li><li>• <b>maximumAge=60000</b> (60 seconds)</li></ul>
LiveTracking	<p>Used to track devices, and get the best position information available.</p> <p>The LiveTracking function has the following properties, which have preset values as follows:</p> <ul style="list-style-type: none"><li>• <b>enableHighAccuracy=true</b></li><li>• <b>maximumAge=100</b> (100 milliseconds)</li></ul>

### Related reference:

Acquires a geographical position.

Starts ongoing acquisition for sensors that are provided in the policy.

## WL.Device.getContext:

Returns the current device context.

### Syntax

```
WL.Device.getContext
```

### Description

The device context object is returned. The object comprises object entries for each sensor, and a timestamp of the last change. This information is returned in the format:

```
Geo:{...},Wifi:{...},lastModified,timezoneOffset
```

Object	Description
<b>Geo</b>	<p>The Geo object has the following parameters:</p> <ul style="list-style-type: none"><li>• <b>timestamp.</b> The timestamp of the last update of the Geo object in the user's time zone. This value is the number of milliseconds that have elapsed since 00:00:00 on 1 January 1970.</li><li>• <b>coords.</b> The coordinates object returned by the navigator service. For more information, see the section on coordinates on the W3C website, at <a href="http://www.w3.org/TR/geolocation-API/#coordinates">http://www.w3.org/TR/geolocation-API/#coordinates</a>.</li></ul> <p>When an acquisition has been started (according to <code>WL.Device.startAcquisition</code>) but no information has been acquired, the object will be empty.</p>
<b>Wifi</b>	<p>The Wifi object has the following parameters:</p> <ul style="list-style-type: none"><li>• <b>timestamp.</b> The timestamp of the last update of the Wifi object in the user's time zone. This value is the number of milliseconds that have elapsed since 00:00:00 on 1 January 1970.</li><li>• <b>accessPoints.</b> An array of access points; these are a subset of all visible access points filtered according to policy. Each access point has the format: {SSID:} or {SSID:, MAC:}</li><li>• <b>connectedAccessPoint.</b> Information about the connected access point if it passes the policy filter. Each access point has the format: {SSID:} or {SSID:, MAC:}, according to the policy.</li></ul> <p>When acquisition has been started (according to <code>WL.Device.startAcquisition</code>) but no information has been acquired, the object will be empty.</p>

Object	Description
<b>lastModified</b>	A timestamp that matches the maximum timestamp of the Geo or Wifi object. If neither object has a timestamp, the <b>lastModified</b> parameter is not present.
<b>timezoneOffset</b>	An integer value that indicates how many minutes must be added to the user's local time in order to arrive at Coordinated Universal Time (UTC). This value is used in analytics; for example, it might be needed to synch users across time zones.

If `WL.Client.startAcquisition` has not been called since startup, or since the latest call to `WL.Client.stopAcquisition`, a null code is returned.

### Parameters

None.

### Example

```
{
  Geo: {
    timestamp:136475322150,
    coords:{
      speed:10.2,
      altitudeAccuracy:null,
      accuracy:166000,
      altitude:null,
      longitude:34.777819,
      latitude:32.066158}
    },
  Wifi: {
    timestamp:136475322168,
    accessPoints:[{SSID:"IBM"}, {SSID:"IBMVISITOR",MAC:"00:01:02:04:06:08"}],
    connectedAccessPoint:{SSID:"IBM"}
  },
  lastModified:136475322150,
  timezoneOffset:420
}
```

### **WL.Device.startAcquisition:**

Starts ongoing acquisition for sensors that are provided in the policy.

### Syntax

```
WL.Device.startAcquisition (policy, triggers, onFailure)
```

### Description

Ongoing acquisition is started for the GPS and WiFi sensors that are provided in the policy. When new sensor information is acquired, the device context is updated, and the specified triggers are evaluated for activation.

For details of the permissions that are required for Android and iOS, see “Location services API” on page 553.

## Parameters

Parameter	Description
<p><b>policy</b></p>	<p>The <b>policy</b> parameter is used to configure the acquisition. The parameter specifies relevant entries for each sensor. These entries are objects, Geo and Wifi, which have associated properties.</p> <p>The Geo object has the following properties:</p> <ul style="list-style-type: none"> <li>• <b>timeout</b>. An optional property. The timeout interval for position acquisition, specified in milliseconds. The default value is infinite. If no position is acquired since the last position was acquired, or since <code>WL.Device.startAcquisition</code> was called, then the <b>onFailure</b> function is called.</li> <li>• <b>enableHighAccuracy</b>. A mandatory property that has a Boolean value indicating whether it is possible to obtain high-accuracy measurements, for example by using GPS.</li> <li>• <b>highAccuracyOptions</b>. An optional property. When <b>enableHighAccuracy</b> is true, this property might specify additional parameters: <ul style="list-style-type: none"> <li>– <b>desiredAccuracy</b>. An optional parameter. An integer value that gives the required accuracy in meters. On some platforms, a higher <b>desiredAccuracy</b> value could result in power savings. For example, the platform might choose to use a WiFi or Network-based approach to determining the position, instead of GPS.</li> <li>– <b>iOSBestAccuracy</b>. An optional parameter. The value can be <code>WL.Device.Geo.IOS_BEST_ACCURACY</code> or <code>WL.Device.Geo.IOS_BEST_ACCURACY_FOR_NAVIGATION</code>. If <b>iOSBestAccuracy</b> is enabled, it overrides the <b>desiredAccuracy</b> setting.</li> </ul> <p>For more information about iOS Accuracy constants, search for “location constants” on the Apple Mac Developer Library at <a href="https://developer.apple.com/library/mac">https://developer.apple.com/library/mac</a>.</p> </li> <li>• <b>minChangeDistance</b>. An optional property that has an integer value giving the minimum distance in meters that the position must change by since the last update in order to receive a new updated position. On some platforms, higher values can improve battery life, although the effect is generally less than that of <b>minChangeTime</b>. The default value is 0.</li> <li>• <b>minChangeTime</b>. An optional property that has an integer value giving the minimum time in milliseconds between updates. On some platforms, higher values can improve battery life. For example, the system might choose to power off hardware between readings. The default value is 0.</li> </ul> <p><b>Note:</b> This property is available only on Android, and is ignored on other platforms.</p> <ul style="list-style-type: none"> <li>• <b>maximumAge</b>. A time, measured in milliseconds. A cached position is returned if the age of that position is less than the specified <b>maximumAge</b> value.</li> </ul> <p><b>Note:</b> The Geo object follows the W3C specification for geolocation. For more information, see the W3C Geolocation API Specification at <a href="http://www.w3.org/TR/geolocation-API/">http://www.w3.org/TR/geolocation-API/</a>.</p> <p>The Wifi object has the following parameters:</p> <ul style="list-style-type: none"> <li>• <b>interval</b>. A polling interval, specified in milliseconds. WiFi polling is performed each interval. The default value is 10000 (10 seconds).</li> </ul>

Parameter	Description
triggers	<p>The <b>triggers</b> parameter holds trigger definitions for all sensors, in the format {Geo:{...}, Wifi:{...}}. The Geo and Wifi objects hold the key-value entries that define triggers, in the format triggerName:{triggerDefinition}. The triggerDefinition object consists of the following parameters:</p> <ul style="list-style-type: none"> <li>• <b>type</b>. The trigger type. <ul style="list-style-type: none"> <li>The Geo object has the following trigger types: <ul style="list-style-type: none"> <li>- PositionChange</li> <li>- Enter</li> <li>- Exit</li> <li>- DwellInside</li> <li>- DwellOutside</li> </ul> </li> <li>The PositionChange trigger has an optional <b>minChangeDistance</b> parameter that has an integer value giving a distance in meters. After the first acquisition, this trigger will be activated only when the reported position has changed by at least this amount. The value should be greater than that of the <b>minChangeDistance</b> parameter defined on the Geo policy, otherwise it will have no effect.</li> <li>The Enter, Exit, DwellInside, and DwellOutside triggers (known collectively as <i>geofence triggers</i>), have the following parameters. <ul style="list-style-type: none"> <li>- <b>area definition</b>. This mandatory parameter is defined as an object, with a value of circle:{longitude:,latitude:,radius:}, or polygon:[{longitude:,latitude:},{longitude:,latitude:},{longitude:,latitude:}].</li> <li>- <b>bufferZoneWidth</b>. Use this optional parameter to change the size of the buffer zone, which is the area in which triggers are activated. The value of <b>bufferZoneWidth</b> indicates in meters how much to change the area. It can have either a positive or negative value. If it has a positive value, the area becomes bigger. If it has a negative value, the area becomes smaller. All geofence triggers operate on this new area. The default value is 0, which leaves the area unchanged.</li> <li>- <b>confidenceLevel</b>. An optional String parameter that has three possible values: <ul style="list-style-type: none"> <li>• low. The trigger is eligible for activation when the position is inside the area (Enter, DwellInside) or outside the area (Exit, DwellOutside), respectively. Low is the default value.</li> <li>• medium. The trigger is eligible for activation when there is approximately a 70% confidence interval that the device is inside the area, or outside the area, respectively. This takes into account the acquired accuracy of the position.</li> <li>• high. The trigger is eligible for activation when there is approximately a 95% confidence interval that the device is inside the area, or outside the area, respectively. This takes into account the acquired accuracy of the position.</li> </ul> </li> </ul> </li> </ul> </li> </ul> <p>For the DwellInside, and DwellOutside triggers, there is an additional parameter, <b>dwellingTime</b>, which is defined in milliseconds. This determines how long the device must be inside or outside the area before the trigger is activated.</p> <p>The Wifi object has the following trigger types:</p>

Experimental IBM Worklight offline documentation. This document is preliminary. In case of issues, refer to the main documentation.

Parameter	Description
<b>onFailure</b>	<p>An object that defines the error callback functions for each sensor. The structure of the object is: {Geo: errorCallbackFunc, Wifi: errorCallbackFunc}.</p> <p>For Geo sensors, the callback function receives a PositionError parameter, as described in the W3C Geolocation specification at <a href="http://www.w3.org/TR/geolocation-API/#position_error_interface">http://www.w3.org/TR/geolocation-API/#position_error_interface</a>.</p> <p>For WiFi sensors, the callback function receives a numeric code that corresponds to the following values:</p> <ul style="list-style-type: none"> <li>• WL.Device.Wifi.PERMISSION: There is a permissions problem.</li> <li>• WL.Device.Wifi.DISABLED: The WiFi on the device is turned off.</li> <li>• WL.Device.Wifi.FAILED_START_SCAN: The device failed to start scanning. It is recommended that you retry after a few seconds.</li> </ul>

## Examples

### Policy example

```
{
  Geo: {
    timeout: 3000,
    enableHighAccuracy: true
  },
  Wifi: {
    interval: 3000,
    signalStrengthThreshold: 15,
    accessPointFilters: [
      {SSID: "net1"},
      {SSID: "net2", MAC: "*"}
    ]
  }
}
```

### Triggers example

```
{
  Geo: {
    userMoved: {
      type: "PositionChange",
      minChangeDistance: 20,
      callback: handleNewPosition,
      eventToTransmit: {
        event: {
          name: "UserMoved",
          userName: "Jane Doe"
        },
        transmitImmediately: true
      }
    }
  },
  Wifi: {
    userMoved: {
      type: "VisibleAccessPointsChanged",
      callback: handleVisibleWifiChange,
      eventToTransmit: {
        event: {
```

```

        name: "
VisibleAccessPointsChanged",
        username: "Jane Doe"
    },
    transmitImmediately: true
}
}
}

```

### **WL.Device.stopAcquisition:**

Stops the ongoing acquisition, and clears all trigger states.

#### **Syntax**

```
WL.Device.stopAcquisition
```

#### **Description**

The ongoing acquisition is stopped. The stop action is delegated to all relevant sensors, and all trigger states are cleared.

When the device context is cleared, `WL.Device.getContext` returns a null value.

#### **Parameters**

None.

### **WL.Device.Wifi.acquireVisibleAccessPoints:**

Acquires the currently visible access points. This API function is available only for Android.

#### **Syntax**

```
WL.Device.Wifi.acquireVisibleAccessPoints(onSuccess, onFailure, policy)
```

#### **Description**

The device attempts to acquire the currently visible access points. If the attempt is successful, and ongoing WiFi acquisition is enabled (using `WL.Device.startAcquisition`), the following actions take place:

- The device context is updated.
- An **onSuccess** callback is performed.
- Triggers are activated.

If there is no ongoing WiFi acquisition, only the **onSuccess** callback function is called.

**Note:** Because `acquireVisibleAccessPoints` might activate triggers, you should not call `acquireVisibleAccessPoints` from a trigger callback. Potentially, this could cause an endless loop of trigger evaluations leading to callbacks leading to `acquireVisibleAccessPoints` calls.

For details of the permissions required for Android, see “Location services API” on page 553.



## Parameters

Parameter	Description
<b>onSuccess</b>	A callback function that is invoked when the visible access points are acquired successfully. The appropriate WiFi access points array, filtered according to the provided <b>policy</b> setting, is passed as a parameter to this function.
<b>onFailure</b>	A callback function that is invoked if the acquisition is unsuccessful. The callback function receives a numeric code that corresponds to the following values: <ul style="list-style-type: none"><li>• <b>WL.Device.Wifi.PERMISSION</b>: There is a permissions problem.</li><li>• <b>WL.Device.Wifi.DISABLED</b>: The wifi on the device is turned off.</li><li>• <b>WL.Device.Wifi.FAILED_START_SCAN</b>: The device failed to start scanning. It is recommended that you retry after a few seconds.</li></ul>
<b>wifiPolicy</b>	A WiFi policy object that is used to configure the acquisition. This object must have the <b>accessPointFilters</b> property, and can optionally have the <b>signalStrengthThreshold</b> property (otherwise, the default value of 15 is used).  Note that the WiFi policy is different from the policy parameter passed to the <b>WL.Device.startAcquisition</b> function. If <b>aPolicy</b> is the <b>policy</b> parameter for <b>WL.Device.startAcquisition</b> , then <b>aPolicy.Wifi</b> would be the WiFi policy that could be passed to this API.

### **WL.Device.Wifi.getConnectionedAccessPoint:**

Acquires the currently connected WiFi access point information.

#### **Syntax**

```
WL.Device.Wifi.getConnectionedAccessPoint(onSuccess, onFailure)
```

#### **Description**

The device attempts to acquire the currently connected WiFi access point information. If the attempt is successful, the access point information is passed to the **onSuccess** callback function. The access point information includes the following fields:

- **SSID**. The connected network SSID
- **MAC**. The connected access point's MAC
- **signalStrength**. The connected access point's signal strength. This field is not available on iOS.

For details of the permissions required for Android and iOS, see “Location services API” on page 553.

### Parameters

Parameter	Description
<b>onSuccess</b>	A callback function that is invoked when access point connection information is acquired successfully. The acquisition result is passed as a parameter to this function.
<b>onFailure</b>	A callback function that is invoked if the acquisition is unsuccessful. The callback function receives a numeric code that corresponds to the following values: <ul style="list-style-type: none"><li>• <code>WL.Device.Wifi.PERMISSION</code>: There is a permissions problem.</li><li>• <code>WL.Device.Wifi.DISABLED</code>: The wifi on the device is turned off.</li><li>• <code>WL.Device.Wifi.FAILED_START_SCAN</code>: The device failed to start scanning. It is recommended that you retry after a few seconds.</li></ul>

### WL.Geo.getDistanceBetweenCoordinates:

Calculates the distance between two coordinates.

#### Syntax

`WL.Geo.getDistanceBetweenCoordinates (coordinate1, coordinate2)`

#### Description

The distance between two coordinates is calculated. The result is returned in meters, using a spherical model of the Earth.

### Parameters

Parameter	Description
<b>coordinate1</b>	The value of the <b>coordinate1</b> parameter is derived from the following properties: <ul style="list-style-type: none"><li>• <b>longitude</b>. The longitude, as a decimal number.</li><li>• <b>latitude</b>. The latitude, as a decimal number.</li></ul>
<b>coordinate2</b>	The value of the <b>coordinate2</b> parameter is derived from the following properties: <ul style="list-style-type: none"><li>• <b>longitude</b>. The longitude, as a decimal number.</li><li>• <b>latitude</b>. The latitude, as a decimal number.</li></ul>

### WL.Geo.getDistanceToCircle:

Calculates the distance of a coordinate from a circle.

## Syntax

WL.Geo.getDistanceToCircle (coordinate, circle, options)

## Parameters

Parameter	Description
<b>coordinate</b>	The value of the <b>coordinate</b> parameter is derived from the following properties: <ul style="list-style-type: none"><li>• <b>longitude</b>. The longitude, as a decimal number.</li><li>• <b>latitude</b>. The latitude, as a decimal number.</li></ul>
<b>circle</b>	The value of the <b>circle</b> parameter is derived from the following properties: <ul style="list-style-type: none"><li>• <b>longitude</b>. The longitude, as a decimal number.</li><li>• <b>latitude</b>. The latitude, as a decimal number.</li><li>• <b>radius</b>. The radius, in meters.</li></ul>
<b>options</b>	An optional parameter that has the following property: <ul style="list-style-type: none"><li>• <b>bufferZoneWidth</b>. The buffer zone width is measured in meters. It enlarges the radius of the circle by this amount. Negative values make the circle smaller. The default value is 0.</li></ul>

## Returned Value

The distance, in meters, to the circle, taking into account the buffer zone. The distance is positive for coordinates outside the circle, and negative for coordinates within the circle.

## WL.Geo.getDistanceToPolygon:

Calculates the distance of a coordinate from a polygon.

## Syntax

WL.Geo.getDistanceToPolygon (coordinate, polygon, options)

## Parameters

Parameter	Description
<b>coordinate</b>	The value of the <b>coordinate</b> parameter is derived from the following properties: <ul style="list-style-type: none"><li>• <b>longitude</b>. The longitude, as a decimal number.</li><li>• <b>latitude</b>. The latitude, as a decimal number.</li></ul>

Parameter	Description
<b>polygon</b>	The <b>polygon</b> parameter consist of an array of coordinates. It has the following properties: <ul style="list-style-type: none"> <li>• <b>longitude</b>. The longitude, as a decimal number.</li> <li>• <b>latitude</b>. The latitude, as a decimal number.</li> </ul>
<b>options</b>	An optional parameter that has the following property: <ul style="list-style-type: none"> <li>• <b>bufferZoneWidth</b>. The buffer zone width is measured in meters. It increases the size of the polygon in all directions by this amount. Negative values decrease the polygon's size. The default value is 0.</li> </ul>

### Returned Value

The distance, in meters, to the polygon, taking into account the buffer zone. The distance is positive for coordinates outside the polygon, and negative for coordinates within the polygon.

### WL.Geo.isInsideCircle:

Returns a Boolean value based on whether a coordinate lies within a circle, based on a given level of confidence.

### Syntax

WL.Geo.isInsideCircle (coordinate, circle, options)

### Parameters

Parameter	Description
<b>coordinate</b>	The value of the <b>coordinate</b> parameter is derived from the following properties: <ul style="list-style-type: none"> <li>• <b>longitude</b>. The longitude, as a decimal number.</li> <li>• <b>latitude</b>. The latitude, as a decimal number.</li> <li>• <b>accuracy</b>. The accuracy of the position.</li> </ul>
<b>circle</b>	The value of the <b>circle</b> parameter is derived from the following properties: <ul style="list-style-type: none"> <li>• <b>longitude</b>. The longitude, as a decimal number.</li> <li>• <b>latitude</b>. The latitude, as a decimal number.</li> <li>• <b>radius</b>. The radius, in meters.</li> </ul>

Parameter	Description
<b>options</b>	<p>An optional parameter that has the following properties:</p> <ul style="list-style-type: none"> <li>• <b>confidenceLevel</b>. An optional String parameter that has three possible values: <ul style="list-style-type: none"> <li>– low. The coordinate lies within the circle. Accuracy is not taken into account.</li> <li>– medium. The coordinate lies within the circle at approximately a 70% confidence interval. Accuracy is taken into account.</li> <li>– high. The coordinate lies within the circle at approximately a 95% confidence interval. Accuracy is taken into account.</li> </ul> <p>The default value is low.</p> </li> <li>• <b>bufferZoneWidth</b>. The buffer zone width is measured in meters. It enlarges the radius of the circle by this amount. Negative values make the circle smaller. The default value is 0.</li> </ul>

### Returned Value

The value `true` is returned if the coordinate lies within the circle, at the given level of confidence. The dimensions of the circle used in this check incorporate any changes specified by the **bufferZoneWidth** parameter.

### WL.Geo.isInsidePolygon:

Returns a Boolean value based on whether a coordinate lies within a polygon, based on a given level of confidence.

### Syntax

WL.Geo.isInsidePolygon (coordinate, polygon, options)

### Parameters

Parameter	Description
<b>coordinate</b>	<p>The value of the <b>coordinate</b> parameter is derived from the following properties:</p> <ul style="list-style-type: none"> <li>• <b>longitude</b>. The longitude, as a decimal number.</li> <li>• <b>latitude</b>. The latitude, as a decimal number.</li> <li>• <b>accuracy</b>. The accuracy of the position.</li> </ul>
<b>polygon</b>	<p>The value of the <b>polygon</b> parameter is derived from the following properties:</p> <ul style="list-style-type: none"> <li>• <b>longitude</b>. The longitude, as a decimal number.</li> <li>• <b>latitude</b>. The latitude, as a decimal number.</li> </ul>

Parameter	Description
<b>options</b>	<p>An optional parameter that has the following properties:</p> <ul style="list-style-type: none"> <li>• <b>confidenceLevel</b>. An optional String parameter that has three possible values: <ul style="list-style-type: none"> <li>– low. The coordinate lies within the polygon. Accuracy is not taken into account.</li> <li>– medium. The coordinate lies within the polygon at approximately a 70% confidence interval. Accuracy is taken into account.</li> <li>– high. The coordinate lies within the polygon at approximately a 95% confidence interval. Accuracy is taken into account.</li> </ul> <p>The default value is low.</p> </li> <li>• <b>bufferZoneWidth</b>. The buffer zone width is measured in meters. It increases the size of the polygon in all directions by this amount. Negative values decrease the size of the polygon. The default value is 0.</li> </ul>

### Returned Value

The value `true` is returned if the coordinate lies within the polygon, at the given level of confidence. The dimensions of the polygon used in this check incorporate any changes specified by the **bufferZoneWidth** parameter.

### WL.Geo.isOutsideCircle:

Returns a Boolean value based on whether a coordinate lies outside of a circle, based on a given level of confidence.

### Syntax

WL.Geo.isOutsideCircle (coordinate, circle, options)

### Parameters

Parameter	Description
<b>coordinate</b>	<p>The value of the <b>coordinate</b> parameter is derived from the following properties:</p> <ul style="list-style-type: none"> <li>• <b>longitude</b>. The longitude, as a decimal number.</li> <li>• <b>latitude</b>. The latitude, as a decimal number.</li> <li>• <b>accuracy</b>. The accuracy of the position.</li> </ul>

Parameter	Description
<b>circle</b>	<p>The value of the <b>circle</b> parameter is derived from the following properties:</p> <ul style="list-style-type: none"> <li>• <b>longitude</b>. The longitude, as a decimal number.</li> <li>• <b>latitude</b>. The latitude, as a decimal number.</li> <li>• <b>radius</b>. The radius, in meters.</li> </ul>
<b>options</b>	<p>An optional parameter that has the following properties:</p> <ul style="list-style-type: none"> <li>• <b>confidenceLevel</b>. An optional String parameter that has three possible values: <ul style="list-style-type: none"> <li>– low. The coordinate lies outside the circle. Accuracy is not taken into account.</li> <li>– medium. The coordinate lies outside the circle at approximately a 70% confidence interval. Accuracy is taken into account.</li> <li>– high. The coordinate lies outside the circle at approximately a 95% confidence interval. Accuracy is taken into account.</li> </ul> <p>The default value is low.</p> </li> <li>• <b>bufferZoneWidth</b>. The buffer zone width is measured in meters. It enlarges the radius of the circle by this amount. Negative values make the circle smaller. The default value is 0.</li> </ul>

### Returned Value

The value true is returned if the coordinate lies outside the circle, at the given level of confidence. The dimensions of the circle used in this check incorporate any changes specified by the **bufferZoneWidth** parameter.

### WL.Geo.isOutsidePolygon:

Returns a Boolean value based on whether a coordinate lies outside a polygon, based on a given level of confidence.

### Syntax

WL.Geo.isOutsidePolygon (coordinate, polygon, options)



## Parameters

Parameter	Description
<b>coordinate</b>	The value of the <b>coordinate</b> parameter is derived from the following properties: <ul style="list-style-type: none"><li>• <b>longitude</b>. The longitude, as a decimal number.</li><li>• <b>latitude</b>. The latitude, as a decimal number.</li><li>• <b>accuracy</b>. The accuracy of the position.</li></ul>
<b>polygon</b>	The value of the <b>polygon</b> parameter is derived from the following properties: <ul style="list-style-type: none"><li>• <b>longitude</b>. The longitude, as a decimal number.</li><li>• <b>latitude</b>. The latitude, as a decimal number.</li></ul>
<b>options</b>	An optional parameter that has the following properties: <ul style="list-style-type: none"><li>• <b>confidenceLevel</b>. An optional String parameter that has three possible values:<ul style="list-style-type: none"><li>– low. The coordinate lies outside the polygon. Accuracy is not taken into account.</li><li>– medium. The coordinate lies outside the polygon at approximately a 70% confidence interval. Accuracy is taken into account.</li><li>– high. The coordinate lies outside the polygon at approximately a 95% confidence interval. Accuracy is taken into account.</li></ul>The default value is low.</li><li>• <b>bufferZoneWidth</b>. The buffer zone width is measured in meters. It increases the size of the polygon in all directions by this amount. Negative values decrease the size of the polygon. The default value is 0.</li></ul>

### Returned Value

The value `true` is returned if the coordinate lies outside the polygon, at the given level of confidence. The dimensions of the polygon used in this check incorporate any changes specified by the **bufferZoneWidth** parameter.

### 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.

**Note:** This API only applies to the Logger running on client-side JavaScript, not the server-side Logger that runs inside IBM Worklight Adapters.

## Setting Logger Windows for Desktop Environments

### Adobe AIR

To use the logger in Adobe Air applications, enable the Adobe Air Introspector. See the Adobe Air Introspector documentation for instructions.

### Windows 8

No configuration is required. Debug messages are displayed in the Microsoft Visual Studio 2012 console.

### Configuring the Worklight Logger:

You can configure how the IBM Worklight Logger runs on a range of client operating systems by modifying `WL.Logger` methods.

*Enable Logger output:*

Enable IBM Worklight Logger output to the client console.

Enable log output to the console:

```
WL.Logger.on();
```

Explicitly send configuration parameters:

```
var options = {
  whitelist: [],
  blacklist: []
};

WL.Logger.on(options);
```

Turn off logging:

```
WL.Logger.off();
```

*Start Logger when IBM Worklight starts:*

You can set the logger to start when IBM Worklight starts.

You can set IBM Worklight options to enable the logger when IBM Worklight starts.

```
var wlInitOptions = {

  //... other options not related to the logger

  logger : {enabled: true, level: 'debug', stringify: true,
            tag: {level: false, pkg: true}, whitelist: [], blacklist: []}
}

WL.Client.init(wlInitOptions);
```

The only parameters required for `wlInitOptions` to enable the logger are `logger {enable: true}` or `enableLogger: true`. All of the other Logger parameters are explicitly set to the default values.

Select log levels:

You can select from among various log levels.

### Debug

Add a debug message.

```
WL.Logger.debug('Loop finished');  
//Loop finished
```

### Log

Add a log message.

```
WL.Logger.log('Got', response.statusCode, 'from server.');
```

//Got 200 from the server.

### Info

Add an informative message.

```
WL.Logger.info('Public IP address is', getIpAddress());  
//Public IP address is 192.168.1.102
```

### Warn

Add a warning message.

```
if (!window.indexedDB) {  
  WL.Logger.warn('IndexedDB not supported, falling back to LocalStorage.');
```

//IndexedDB not supported, falling back to LocalStorage.

```
}
```

### Error

Add an error message.

```
try {  
  //code that may throw new Error('Something failed here.')} catch (e) {  
  WL.Logger.error('Caught an exception', e);  
  //Caught an exception Error: Something failed here.  
}
```

Log different data types:

You can log a range of data types including numbers, strings, and arrays

### Strings

```
WL.Logger.info('Hello', 'world.');
```

//Hello World.

### Booleans

```
WL.Logger.info(true, false);  
//true false
```

### Numbers

```
WL.Logger.info(1,2,3.14,4,5,6,7);  
//1 2 3.14 4 5 6 7
```

### Arrays

```
WL.Logger.info([1,2,3], [[1,2,3], [1,2,3]])  
//[1,2,3] [[1,2,3], [1,2,3]]
```

### Objects

```
WL.Logger.info({hello: 'world'}, {hey: {test: [1,2,3, {hello: 'world'}]}});  
//{"hello": "world"} {"hey": {"test": [1,2,3, {"hello": "world"}]}}
```

### Exceptions

```
var e = new Error('Something failed');
var te = new TypeError('Wrong type');
WL.Logger.info(e, te);
//Error: Something failed TypeError: Wrong type
```

#### **undefined**

```
var undef;
WL.Logger.info(undefined, undef);
//undefined undefined
```

#### **null**

```
var n = null;
WL.Logger.info(null, n);
//null null
```

#### **Any Combination**

```
WL.Logger.info('Hey', 1, 2, true, false, [1,2,3], {hey: 'world'}, new Error('Uh oh'), undefined)
//Hey 1 2 true false [1,2,3] {"hey": "world"} Error: Uh oh undefined null
```

*Set Logger priority:*

You can configure the Logger to display only warning and error log messages.

To display only warning and error log messages, set the priority of the logger to 'warn' or 200.

```
WL.Logger.on({level: 'warn'});
```

```
var WARN = 200;
WL.Logger.on({level: WARN});
```

Possible string values:

```
'log',
'info',
'warn',
'error'
```

**Note:** These values are not case-sensitive, for example LOG and Log are also possible values.

Possible int values:

```
100 (error)
200 (warn)
300 (info)
400 (log)
500 (debug)
```

*Filter log levels:*

You can filter logs to display only logs of one level.

Show only info-level logs:

```
WL.Logger.on({level: ['info']});
```

Show only info-level and error-level logs:

```
WL.Logger.on({level: ['warn', 'error']});
```

By default, display logs of all levels:

```
WL.Logger.on(); //same as: WL.Logger.on({level: []})
```

*Log package whitelist and blacklist:*

You can associate a set of log messages with a specific part of the application.

Add packages to the whitelist to include the packages in logging:

```
WL.Logger.on({whitelist: ['wl.jsonstore']});
```

Associate a log message with a package, and log a message:

```
var JSONSTORE_PKG = 'wl.jsonstore';
```

```
WL.Logger.ctx({pkg: JSONSTORE_PKG}).info('JSONStore started');  
//JSONStore started
```

```
WL.Logger.info('Hey!'); //Ignored
```

```
WL.Logger.ctx({pkg: JSONSTORE_PKG}).warn('JSONStore finished executing find.');  
//JSONStore finished executing find.
```

```
WL.Logger.ctx({pkg: 'wl.analytics'}).warn('Hello.');//Ignored
```

To exclude packages from logging, add them to the blacklist.

```
WL.Logger.on({blacklist: ['wl.jsonstore', 'wl.analytics']});
```

```
WL.Logger.ctx({pkg: 'wl.jsonstore'}).info('Hello world');//Ignored
```

```
WL.Logger.info('Hey!');//Not ignored.
```

```
WL.Logger.ctx({pkg: 'wl.analytics'}).info('Hey world');//Ignored
```

```
WL.Logger.ctx({pkg: 'wl.adapter'}).info('Hola');//Not ignored
```

You can list the packages that are on the whitelist or the blacklist:

```
WL.Logger.status();
```

```
//{ enabled : true, stringify: true, whitelist : [], blacklist : [], level : [], pkg : '', tag: {} }
```

The list of keys returned match the options that you can pass to **WL.Logger.on**.

*Create log for package:*

You can create a logger for a specific package.

To avoid writing a package name every time a log message is written, you can create a logger for a specific package.

```
var JSONStoreLogger = new WL.Logger.create({pkg: 'wl.jsonstore'});
```

```
JSONStoreLogger.info('Hello', 'world.');
```

```
//Hello world.
```

```
JSONStoreLogger.warn(1,2,3,4);
```

```
//1 2 3 4
```

```
var AnalyticsLogger = new WL.Logger.create({pkg: 'wl.analytics'});
```

```
AnalyticsLogger.error(new Error('BOOM.'));
```

```
//Error: BOOM.
```

*Stringify:*

You can convert arguments to strings using the **stringify** function.

Some environments, for example the Xcode console, can print the arguments passed to the logger only if the arguments are converted to strings and

concatenated first. Other environments, for example Google Chrome, can provide better visualization of arguments if the arguments are not turned into strings and concatenated.

Turn on the stringify function:

```
var obj = {name: 'carlos', age: 100};  
WL.Logger.on({stringify: true}); //default
```

```
> WL.Logger.log(obj);  
{"name":"carlos","age":100}
```

```
WL.Logger.on({stringify: false});
```

```
> WL.Logger.log(obj);  
Object {name: "carlos", age: 100}
```

*Callback:*

You can pass a callback function to `WL.Logger.on` that will be called after every log message.

The callback takes these arguments:

- message (string or array)
- priority (string)
- package (string)

If `stringify : true` is set, the message is a string. Otherwise it is an array. If the package is not defined, the message is an empty string.

Send all log messages to a backend using `jQuery.ajax`:

```
var ajaxSender = function (message, priority, pkg) {  
    $.ajax({  
        url: 'http://localhost:3000/logs'  
        type: 'POST',  
        data: {  
            message: message,  
            priority: priority,  
            pkg: pkg  
        }  
    });  
};
```

```
WL.Logger.on({callback: ajaxSender});
```

Sends all log messages to a backend using a “`WL.Client.invokeProcedure`” on page 513

```
var adapterSender = function (message, priority, pkg) {  
    var invocationData = {  
        adapter: 'Logger',  
        procedure: 'sendLogs',  
        parameters: [message, priority, pkg]  
    }  
};
```

```
    WL.Client.InvokeProcedure(invocationData);
};
```

```
WL.Logger.on({callback: adapterSender});
```

Log JavaScript errors for a specific package using “WL.Client.logActivity” on page 516:

```
var activitySender = function (message, priority, pkg) {
    if (priority === 'ERROR' && pkg === 'my.app.db') {
        WL.Client.logActivity(message);
    }
};
```

```
WL.Logger.on({callback: activitySender});
```

*Log message tags:*

You can add context to a log message by appending the level tag, the package tag, or both.

Add level and error tags to a defined package:

```
WL.Logger.on({tag: {level: true, pkg: true} });
```

```
WL.Logger.info('Hello');
// [INFO] Hello
```

```
WL.Logger.ctx({pkg: 'wl.jsonstore'}).error('Hey');
// [ERROR] [wl.jsonstore] Hey
```

Turn off the tags:

```
WL.Logger.on({tag: {level: false, pkg: false} });
```

```
WL.Logger.info('Hello');
// Hello
```

```
WL.Logger.ctx({pkg: 'MYPKG'}).error('Hey');
// Hey
```

*Method chaining:*

You can invoke multiple method calls by chaining logger methods.

You can chain these logger methods:

```
WL.Logger.on
WL.Logger.off
WL.Logger.ctx
```

This example carries out these steps:

- Turns the logger off
- Turns the logger on
- Sets the package context to com.my.app
- Logs Hello.

```
WL.Logger.off().on().ctx({pkg: 'com.my.app'}).log('Hello')
// '[com.my.app] Hello'
```



### Pretty-print JSON objects:

You can format JSON objects by enabling stringify.

By enabling “Stringify” on page 577 (stringify: true) you can display JSON objects in a more readable format.

```
var obj = {name: 'carlos', age: 100};  
WL.Logger.on({stringify: true, pretty: true});
```

```
> WL.Logger.debug(obj)  
Q {  
  "name": "carlos",  
  "age": 100  
}
```

```
WL.Logger.on({pretty: false});
```

```
> WL.Logger.debug(obj)  
Q {"name":"carlos","age":100}
```

### Print stack traces:

You can print stack traces for certain objects.

You can print stack traces for an object if the object is an instance of Error (if (object instanceof Error) evaluates true) and “Stringify” on page 577 is enabled (stringify: true).

```
var object = new Error('Boom');  
WL.Logger.on({stringify: true, stacktrace: true});  
WL.Logger.error(object);
```

```
✖ ▶ Error: Boom  
  at Object.<anonymous> (http://localhost:10080/wlp  
  at Object.Test.run (http://localhost:10080/wlproj  
  at Test.queue.bad (http://localhost:10080/wlproj/  
  at process (http://localhost:10080/wlproj/apps/se
```

```
WL.Logger.on({stacktrace: false});  
WL.Logger.error(object);
```

```
WL.Logger.error(object);  
▶ Error: BOOM
```

### Logger Android check and override:

Logger checks the operating system on which it is running to determine whether to use the Android logger. You can override this behavior.

By default, WL.Logger checks the operating system at run time, and if it is running on Android it attempts to use the cordova plug-in. If the plug-in fails, it falls back to console.log. There are several differences between the cordova plug-in logger and console.log:

#### Cordova plugin logger

Asynchronous

Provides better output in LogCat

Requires that the deviceready event previously fired.

#### Console log

Synchronous

#### Native logger + LogCat:

```
com.wlapp    wlapp    hey1
com.wlapp    wlapp    hey2
com.wlapp    wlapp    hey3
com.wlapp    wlapp    hey4
com.wlapp    wlapp    hey5
```

#### console.log + LogCat:

```
com.wlapp CordovaLog hey1
com.wlapp CordovaLog hey2
com.wlapp CordovaLog hey3
com.wlapp CordovaLog hey4
com.wlapp CordovaLog hey5
```

To override the Android check, do one of the these:

- Pass `android: false` to `WL.Logger.on`
- Add `android: false` to the logger key for `wlInitOptions`

**Note:** Logs with `WL.Logger.log` are treated as verbose by LogCat.

#### Environment-specific settings:

You can specify that logger options are selected according to the client environment.

Use `initOptions.js` to select options for each environment:

```
//General init options
var wlInitOptions = {connectOnStartup : false};

//General logger options
wlInitOptions.logger = {enabled: true, stringify: false};

//Environment specific logger options
if (WL.Client.getEnvironment() === WL.Environment.IPHONE) {
    wlInitOptions.logger.stringify = true;
}

WL.Client.init(wlInitOptions);
```

For other options that can take advantage of environment-specific settings, see “Logger Android check and override” on page 580 “Callback” on page 578, “Log message tags” on page 579, “Log package whitelist and blacklist” on page 577, and “Select log levels” on page 575.

As examples, you can use environment-specific options settings to specify no logs in production, selected logs for development, and only error logs for testing:

```
//Change accordingly
var CURRENT_ENV = 'production';

//General init options
var wlInitOptions = {connectOnStartup : false};

//General logger options
wlInitOptions.logger = {enabled: true};

//Give your application a small speed boost by not logging in production
if (CURRENT_ENV === 'production') {
    wlInitOptions.logger.enabled = false;
}

WL.Client.init(wlInitOptions);
```

*JavaScript module example:*

View an example of how to use WL.Logger to add log messages to a JavaScript module.

This example demonstrates how to use WL.Logger to add log messages to a JavaScript module by using these methods:

- `myApp.Greeter.start()` Initializes the module.
- `myApp.Greeter.sayHello(name)` Alerts a greeting to the name that is passed.

The example uses the default `initOptions.js` file for IBM Worklight 6.0. This list contains some of the principles that are demonstrated by the example:

- The module, **myApp.Greeter.js**, uses the JavaScript Revealing Module Pattern, however the concepts in the example apply no matter how you structure your JavaScript code.
- By using `WL.Logger.create({pkg: '[package-name]'})` you can create a `LogInstance` linked to a package.
- Using a short variable name such as `l` for the `LogInstance` makes it easier to write logs, for example: `(l.log(msg), l.info(msg))`
- You can log errors by using the JavaScript try/catch block (synchronous code) and failure callbacks (asynchronous code).
- You can avoid problems by using correct log levels, precise package names, and by filtering as necessary.

#### **myApp.Greeter.js**

```
var myApp = myApp || {};
myApp.Greeter = (function (WL) {

    //ECMAScript 5 strict mode
    'use strict';

    //Dependencies
    var WL_LOGGER = WL.Logger;
    //... other dependencies
```

```

//Constants
var PKG_NAME = 'myApp.Greeter';
var DEFAULT_NAME = 'Stranger';
//... other constants

//LogInstance local to this module
var l = WL_LOGGER.create({pkg: PKG_NAME});

//Private function to the module that does validation and alerts a name
var __alertName = function (name) {

    l.debug('Calling __alertName with name =', name);

    if (typeof name !== 'string' || name.length < 1) {
        l.warn('Name was not a string or empty string, setting name to', DEFAULT_NAME);
        name = DEFAULT_NAME;
    }

    else if (name === '*') {
        throw new Error('Name can not be *');
    }

    //Assume 'alert' is always a global function that exists
    alert('Hello ' + name);

    l.debug('Done calling __alertName');
};

//Public API function that does initialization
var _start = function () {

    l.info('Started', PKG_NAME , 'module');

    //... init code
};

//Public API function that alerts 'Hello [name]'
var _sayHello = function (name) {

    l.debug('Starting _sayHello');

    try {
        __alertName(name);
    } catch (e) {
        //Log any errors
        l.error(e);
    }

    l.debug('End _sayHello');
};

//Public API
return {
    start : _start,
    sayHello: _sayHello
};

}(WL)); //Pass global variables to the module

```

**[app-name].js**

```
function wlCommonInit () {
    myApp.Greeter.start(); //Start our application's greeter module
    myApp.Greeter.sayHello('Carlos'); //should alert 'Hello Carlos'
    myApp.Greeter.sayHello(); //should alert 'Hello Stranger'
    myApp.Greeter.sayHello('*'); //should log an error
}
```

### [app-name].html

```
<!-- ... other html tags -->
<body id="content" style="display: none;">

    <!-- ... application UI -->

    <script src="js/initOptions.js"></script>
    <script src="js/myApp.Greeter.js"></script>
    <script src="js/[app-name].js"></script>

    <!-- ... other script tags -->
</body>
```

```
[myApp.Greeter] Started myApp.Greeter module
q [myApp.Greeter] Starting _sayHello
q [myApp.Greeter] Calling __alertName with name = Carlos
q [myApp.Greeter] Done calling __alertName
q [myApp.Greeter] End _sayHello
q [myApp.Greeter] Starting _sayHello
q [myApp.Greeter] Calling __alertName with name = undefined
⚠ [myApp.Greeter] Name was not a string or empty string, setting name to Stranger
q [myApp.Greeter] Done calling __alertName
q [myApp.Greeter] End _sayHello
q [myApp.Greeter] Starting _sayHello
q [myApp.Greeter] Calling __alertName with name = *
❌ ▶ [myApp.Greeter] Error: Name can not be *
```

Figure 67. Log output

### WL.Logger methods:

WL.Logger methods output a specified message to the environment log.

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.

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 `logger: {enabled: false}` in the object `wlInitOptions` in the `initOptions.js` file.

```
var wlInitOptions = {
    // # Should application produce logs
    // # Default value is true
    logger: {enabled: false}
}
```

*WL.Logger.on:*

Turns on the logger or changes status.

## Syntax

`WL.Logger.on(options)`

## Description

Turns on the logger with various parameters that define the logger's behavior.

## Parameters

Table 97.

Parameter	Description
<b>options (object)</b>	Defines the logger's behavior.
<b>options.enabled (boolean)</b>	enabled (true) or disabled (false). Default is true.
<b>options.stringify (boolean)</b>	Turns arguments to strings and concatenates them (true) or leaves arguments as an array (false). Default is true.
<b>options.android (boolean)</b>	Uses the native logger on Android (true) or uses <code>console.log</code> (false). Default: is true if the current environment is Android, false otherwise.
<b>options.callback (function)</b>	Called after a log message is sent to the console with the following parameters: <ul style="list-style-type: none"><li>• <code>message</code> (string or array) The log message. String if <code>stringify</code> is true, array otherwise.</li><li>• <code>priority</code> (string) Log priority.</li><li>• <code>package</code> (string) Package that is associated with the log or empty string.</li></ul>
<b>options.pkg (string)</b>	Associates log messages with a package. By default does not associate (includes an empty string).
<b>options.tag (object)</b>	Contains keys: <code>level</code> and <code>tag</code> .
<b>options.tag.level (boolean)</b>	Appends <code>log level</code> tag to the string log message (true) or does nothing (false). Default is false.
<b>options.tag.pkg (boolean)</b>	Appends package name string message (true) or does nothing (false). Default is false.
<b>options.whitelist (array of strings)</b>	List of packages from which to log messages. By default logs all messages (empty array).
<b>options.blacklist (array of strings)</b>	List of packages to ignore log messages from. By default does not ignore any packages (empty array).
<b>options.level (array of strings or string or number)</b>	A list of one or more of the following items: <ul style="list-style-type: none"><li>• the log levels to show</li><li>• the name of the log level at which to set the logger</li><li>• the Numeric Priority Level at which to set the logger.</li></ul> Default: Show all logs (empty array).

### **Return Value**

#### **WL.Logger**

*WL.Logger.off:*

Turns off the logger.

### **Description**

A shorthand term for `WL.Logger.on()` with enabled option set to false.

### **Syntax**

`WL.Logger.off()`

### **Parameters**

None

### **Return Value**

#### **WL.Logger**

*WL.Logger.status:*

Shows the status of the logger.

### **Syntax**

`WL.Logger.status()`

### **Parameters**

None

### **Return Value**

Options. See arguments in “`WL.Logger.on`” on page 584

*WL.Logger.ctx:*

Updates the status of the logger.

### **Syntax**

`WL.Logger.ctx()`

### **Parameters**

Options

### **Return Value**

Options. See arguments in “`WL.Logger.on`” on page 584

*WL.Logger.debug:*

Prints arguments to the console.



**Syntax**

`WL.Logger.debug()`

**Parameters**

One or more messages of any data type.

**Priority**

500

**Level**

'DEBUG'

*WL.Logger.log:*

Prints arguments to the console

**Syntax**

`WL.Logger.log()`

**Parameters**

One or more message of any data type

**Priority**

400

**Level**

'LOG'

*WL.Logger.info:*

Prints arguments to the console.

**Syntax**

`WL.Logger.info()`

**Parameters**

One or more messages of any data type.

**Priority**

300

**Level**

'INFO'

*WL.Logger.warn:*

Prints arguments to the console.

**Syntax**`WL.Logger.warn()`**Parameters**

One or more messages of any data type.

**Priority**

200

**Level**

'WARN'

*WL.Logger.error:*

Prints arguments to the console.

**Syntax**`WL.Logger.error()`**Parameters**

One or more message of any data type.

**Priority**

100

**Level**

'ERROR'

*WL.Logger.create:*

Creates a new log instance.

**Syntax**`WL.Logger.create()`**Parameters**

*Table 98.*

Parameter	Description
<b>Options (object)</b>	
<b>Options.pkg (string)</b>	Name of the new package. Default is no package (empty string).

**Returns**

"LogInstance" on page 589

*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
<b>object</b>	Optional. An object that can contain any of the following key/value pairs: <b>maxFileSize: integer</b> Minimum value is 10000 (bytes) <b>level: String</b> Any of the following values: debug, log, info, warn, error <b>capture: boolean</b> true or false

#### Return Value

None

*LogInstance:*

Displays a message when a log instance is created.

## Description

See “Create log for package” on page 577

*LogInstance.debug:*

Prints arguments to the console.

## Syntax

`LogInstance.debug()`

## Parameters

One or more messages of any data type.

## Priority

500

## Level

'DEBUG'

## Package

Package name set by “WL.Logger.create” on page 588

*LogInstance.log:*

Prints arguments to the console.

## Syntax

`LogInstance.log()`

## Parameters

One or more messages of any data type.

## Priority

400

## Level

'LOG'

## Package

Package name set by “WL.Logger.create” on page 588

*LogInstance.info:*

Prints arguments to the console.

## Syntax

`LogInstance.info()`

**Parameters**

One or more messages of any data type.

**Priority**

300

**Level**

'INFO'

**Package**

Package name set by "WL.Logger.create" on page 588

*LoginInstance.warn:*

Prints arguments to the console.

**Syntax**

`LogInstance.warn()`

**Parameters**

One or more messages of any data type.

**Priority**

200

**Level**

'WARN'

**Package**

Package name set by "WL.Logger.create" on page 588

*LoginInstance.error:*

Prints arguments to the console

**Syntax**

`LogInstance.error()`

**Parameters**

One or more messages of any data type

**Priority**

100

## Level

'ERROR'

## Package

Package name set by “WL.Logger.create” on page 588

## 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.

SMS push notifications are supported on iOS, Android, Windows Phone 7.5 (deprecated in IBM Worklight V6.0.0), 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 413 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

#### **Parameters**

*Table 99. WL.Client.Push.isSMSSubscribed parameters*

Parameter	Description
<code>alias</code>	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 100. WL.Client.Push.isSubscribed parameters*

Parameter	Description
<code>alias</code>	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', notificationArr
}

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.

#### 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 101. WL.Client.Push.registerEventSourceCallback parameters

Parameter	Description
<b>alias</b>	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.
<b>adapter</b>	Mandatory string. The name of the adapter that contains the event source.
<b>eventSource</b>	Mandatory string. The name of the event source.
<b>callback</b>	Mandatory function. The function that is called if a notification arrives. The function receives two parameters when invoked: <p><b>props</b> A JSON block, containing the notification properties from the platform</p> <p><b>payload</b> A JSON block, containing other data that is sent from the IBM Worklight Server</p>

### 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 102. WL.Client.Push.subscribe parameters

Parameter	Description
<b>alias</b>	Mandatory string. The event source alias, as defined in the invocation of <code>WL.Client.Push.onReadyToSubscribe</code> .
<b>options</b>	Optional JSON block, containing: <ul style="list-style-type: none"><li>• Custom subscription parameters that are supported by the event source in the adapter</li><li>• <code>onFailure</code>: 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.</li><li>• <code>onSuccess</code>: JavaScript function, called when the subscription was successful. Default value is an empty function.</li></ul>

```
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 103. WL.Client.Push.subscribeSMS parameters

Parameter	Description
<b>alias</b>	Mandatory string. A short ID defining the event source.
<b>adapterName</b>	Mandatory String. The name of the adapter that sets up the event source and communicates with the Worklight server.
<b>eventSource</b>	Mandatory String. The name of the event source.
<b>phoneNumber</b>	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.
<b>options</b>	Optional JSON block, containing: <ul style="list-style-type: none"><li>• Custom subscription parameters that are supported by the event source in the adapter</li><li>• <code>onFailure</code>: 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.</li><li>• <code>onSuccess</code>: a JavaScript function, called when the subscription was successful. The default value is an empty function.</li></ul>

```
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 104. *WL.Client.Push.unsubscribe* parameters

Parameter	Description
<b>alias</b>	Mandatory string. The event source alias, as defined in the invocation of <code>WL.Client.Push.onReadyToSubscribe</code> .
<b>options</b>	Optional JSON block, containing: <ul style="list-style-type: none"><li>• Custom subscription parameters that are supported by the event source in the adapter</li><li>• <code>onFailure</code>: 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.</li><li>• <code>onSuccess</code>: a JavaScript function, called when the subscription was successful. Default value is an empty function.</li></ul>

### **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 105. *WL.Client.Push.unsubscribeSMS* parameters

Parameter	Description
<b>alias</b>	Mandatory string. The alias defined when subscribing.
<b>options</b>	Optional JSON block, containing: <ul style="list-style-type: none"><li>• <code>onFailure</code>: a JavaScript function, called when communication with the IBM Worklight Server did not succeed. Default value is a function that logs the failure.</li><li>• <code>onSuccess</code>: a JavaScript function, called when unsubscription was successful. Default value is a function that logs the success.</li></ul>

### **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 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.

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 106. options object properties

Property	Description
<b>onSuccess</b>	<p>Optional. The function to be invoked on successful completion of the asynchronous call.</p> <p>The syntax of the onSuccess function is:  <code>success-handler-function(response)</code></p> <p>where <i>response</i> is an object that contains at a minimum the following property:</p> <p><b>invocationContext</b>            The invocationContext object that was originally passed to the Worklight Server in the options object, or undefined if no invocationContext object was passed.</p> <p><b>status</b> The HTTP response status</p> <p><b>Note:</b> For methods for which the <i>response</i> object contains additional properties, these properties are detailed as part of the description of the specific method.</p>

Table 106. *options* object properties (continued)

Property	Description
<b>onFailure</b>	<p data-bbox="966 256 1438 428">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.</p> <p data-bbox="966 457 1438 541"><b>Note:</b> The function is not called for client-side errors that stop the execution by throwing an exception.</p> <p data-bbox="966 571 1398 634">The syntax of the onFailure function is: failure-handler-function(response)</p> <p data-bbox="966 663 1438 726">where <i>response</i> is an object that contains the following properties:</p> <p data-bbox="966 735 1170 756"><b>invocationContext</b></p> <p data-bbox="1060 764 1438 936">The invocationContext object that was originally passed to the Worklight Server in the options object, or undefined if no invocationContext object was passed.</p> <p data-bbox="966 953 1073 974"><b>errorCode</b></p> <p data-bbox="1060 982 1390 1121">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.</p> <p data-bbox="966 1146 1062 1167"><b>errorMsg</b></p> <p data-bbox="1060 1176 1438 1348">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.</p> <p data-bbox="966 1365 1344 1386"><b>status</b> The HTTP response status</p> <p data-bbox="966 1394 1438 1499"><b>Note:</b> For methods for which the <i>response</i> object contains additional properties, these properties are detailed as part of the description of the specific method.</p>

Table 106. *options* object properties (continued)

Property	Description
<b>invocationContext</b>	<p>Optional. An object that is returned to the success and failure handlers.</p> <p>The <code>invocationContext</code> object is used to preserve the context of the calling asynchronous service upon returning from the service.</p> <p>For example, the <code>invokeProcedure</code> method might be called successively, using the same success handler. The success handler needs to be able to identify which call to <code>invokeProcedure</code> is being handled. One solution is to implement the <code>invocationContext</code> object as an integer, and increment its value by one for each call of <code>invokeProcedure</code>. When it invokes the success handler, Worklight passes it the <code>invocationContext</code> object of the <code>options</code> object associated with the <code>invokeProcedure</code> method. The value of the <code>invocationContext</code> object can be used to identify the call to <code>invokeProcedure</code> with which the results that are being handled are associated.</p>

## 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 (deprecated in IBM Worklight V6.0.0), 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 the title or icon of an item, or enable or disable an item.

#### Syntax

```
WL.Item.setTitle (title)
WL.Item.setImagePath (imagePath)
WL.Item.setEnabled (enabled)
```

#### Description

Changes the title or icon of an item, or enables or disables an item.



**Note:** You cannot instantiate a new `WL.Item` object; you can receive one as a result of calling `WL.OptionsMenu.getItem()`.

### Parameters

Table 107. *WL.Item* method parameters

Parameter	Description
<b>title</b>	Mandatory string. The title of the item.
<b>iconPath</b>	Mandatory string. The path to the icon.  See <code>WL.OptionsMenu.addItem</code> for an explanation of the icon path and format for Android, Windows Phone 7.5 (deprecated in IBM Worklight V6.0.0), Windows Phone 8, and Windows 8.
<b>enabled</b>	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 108. *WL.OptionsMenu.addItem* parameters

Parameter	Description
<b>id</b>	Mandatory string. Identifies the item.
<b>callbackFunction</b>	Mandatory JavaScript function. The callback function that is invoked when the user selects the item in the options menu.
<b>title</b>	Mandatory string. The title of the item.

Table 108. WL.OptionsMenu.addItem parameters (continued)

Parameter	Description
<p><b>options</b></p>	<p>The <b>options</b> parameter is mandatory, and the <b>image</b> property within it is also mandatory.</p> <p>Contains the following fields:</p> <p><b>image</b></p> <p>For Android, this field contains the name of the resource that contains the icon image for the item. For Windows Phone 7.5 (deprecated in IBM Worklight V6.0.0), Windows Phone 8, and Windows 8, this field contains the path to the icon image for the item.</p> <p>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.</p> <p>For Windows Phone 8, the path starts from the folder /nativeResources/applicationBar. Do not explicitly mention this folder within the path.</p> <p>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.</p> <p>The same set of images can be used for Android, Windows Phone 7.5, Windows Phone 8, and Windows 8.</p> <p>For Android, the image size depends on the density of the device. See the Android Options Menu documentation for details.</p> <p>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.</p> <p><b>Enabled</b></p> <p>Optional Boolean. Defines whether the item is enabled or disabled.</p>

**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: true});

// Windows 8
WL.OptionsMenu.addItem("first", function(){alert("hello one")}, 'one', {image: 'one.png', enabled: true});
```

### 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 109. 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 (deprecated in IBM Worklight V6.0.0) and Windows Phone 8, the default opacity of the application bar is 1.0 (opaque).

#### Parameters

A JSON block with the following properties:

Table 110. WL.OptionsMenu.init parameters

Property	Description
msg	Mandatory. The string to be displayed in the logger window.

Table 110. *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 (deprecated in IBM Worklight V6.0.0) 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.

## 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 (deprecated in IBM Worklight V6.0.0) 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 111. WL.OptionsMenu.removeItem parameters

Parameter	Description
<b>id</b>	Mandatory string. Identifies the item to be removed.

## Return Value

None

```
WL.OptionsMenu.removeItem("first");
```

## WL.OptionsMenu.removeItem:

Removes all items from the options menu or application bar.

## Syntax

```
WL.OptionsMenu.removeItem()
```

## Description

Removes all items from the options menu or application bar. Can be called only after the menu is initialized.

### Parameters

None

### Return Value

None

```
WL.OptionsMenu.removeItem();
```

### 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 112. WL.OptionsMenu.setEnabled parameters

Parameter	Description
<b>isEnabled</b>	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 (deprecated in IBM Worklight V6.0.0) 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.

## Parameters

Table 113. *WL.OptionsMenu.setOpacity* parameters

Parameter	Description
<b>number</b>	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 114. *WL.OptionsMenu.setVisible* parameters

Parameter	Description
<b>isVisible</b>	Mandatory Boolean.  true: Makes the menu visible on pressing <b>Menu</b>  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.



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 115. WL.SimpleDialog.show parameters

Parameter	Description
<b>title</b>	Mandatory string. The title of the dialog box.
<b>text</b>	Mandatory string. The text to show in the dialog box.
<b>buttons</b>	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:  <b>text</b> Mandatory string. The text of the button.  <b>handler</b> Optional function. The function that is invoked when the button is pressed.
<b>options</b>	Ignored on iPhone and Android.  Optional. An object of the following form: <pre>{   title: string,   text: string, }</pre>

#### Returned Values

None.

*Example*

#### Example

```
WL.SimpleDialog.show(  
  "My Title", "My Text",  
  [{text: "First Button", handler: function() {WL.Logger.debug("First button pressed"); }  
}]  
)
```

## 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.

### 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
<b>fragmentPath</b>	Mandatory. String. The path of the file that contains the HTML fragment, relative to the main HTML file.
<b>parent</b>	Mandatory. Object. The parent element which contains the fragment.

Parameter	Description
<b>options</b>	<p>Mandatory. A hash object supporting the following option:</p> <ul style="list-style-type: none"> <li>• <code>onComplete</code>: Function. Callback function, called after the fragment is appended to the parent and all its JavaScript and CSS files are loaded.</li> <li>• <code>onUnload</code>: Function. Callback function, called when the fragment is unloaded. You are responsible to delete all objects that were created by the fragment. See The <code>onUnload</code> Callback for more details about this callback function.</li> </ul>

### Example

```
WL.Fragment.load("./list.html", document.getElementById("listContainer"), {onComplete: onListLoad});
```

### 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 116. *WL.Page.back* parameters

Parameter	Description
<b>options</b>	<p>A hash object supporting the options of the <code>WL.Page.load</code> method, and in addition:</p> <p><code>pagesBack</code>: 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.</p>

### 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 117. *WL.Page.load* parameters

Parameter	Description
<b>pagePath</b>	String. The path of the file that contains the HTML fragment that implement the page, relative to the main HTML file.
<b>options</b>	A hash object supporting the following option: <ul style="list-style-type: none"><li>• <b>onComplete</b>: Function. Callback function, called after the page is appended to the parent and all its JavaScript and CSS files are loaded.</li><li>• <b>onUnload</b>: Function. Callback function, called when the page is unloaded. You are responsible for deleting all objects that were created by the page. See the <b>onUnload</b> callback for more details about this callback function.</li></ul>

## 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.



Figure 68. 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 118. *WL.TabBar.addItem* parameters

Parameter	Description
<b>id</b>	Mandatory string. Identifies the tab.
<b>callback</b>	Mandatory function. The callback function that is invoked when the user touches the tab.
<b>title</b>	Mandatory string. The title of the tab. If null is passed, no title is displayed.
<b>options</b>	Options for customizing the tab item. <ul style="list-style-type: none"><li>• On iPhone:<ul style="list-style-type: none"><li>– 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.</li><li>– badge: String. A string to display in the optional circular badge on the item; if null or unspecified, no badge is displayed.</li></ul></li><li>• On Android:<ul style="list-style-type: none"><li>– image: String. File name or path relative to the application root directory, with a PNG image for the tab in unselected mode.</li><li>– ImageSelected: String. File name with an image for the tab in selected mode.</li></ul></li></ul>

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



## 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.

### 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 119. WL.TabBar.setEnabled parameters

Parameter	Description
<b>isEnabled</b>	Mandatory Boolean. <ul style="list-style-type: none"><li>• true: Enable the tab bar</li><li>• false: Disable the tab bar</li></ul>

### 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 Worklight Studio, the <body> element has this ID. Use this function to place the tab bar within an arbitrary element. This function is

useful when the location of the tab bar is not at the top of the application screen.

### Parameters

Table 120. *WL.TabBar.setParentDivId* parameters

Parameter	Description
<b>parentId</b>	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 121. *WL.TabBar.setSelectedItem* parameters

Parameter	Description
<b>id</b>	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.

## Parameters

Table 122. WL.TabBar.setVisible parameters

Parameter	Description
<b>isVisible</b>	Mandatory Boolean. <ul style="list-style-type: none"><li>• true: Shows the tab bar</li><li>• false: Hides the tab bar</li></ul>

## 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 123. WL.TabBarItem.setEnabled parameters

Parameter	Description
<b>isEnabled</b>	Mandatory Boolean. <ul style="list-style-type: none"><li>• true: Enable the tab bar item</li><li>• false: Disable the tab bar item</li></ul>

## Return Value

None

For iPhone:

```
var creditTab = WL.TabBar.addItem("CREDIT", selectCredit, "Visa", {image:"images/credit.png", badge:  
creditTab.setEnabled(false);
```

For Android:

```
var tabFeeds = WL.TabBar.addItem (  
'tab2',  
function(){worklightStarterApplication.selectTab('feedsWrapper');},
```

```
"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 124. WL.TabBarItem.updateBadge parameters

Parameter	Description
<b>badge</b>	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: "3"});
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-scale=1.0">
```
2. Update the Android CSS BODY tag to also apply to the HTML tag, as follows:

```
html, body {
height: auto;
overflow: auto;
}
```

## 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. For more information about these APIs, expand the entry for this topic in the **Contents** panel and see the *Overview* topic listed there.

## 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. For more information about these interfaces, expand the entry for this topic in the **Contents** panel and see the *Overview* topic listed there.

---

## 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 125. 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
Calling other procedures	WL.Server.invokeProcedure
Accessing an HttpServletRequest object	WL.Server.getClientRequest

Table 125. JavaScript API methods and objects (continued)

Function	Description
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

### Geo utility APIs

The Geo utility APIs, which can be used on both the client and the server, enable you to perform tasks such as identifying whether a coordinate lies inside or outside a geofence.

IBM Worklight provides the following Geo utility APIs:

- WL.Geo.getDistanceBetweenCoordinates
- WL.Geo.getDistanceToCircle
- WL.Geo.getDistanceToPolygon
- WL.Geo.isInsideCircle
- WL.Geo.isInsidePolygon
- WL.Geo.isOutsideCircle
- WL.Geo.isOutsidePolygon

### 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 126. JSON block properties

Property	Description
<b>method</b>	Mandatory. Defines the HTTP method. Valid values are get, post, put, and delete.



Table 126. JSON block properties (continued)

Property	Description
<b>returnedContentType</b>	<p>Optional. Defines the type of the content that is returned by the called HTTP service, overriding the HTTP response's mime type.</p> <p>If this parameter is not provided, the Worklight Server handles the response according to the mime type.</p> <p>If it is provided, the Worklight Server handles the response according to the indicated value. The field can receive the following values:</p> <ul style="list-style-type: none"> <li>css, csv, html, javascript, json, plain, and xml</li> <li>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).</li> </ul>
<b>returnedContentEncoding</b>	Optional. Defines the encoding of the returned content. Default is utf-8.
<b>path</b>	Mandatory. Defines the path of the URL defining the HTTP service.
<b>parameters</b>	Optional. Defines the set of parameters that need to be passed to the HTTP service.
<b>headers</b>	Optional. Defines the headers for the HTTP request.
<b>cookies</b>	Optional. Defines cookies to be passed with the HTTP request.
<b>body</b>	<p>Defines the content of the request body.</p> <ul style="list-style-type: none"> <li>When the method is GET, this property is not allowed.</li> <li>When the method is POST, this property is optional.</li> </ul> <p><b>Note:</b> <code>body.content</code> must be a string. If you are explicitly creating a DOM object, such as in: <code>var request = &lt;soap:Envelope ... &lt;/soap:Envelope&gt;;</code>, you must convert the object to string before you assign it to <code>body.content</code>, for example: <code>request.toString();</code></p>
<b>transformation</b>	Optional. If defined, the response of the service undergoes the defined XSL transformation. If the service returns HTML, the Worklight Server first converts the response to XHTML, and then runs the XSL transformation on the XHTML response.

**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 Worklight Server converts the HTML response to XHTML. If the service returns XML, the Worklight Server keeps it as is.
2. If an XSL transformation is defined in the `transformation` property, the 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 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
```

## 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.SOAPMessageFactory
```

Then, you must restart the JVM.

**Important:** This workaround is only for IBM WebSphere.

## Parameters

Table 127. `WL.Server.signSoapMessage` method parameters

Parameter	Description
<b>envelope</b>	Mandatory. The SOAP envelope that contains the fragment to sign.

Table 127. *WL.Server.signSoapMessage* method parameters (continued)

Parameter	Description
<b>wsId</b>	Mandatory. The value of the wsu:Id attribute, within the envelope, which identifies the fragment to be signed
<b>keystoreAlias</b>	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 properties listed in the following table. For information about how to specify IBM Worklight configuration properties, see “Configuration of IBM Worklight applications on the server” on page 714

Table 128. *Worklight Server Configuration* properties

Property	Description	Example
<b>ssl.keystore.path</b>	Path to the certificate that contains the key with which the envelope fragment must be encrypted	/conf/default.keystore
<b>ssl.keystore.type</b>	Type of the keystore	jks
<b>ssl.keystore.password</b>	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-200401-
...
</soapenv:Body>
</soapenv:Envelope>;
```

```
WL.Server.signSoapMessage(myEnvelope , "an-element-id", keystoreAlias);
```

## 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:

```
{
  procedure: 'procedure-name',
  parameters: [value-1, value-2, ...]
}
```

The JSON block contains the following properties:

Table 129. JSON block properties

Property	Description
<b>procedure</b>	Mandatory. This string defines the name of the stored procedure to call.
<b>parameters</b>	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.

## 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 130. `invokeSQLStatement` method parameter

Parameter	Description
<b>statement</b>	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 where
```

## 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 131. JSON block properties

Property	Description
<b>preparedStatement</b>	Mandatory. The prepared statement that was defined previously with <code>createSQLStatement</code> .
<b>parameters</b>	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 where

function procedure1(param) {
  return WL.Server.invokePreparedStatement({
    preparedStatement : procedure1Statement,
    parameters : [param]
  });
}
```

## 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.

IBM Worklight does not support reading from a topic in JMS adapters. The destination specified must be a queue.

## 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 132. JSON block properties

Property	Description
<b>destination</b>	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 <code>java:comp/env/...</code>
<b>timeout</b>	Optional. The time, in milliseconds, that the method will wait for a message, if a message is not immediately available. If <b>timeout</b> is not specified, the method will not wait for a message. <b>Special values for this parameter:</b> 0        wait indefinitely <0      do not wait
<b>filter</b>	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:

Table 133. Invocation results object properties

Property	Description
<b>isSuccessful</b>	<p>Identifies whether the method invocation succeeded or failed. Valid values are:</p> <p><b>true</b> The method invocation succeeded. This is the default value.</p> <p>Also set to true if there is no message on the destination and the method returns without error.</p> <p><b>false</b> The method invocation failed.</p>
<b>errors</b>	Optional. Any errors during processing will appear here.
<b>warnings</b>	Optional. Any warnings during processing will appear here. This includes warnings about any messages not of a supported JMS Message Type.
<b>message</b>	<p>Optional. The message, the message body, and the message properties are all optional.</p> <p><b>body</b> The message body. If no message is returned, this will not be available.</p> <p><b>properties</b> An array of message properties which follow the JMSMessage property rules. The following message properties can be returned:</p> <ul style="list-style-type: none"> <li>• JMS* properties. For example: JMSCorrelationID.</li> <li>• JMS_Provider_* properties. For example: JMS_IBM_Format.</li> <li>• user properties. For example: my_user_property.</li> </ul>

## 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.

IBM Worklight does not support reading from a topic in JMS adapters. The destination specified must be a queue.

## Parameters

The `readAllJMSMessages` function accepts the same set of parameters as `readSingleJMSMessage`.

## 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,
                   ....}
    }
  ]
}
```

## 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
}
```

The JSON block contains the following properties:

Table 134. JSON block properties

Property	Description
<b>destination</b>	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 <code>java:comp/env/...</code>
<b>message</b>	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.
<b>ttl</b>	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 135. Invocation results object properties

Property	Description
<b>isSuccessful</b>	Identifies whether the method invocation succeeded or failed. Valid values are:  <b>true</b> The method invocation succeeded. This is the default value.  <b>false</b> The method invocation failed.
<b>errors</b>	Optional. Any errors during processing will appear here.
<b>JMSMessageID</b>	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.

### 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 136. JSON block properties

Property	Description
<b>destination</b>	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 <code>java:comp/env/...</code>
<b>message</b>	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.
<b>timeout</b>	Optional. The time, in milliseconds, that the method waits for a message, if a message is not immediately available. If <b>timeout</b> is not specified, the method will not wait for a message.  <b>Special values for this parameter:</b>  0           wait indefinitely <0         do not wait
<b>ttl</b>	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`.

## 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 137. JSON block properties

Property	Description
<b>adapter</b>	Mandatory. A string that contains the name of the adapter as specified when the adapter was defined.
<b>procedure</b>	Mandatory. A string that contains the name of the procedure as specified when the adapter was defined.

Table 137. JSON block properties (continued)

Property	Description
<b>parameters</b>	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],
};
```

### Returned Value

```
{
  isSuccessful: Boolean,
  errorMessages: ["Error Msg1", ...],
  // Application object returned by procedure
}
```

The invocation results object contains the following properties:

Table 138. Invocation results object properties

Property	Description
<b>isSuccessful</b>	Optional. Identifies whether the procedure invocation succeeded or failed. Valid values are:  <b>true</b> The procedure invocation succeeded. This is the default value.  <b>false</b> The procedure invocation failed.
<b>errorMessages</b>	Optional. An array of strings that contain error messages. If no errors are provided, the returned array is empty.
<b>Application object</b>	Any object that is returned by the procedure.

### 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");
```

## WL.Server.getClientDeviceContext

Return a copy of the client device context.

## Syntax

```
WL.Server.getClientDeviceContext
```

## Description

The latest copy of the client device context is returned. See “`WL.Device.getContext`” on page 560. When this method is called in the context of an invoke procedure call, the copy is at least as recent as when that call was made. When this method is called in the context of an event handler, the copy is at least as recent as when the event was transmitted to the server.

This copy is synchronized transparently between the client and the server. Calling this method does not result in communications with the client.

## Parameters

None.

## Returned Value

The method returns a copy of the client device context, as would be obtained from the `WL.Device.getContext` API function.

## WL.Server.getActiveUser

Return an object that contains user identity properties.

## Syntax

```
WL.Server.getActiveUser ()
```

## 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"}
}
```

## 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.

## Parameters

Table 139. WL.Server.setActiveUser method parameters

Property	Description
<b>realm</b>	Mandatory. The realm to log in to (as defined in the authenticationConfig.xml file).



Table 139. *WL.Server.setActiveUser* method parameters (continued)

Property	Description
<b>identity</b>	<p>Mandatory. A user identity object, as returned by <code>WL.Server.getActiveUser</code>, with the following structure:</p> <p><b>userId</b> Mandatory. The login ID.</p> <p><b>displayName</b> Optional.</p> <p><b>credentials</b> Optional. A string with the user's credentials, such as password.</p> <p><b>attributes</b> Optional. Custom user attributes,</p>

## 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",
  }
})
```

## WL.Server.createEventHandler

Create an event handler.

## Syntax

```
WL.Server.createEventHandler(aFilter, aHandler)
```

## Description

An event handler is created. To set the event handler, to implement callbacks for received events, see “`WL.Server.setEventHandlers`” on page 637.

## Parameters

Table 140. *WL.Server.createEventHandler* method parameters

Property	Description
<b>aFilter</b>	Mandatory. An object that is used to filter incoming events. Events that match the filter are passed to <b>aHandler</b> .
<b>aHandler</b>	Mandatory. A function that is used to handle incoming events. Use a named function such as <code>function name(event)</code> in order for name to appear in reports and analytic output.

## Returned Value

An event handler is returned, which is defined as:

```
{
  filter: aFilter
  handler: aHandler
}
```

## WL.Server.setEventHandlers

Set event handlers, to implement callbacks for received events.

### Syntax

`WL.Server.setEventHandlers(eventHandlers)`

### Description

Event handlers are set, to implement callbacks for received events. To disable all event handlers, pass an empty array to the `WL.Server.setEventHandlers` method.

For information on creating an event handler, see “`WL.Server.createEventHandler`” on page 636.

### Parameters

Table 141. `WL.Server.setEventHandlers` method parameters

Property	Description
<code>eventHandlers</code>	Mandatory. An array, where each handler consists of an object that contains a filter and a handler function. Only events that match the filter are passed to the handler function. Each event handler has the format: <code>{filter: filterObject, handler: handlerFunction}</code> .

## Returned Value

None.

## 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 142. JSON block properties

Property	Description
<b>name</b>	Mandatory. A string that contains the name of the event source.
<b>onUserSubscribe</b>	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.
<b>onUserUnsubscribe</b>	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.
<b>onDeviceUnsubscribe</b>	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.
<b>onUserChange</b>	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 <b>isSuccessful</b> 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.
<b>poll</b>	Optional. If the method of getting the notification data from the back-end is polling, provide the following properties:  <b>interval</b> Mandatory. The interval in seconds between the polls.  <b>onPoll</b> Mandatory. The name of JavaScript function which is called on each poll.
<b>securityTest</b>	Mandatory. Declares the appropriate securityTest from authenticationConfig.xml to be used for this event source.

## Example

```
WL.Server.createEventSource({
  name: 'newCoupons',
  onUserSubscribe: 'subscribeUser',
  onUserUnsubscribe: 'unsubscribeUser',
  onUserChange: 'onUserChange',
  poll : {
    interval: 2,
    onPoll: 'produceNotifications'
  }
});
```

Related links: Security Tests

## 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 (deprecated in IBM Worklight V6.0.0), Windows Phone 8, and BlackBerry devices that support SMS functions

### Parameters

Table 143. *WL.Server.createDefaultNotification(notificationText, badge, payload) method parameters*

Property	Description
<b>notificationText</b>	Optional. The string that is displayed in the alert. On Windows Phone 8 the string is displayed in the application tile title.
<b>badge</b>	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.
<b>payload</b>	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,
  payload: payload
},
SMS: {
  text: notificationText
},
MPNS: {
  raw: {
    payload: payload
  },
  toast: null,
  tile: {
    title: notificationText,
    count: badge
  }
}

```

### Example

```
var notification = WL.Server.createDefaultNotification("You have " + numCoupons + " coupons.", numCoupons);
```

### 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 144. WL.Server.getUserNotificationSubscription method parameters

Parameter	Description
<b>eventSourceName</b>	Mandatory. A string that contains the name of the event source.
<b>userId</b>	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}}
```

## **userSubscription.getDeviceSubscriptions**

Return an array of device subscriptions.

## Syntax

```
userSubscription.getDeviceSubscriptions();
```

## 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}
  }
]
```

## **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.

## WL.Server.setApplicationContext

Set the application context, for auditing or reporting purposes.

### Syntax

```
WL.Server.setApplicationContext(applicationContext)
```

### Description

The application context that you set is saved in the raw data reports database. For more information, see “Using raw data reports” on page 861

### Parameters

Table 145. WL.Server.setApplicationContext method parameters

Property	Description
<code>applicationContext</code>	Mandatory. The application context to set. Call with null to clear.

### Returned Value

None.

## WL.Server.logActivity

Report user activity.

### Syntax

```
WL.Server.logActivity(activityType, deviceContext)
```

### 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 861.

**Note:** To ensure that the activity is stored in the database, set `reports.exportRawData` to true in the `worklight.properties` file. For information about how to specify IBM Worklight configuration properties, see “Configuration of IBM Worklight applications on the server” on page 714.

### Parameters

Property	Description
<code>activityType</code>	Mandatory. A string that identifies the activity.
<code>deviceContext</code>	Optional. A device context object that contains information about the device, such as its geo location. If a value is not provided, then <code>WL.Server.getClientDeviceContext</code> is used. You can use <code>deviceContext</code> when handling events with <code>event.deviceContext</code> .



## Returned Value

None.

## 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.

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 (deprecated in IBM Worklight V6.0.0), 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 649 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
  toast
    payload
    text1
    text2
    param
  tile
    id
    count
    title
    backgroundImage
    smallBackgroundImage
    wideBackgroundImage
    wideBackBackgroundImage
    wideBackContent
    backBackgroundImage
    backTitle
    backContent
    smallIconImage
    iconImage
    wideContent1
    wideContent2
    wideContent3
    backgroundColor
    cycleImage1
    cycleImage2
    cycleImage3
    cycleImage4
    cycleImage5
    cycleImage6
    cycleImage7
    cycleImage8
    cycleImage9

```

Table 146. *WL.Server.notifyAllDevices(userSubscription, notificationOptions)* method parameters

Name	Description
<b>userSubscription</b>	Mandatory. A user subscription object, obtained by calling <code>WL.Server.getUserNotificationSubscription</code>

Table 146. *WL.Server.notifyAllDevices(userSubscription, notificationOptions)* method parameters (continued)

Name	Description
<b>notificationOptions</b>	<p>Mandatory. The JSON block contains the following properties:</p> <p><b>APNS</b></p> <p><b>alert</b> Optional. A string to be displayed in the alert.</p> <p><b>badge</b> Mandatory. An integer value to be displayed in a badge on the application icon.</p> <p><b>sound</b> Optional. The name of a file to play when the notification arrives.</p> <p><b>actionKey</b> Optional. The label of the dialog box button that allows the user to open the app upon receiving the notification.</p> <p><b>payload</b> 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.</p> <p><b>GCM</b></p> <p><b>alert</b> Optional. A string to be displayed in the alert.</p> <p><b>sound</b> Optional. The name of a file to play when the notification arrives.</p> <p><b>payload</b> 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.</p> <p><b>delayWhileIdle</b> 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.</p> <p><b>timeToLive</b> 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.</p> <p><b>SMS</b></p> <p><b>text</b> Mandatory. A string to be displayed in the alert.</p> <p><b>MPNS</b></p> <p><b>raw</b></p>
	<p>Chapter 7. API reference <b>645</b></p> <p><b>payload</b> Optional. A JSON block that is transferred to the</p>

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 146 on page 644.
- 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 146 on page 644.

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 “`WL.Server.createDefaultNotification`” on page 639 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.

**Return value**

This method returns a value of type `long` that is a number between 0 and any number above 0. This value is a timeout that indicates, in milliseconds, the time you must wait before you submit a notification again.

**Example**

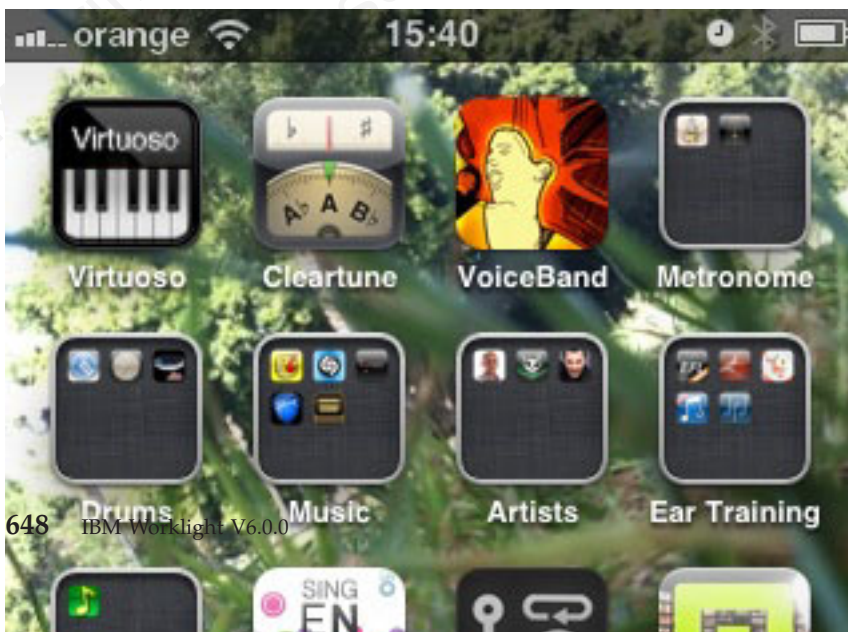
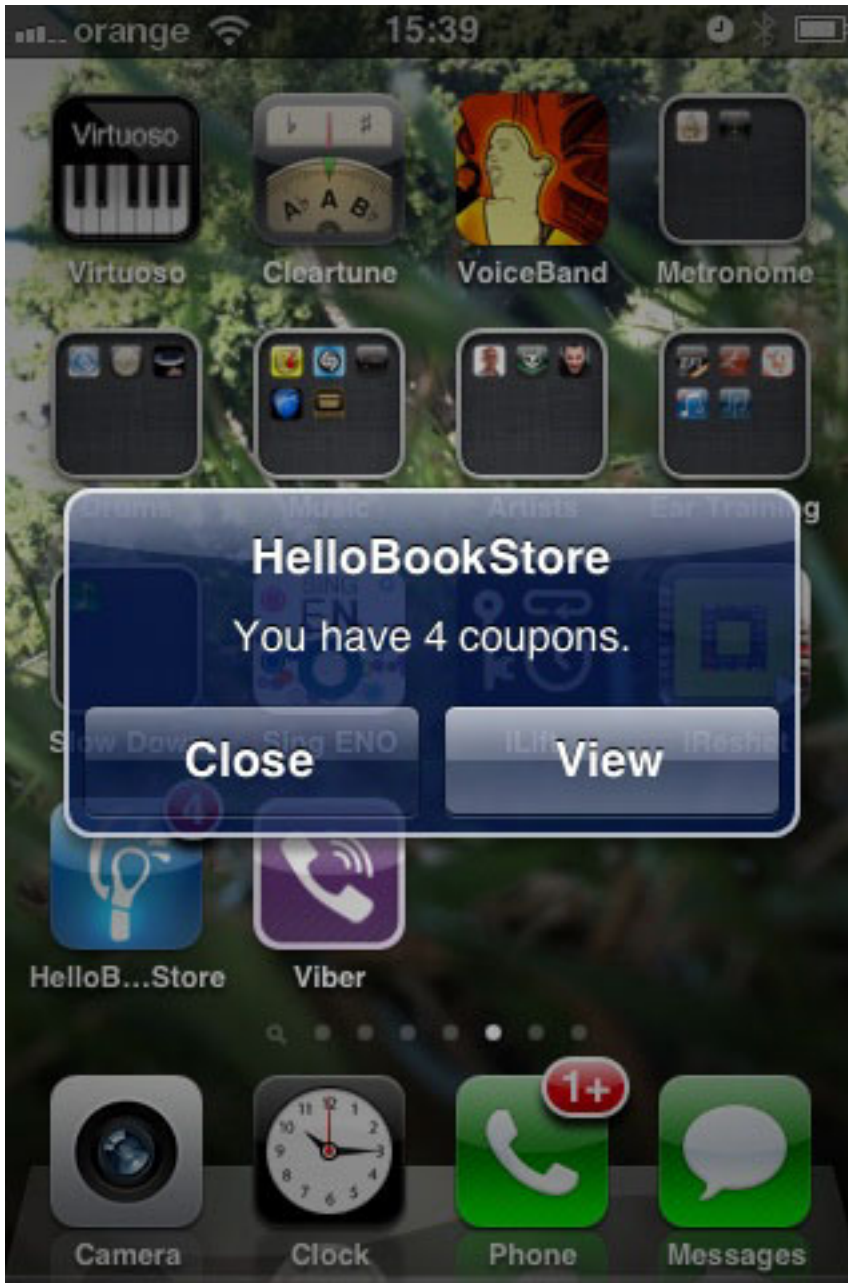
```
userSubscription = WL.Server.getUserNotificationSubscription ("MyEventSource", userID);  
  
var notification = WL.Server.createDefaultNotification("You have " + numCoupons + " coupons.", numCoupons);  
  
// change the sound for APNS  
notification.APNS.sound = mySound;
```

```
// 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";
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/00e45896-68c6-440";

var delayTimeout = 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





## 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 147. *WL.Server.notifyAllDevices(userSubscription, notificationOptions) method parameters*

Name	Description
<b>userSubscription</b>	Mandatory. A user subscription object, obtained by calling <code>WL.Server.getUserNotificationSubscription</code>
<b>notificationOptions</b>	Mandatory. The JSON block contains the following properties:  <b>badge</b> Mandatory. An integer value to be displayed in a badge on the application icon.  <b>sound</b> Optional. The name of a file to play when the notification arrives.  <b>alert</b> Optional. A string to be displayed in the alert.  <b>activateButtonLabel</b> Optional. The label of the dialog box button that allows the user to open the app upon receiving the notification.  <b>payload</b> 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.

### **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.



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 (deprecated in IBM Worklight V6.0.0), 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 148. WL.Server.notifyDevice method parameters

Name	Description
<b>userSubscription</b>	Mandatory. A user subscription object, obtained by calling WL.Server.getUserNotificationSubscription
<b>device</b>	Mandatory. The device ID that is used to identify the device by the Worklight Server.
<b>options</b>	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 "WL.Server.createDefaultNotification" on page 639 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.

## Return value

This method returns a value of type long that is a number between 0 and any number above 0. This value is a timeout that indicates, in milliseconds, the time you must wait before you submit a notification again.

## Example

```
userSubscription = WL.Server.getUserNotificationSubscription ("MyEventSource", userID);  
  
var notification = WL.Server.createDefaultNotification("You have " + numCoupons + " coupons.", numCoupons);  
  
var delayTimeout = WL.Server.notifyDevice( userSubscription, userSubscription.getDeviceSubscriptions());
```

## WL.Server.notifyDeviceSubscription

Submit a notification to the specified device of a subscribed user.

### Syntax

```
WL.Server.notifyDeviceSubscription (deviceSubscription, notificationOptions)
```

### Description

This method replaces the deprecated method `WL.Server.submitNotification`.

You can submit 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 (deprecated in IBM Worklight V6.0.0), 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 149. WL.Server.notifyDeviceSubscription method parameters

Name	Description
<b>deviceSubscription</b>	Mandatory. The device subscription, obtained by calling <code>getDeviceSubscriptions</code> on the user subscription object.
<b>notificationOptions</b>	Mandatory. See Method <code>WL.Server.notifyAllDevices</code> .

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 “`WL.Server.createDefaultNotification`” on page 639 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.

### Return value

This method returns a value of type `long` that is a number between 0 and any number above 0. This value is a timeout that indicates, in milliseconds, the time you must wait before you submit a notification again.

## Example

```
userSubscription = WL.Server.getUserNotificationSubscription ("MyEventSource", userID);

var notification = WL.Server.createDefaultNotification("You have " + numCoupons + " coupons.", numCoupons);

var delayTimeout = WL.Server.notifyDeviceSubscription(
    userSubscription.getDeviceSubscriptions()[0], notification
);
```

## WL.Server.submitNotification (Deprecated)

This method is deprecated.

### 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 150. *Deprecated Method WL.Server.submitNotification parameters*

Name	Description
<b>deviceSubscription</b>	Mandatory. The device subscription obtained by calling <code>getDeviceSubscriptions</code> on the user subscription object.
<b>notificationOptions</b>	Mandatory. See Method <code>WL.Server.notifyAllDevices</code> .

## WL.Logger.debug, info, warn, and error

Logger methods for server-side code.

### Syntax

- `WL.Logger.debug (message)`
- `WL.Logger.info (message)`
- `WL.Logger.warn (message)`
- `WL.Logger.error (message)`

### Description

The following methods are logger methods for server-side code. Since IBM Worklight 6.0.0, the development server is Liberty Profile. Liberty Profile does not support messages of the DEBUG level in the console log. The default log level is AUDIT. You can configure the log level in the logging element of the `server.xml` configuration file. Messages of the INFO, WARNING, and ERROR level are printed to the messages log file. All log messages, including the messages of DEBUG level, can be traced by using the Liberty Profile trace mechanism.

**Note:** The `WL.Logger.log` method is deprecated in IBM Worklight 6.0.0. You must use the `WL.Logger.info` method instead.

**Note:** If you are using a standalone 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 833.

## Parameters

Table 151. Methods `WL.Logger.debug`, `info`, `warn`, and `error` parameters

Name	Description
<code>message</code>	Mandatory. A string that contains the message to be written to the log file.

## 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 three interfaces.

For more information about these interfaces, expand the entry for this topic in the **Contents** panel and see the *Overview* topic listed there.

---

## Internal IBM Worklight database tables

You can access a database of common tables from the 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
<code>ADAPTER_SYNC_DATA</code>	Stores the adapter deployable elements. This table is used to synchronize the adapter deployable elements between cluster nodes.	10s of rows.

Name	Description	Order of Magnitude
APP_SYNC_DATA	Stores the application deployable elements. This table is used to synchronize the application deployable elements between cluster nodes.	10s of rows.
APP_VERSION_ACCESS_DATA	Stores the applications that have the remote disable mode to block or notify.	10s of rows.
CLUSTER_SYNC	Internal cluster synchronization tasks.	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. No user preferences are ready for immediate use. The App developer can add preferences.	If used, this table can contain 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 1 relationships with NOTIFICATION_USER table.	1 row per device subscribed 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 subscribed to an event.
SSO_LOGIN_CONTEXTS	Stores the active sessions that use the SSO feature.	Depends if SSO is enabled. If enabled, there is one entry per session.
WORKLIGHT_VERSION	The Worklight version.	1 row.

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 the last 30 days for faster access to the last 30 days of activities.

Name	Description	Order of Magnitude
FACT_ACTIVITIES	<p>Summarization of activities that are used for device analytics.</p> <p>Updated by Worklight Server every 24 hours with data from the APP_ACTIVITY_REPORT table. Primarily used by BIRT reports and by other reporting tools. The update interval can be configured with the <code>wl.db.factProcessingInterval</code> property. For more information about the property, see "Device usage reports" on page 865.</p>	Size depends on app/device usage.
NOTIFICATION_ACTIVITIES	<p>Summarization of activities that are used for notification analytics.</p> <p>Updated with data from the NOTIFICATION_REPORT table. Primarily used by BIRT reports and by other reporting tools.</p>	Size depends on app/notification usage.
PROC_REPORT	Internal table that is used for housekeeping and maintaining the state of the scheduler tasks.	About 72 rows per day.
APP_ACTIVITY_REPORT	<p>The reports row data. Data is aggregated by either our aggregation task or by the customer aggregation task.</p> <p>For more information about using the row data, see "Using raw data reports" on page 861.</p>	The size depends on application. The customer is responsible for purging older entries after aggregating to Data Warehouse.

---

## 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}/{Acti
```

#### Application API request headers

Header Name	Data Type	Description	Valid values
<b>x-wl-app-version</b>	String	Version of the application	
<b>WL-Instance-ID</b>	String	Protection mechanism for XSS attacks.	

#### Application API request elements

Header Name	Data Type	Description	Valid values
<b>Protocol</b>	String		HTTP
<b>Worklight Server</b>	String	Host name or IP address (and possibly port) identifying the IBM Worklight Server	
<b>Application ID</b>	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
<b>Application Environment</b>	String	Name of the environment the <b>application</b> is running on	air, android, Androidnative, blackberry, desktopbrowser, iOSnative, ipad, iphone, JavaMEnative, mobilewebapp, windows8, windowsphone
<b>Action</b>	String	Requested action	Details in following table

#### Actions

Action	HTTP Request	Parameters
<b>init</b>	POST	<b>x</b> , <b>isAjaxRequest</b> – see the following table showing common parameters.
<b>heartbeat</b>	POST	<b>x</b> , <b>isAjaxRequest</b> – see the following table showing common parameters.



Actions

Action	HTTP Request	Parameters
<b>logactivity</b>	POST	<b>x, isAjaxRequest</b> – see the following table showing common parameters.  <b>activity</b> – string.
<b>query</b>	POST	<b>x, isAjaxRequest</b> – see the following table showing common parameters. <b>filterList</b> – JSON block  <b>parameterList</b> – JSON block  <b>sorterList</b> – JSON block <b>Note:</b> When the action is <b>query</b> , the request URL has the following structure: .../query/{Adapter Name}/{Procedure Name} where <b>Adapter Name</b> and <b>Procedure Name</b> are strings.
<b>logout</b>	POST	<b>x, isAjaxRequest</b> - see the following table showing common parameters.
<b>login</b>	POST	<b>x, isAjaxRequest</b> – see the following table showing common parameters.  <b>realm</b> – string.
<b>updates</b>	POST	<b>x, isAjaxRequest</b> – see the following table showing common parameters.  <b>skin</b> – current skin name (string)  <b>checksum</b> – the checksum of the current skin (string)  <b>skinLoaderChecksum</b> – the checksum of the skin selection code (string)
<b>getup</b>	POST	<b>x, isAjaxRequest</b> - see the following table showing common parameters.
<b>deleteup</b>	POST	<b>x, isAjaxRequest</b> – see the following table showing common parameters.  <b>userprefkey</b> – the user preference to delete.
<b>getuserinfo</b>	POST	<b>x, isAjaxRequest</b> – see the following table showing common parameters.

Actions

Action	HTTP Request	Parameters
<b>getgadgetprefs</b>	POST	<b>x, isAjaxRequest</b> - see the following table showing common parameters.
<b>notifications</b>	POST	<p><b>x, isAjaxRequest</b> - see the following table showing common parameters.</p> <p><b>subscribe</b> - JSON string containing subscribe options</p> <p><b>unsubscribe</b> - when specified, designates an unsubscribe action</p> <p><b>updateToken</b> - the update notification token (string)</p> <p><b>adapter</b> - the name of the notification adapter (string)</p> <p><b>eventSource</b> - the name of the notification event source (string)</p> <p><b>alias</b> - notification subscription alias (string)</p>
<b>fbcallback</b>	GET or POST	<p><b>x, isAjaxRequest</b> - see the following table showing common parameters.</p> <p><b>popup</b> - string</p>
<b>composite</b>	POST	<p><b>x, isAjaxRequest</b> - see the following table showing common parameters.</p> <p><b>requests</b> - a JSON string containing information about other actions to invoke.</p> <p>This action is used to combine several actions in a single HTTP request.</p>
<b>appversionaccess</b>	GET	<b>x, isAjaxRequest</b> - see the following table showing common parameters.
<b>setup</b>	POST	<p><b>x, isAjaxRequest</b> - see the following table showing common parameters.</p> <p><b>userprefs</b> contains JSON pairs of preference key and value</p>

## Actions

Action	HTTP Request	Parameters
<b>authentication</b>	POST	<b>x, isAjaxRequest</b> - see the following table showing common parameters.  <b>action</b> values are popup, test, or test_img
<b>authenticate</b>	POST	<b>x, isAjaxRequest</b> - 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
<b>isAjaxRequest</b>	true	Included with every <b>GET</b> and <b>POST</b> request only from Adobe™ AIR application.
-	None	Included with every <b>POST</b> 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}/{App
```

### 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
<b>Application Resource Path</b>	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}/{Ap

### 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
<b>Application Resource Path</b>	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	<b>delete</b>	POST	<b>adapterName</b>
	<b>get</b>	GET	<b>adapterName</b>
	<b>all</b>	GET	None
	<b>upload</b>	POST	<b>adapterName, input</b>
Applications	<b>getPublishUrl</b>	GET	<b>gadgetAppId</b>
	<b>parseCSV</b>	POST	<b>CSV file</b>
	<b>delete</b>	POST	<b>applicationName</b>
	<b>deleteGadgetApplication</b>	POST	<b>gadgetAppId</b>
	<b>setAccessRule</b>	POST	<b>gadgetAppId</b> (mandatory), <b>action</b> (mandatory: delete, block, or notify), <b>message</b> (mandatory), <b>downloadLink</b> (optional)
	<b>setAuthenticityRule</b>	POST	<b>gadgetAppId</b> (mandatory), <b>action</b> (mandatory: disabled, ignored, or enabled)
	<b>setVersionLock</b>	POST	<b>gadgetAppId, lock</b> (true or false)
	<b>getBinaryApp</b>	GET	<b>gadgetAppId</b>
	<b>all</b>	GET	None
	<b>get</b>	GET	<b>applicationName</b>
<b>upload</b>	POST	<b>input, applicationFolderPath</b>	
Push	<b>unsubscribeSMS</b>	POST	<b>phoneNumbers</b>
Push, Applications	<b>all</b>	GET	None

## Actions

API Context	Action	HTTP Request	Parameters
Push, Mediators	<b>all</b>	GET	None
	<b>get</b>	GET	gcm, apns, or mpns
Push, Event Sources	<b>all</b>	GET	None
	<b>get</b>	GET	<b>adapterName/eventSourceName</b>
Users	<b>userName</b>	GET	None
	<b>logout</b>	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.

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 IBM Worklight projects

After you have created projects and apps with Worklight Studio, you must deploy them to the production environment.

You can deploy several Worklight projects (that is, several project WAR files) to an application server just as you would deploy any JEE application: each deployed project must have a unique name and a unique context path. You can choose between having several projects use the same database server, or making each project use a different database server. If you configure several projects to use the same database, you must configure each data source to connect to an independent data storage structure (for example, different schemas on DB2, or different user names on Oracle). Database sharing is not relevant for MySQL and Apache Derby.

Several Worklight Servers with different versions of Worklight installed can share the same application server. For example, different Worklight project WAR files that use different versions of `worklight-jee-library.jar` can coexist on the same application server.

Read this series of topics to learn how to deploy your IBM Worklight projects and apps to the production environment.

---

### Deploying IBM Worklight applications to test and production environments

When you have developed an application, deploy it to a separate test and production environment.

#### About this task

When you finish a development cycle of your application, you usually deploy it to a testing environment, and then to a production environment.

The tools that you can use to deploy apps and adapters across development, QA, and production environments are described in the following topics.

#### Deploying an application from development to a test or production environment

This section describes the steps to move from a development environment and deploy a Worklight project to a test or production environment.

#### Before you begin

You have built a Worklight project containing one or more applications in IBM Worklight Studio. A WAR file and a set of `.wlap` files are created in the `bin` folder of your IBM Worklight project. You now want to deploy the project and the applications to a test or production environment.

- A WAR file is created by Worklight Studio for every Worklight project, regardless of the number of apps it contains.



- 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-all.wlapp*.
- 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.
  - Settings screen
  - Device provisioning
  - Application authenticity
  - User authentication
  - The Android shared user ID

For more information, see “The application descriptor” on page 255

2. You might want to look at the settings in the *worklight.properties* file. Those settings define the default values for the IBM Worklight configuration properties on the server. When you deploy your Worklight project on the server, you can replace the default settings that are in the *worklight.properties* file with values that are relevant for the production environment. For more information, see “Configuring an IBM Worklight project in production by using JNDI environment entries” on page 723.
3. Build each application in either of two ways:
  - Right-click the application and click **Run As > Build For Remote Server**.
  - Use the Ant script tool that is described in “Ant tasks for building and deploying applications and adapters” on page 730

If you use Worklight Studio, the project WAR file is named *projectName.war* and is located 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. Create and configure a database for your server following the instructions in “Creating and configuring the databases” on page 666.
5. Deploy the project WAR file to the remote server, as described in “Deploying a project WAR file and configuring the application server” on page 694. In this step, you can also configure the project on the server using JNDI env entries. See details in “Configuring an IBM Worklight project in production by using JNDI environment entries” on page 723.
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 Worklight Console, deploy the relevant *.wlapp* files from the *bin* folder of your Worklight project.

- For more information about how to deploy an app by using IBM Worklight Console, see “Deploying apps” on page 736.
  - 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 731.
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 737.
    - 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 732.

## Building a project WAR file with Ant

Build the project WAR file with Ant tasks as detailed here.

The following section documents an example of the Ant XML file used to build an IBM Worklight project WAR file.

### Building the project WAR file

The Ant task for building the project WAR file has the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="myProject" default="all">
  <taskdef resource="com/worklight/ant/defaults.properties">
    <classpath>
      <pathelement location="WL_INSTALL_DIR/WorklightServer/worklight-ant.jar"/>
    </classpath>
  </taskdef>
  <target name="all">
    <war-builder projectfolder="."
      destinationfolder="bin/war"
      warfile="bin/project.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

## Deploying an IBM Worklight project

Deploy an IBM Worklight project by following the steps as detailed here.

## Ant tasks for deploying a project WAR file and configuring an application server

A set of Ant tasks is supplied with Worklight Server. The tasks in this section are used to deploy a project WAR file, and configure your databases and application servers.

Apache Ant is required to run these tasks. The minimum supported version of Ant is listed in System Requirements for IBM Worklight and IBM Mobile Foundation.

### Creating and configuring the databases

Create and configure the databases by following the steps detailed here.

#### Optional creation of databases before you use the Ant tasks:

If you plan to use the Ant tasks to create and configure your databases, you must have the proper database access rights to run the Ant scripts.

If you want to create the Worklight databases with the Ant tasks described in the following topic, you must have certain database access rights that entitle you to create the tables that are required by IBM Worklight. If you have sufficient database administration credentials, and if you enter the administrator user name and password in the Ant file, the Ant tasks can create the databases for you.

If you do not have these permissions, you must ask your database administrator to create the required databases for you. The databases must be created before you run the Ant tasks to configure the databases.

The following topics describe the procedures a database administrator uses to create each of the supported databases.

#### *Creating the DB2 databases:*

This section explains the procedures used to create the DB2 databases.

#### About this task

The <configureDatabase> Ant task 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.

You can replace the database names (here WRKLGHT and WLREPORT) and passwords with database names and passwords of your choosing.

**Important:** You can name your databases and user differently, or set a different password, but ensure that you enter the appropriate database names, user name, and password correctly across the DB2 database setup. DB2 has a database name limit of 8 characters on all platforms, and has a user name and password length limit of 8 characters for Unix and Linux systems, and 30 characters for Windows.

You can also choose to have the IBM Worklight data and the Worklight reports data be stored in a single database, as different schemas. To this effect, in the procedure below, use a single database name of your choosing instead of WRKLGHT and WLREPORT.

## Procedure

1. Create a system user, for example, named `wluser` in a DB2 admin group such as `DB2USERS`, using the appropriate commands for your operating system. Give it a password, for example, `wluser`. If you want multiple Worklight Servers to connect to the same database, use a different user name for each connection. Each database user has a separate default schema. For more information about database users, 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 `~/sql1lib/bin` and enter `./db2`.
  - Enter database manager and SQL statements similar to the following example to create the two databases, replacing the user name `wluser` with your chosen user names:

```
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
QUIT
```
3. It is also possible to use only one database (with pagesize settings compatible with what is listed above), and to create the databases for Worklight in different schemas. In that case, only one database is required. If the `IMPLICIT_SCHEMA` authority is granted to the user created in step 1 (the default in the database creation script in step 2), no further action is required. If the user does not have the `IMPLICIT_SCHEMA` authority, you need to create a `SCHEMA` for the Worklight database tables and objects and a `SCHEMA` for the Worklight Report database tables and objects.

### *Creating the MySQL databases:*

This section explains the procedures used to create the MySQL databases.

### About this task

The `<configureDatabase>` Ant task 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 databases, you can replace the database names (here `WRKLGHT` and `WLREPORT`) and the password with database names and a password of your choosing. Note that MySQL database names are case-sensitive on Unix.

## 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 'password';
GRANT ALL privileges ON WRKLGHT.* TO 'worklight'@'localhost' IDENTIFIED BY 'password';
Flush privileges;
```

```

CREATE DATABASE WLREPORT CHARACTER SET utf8 COLLATE utf8_general_ci;
GRANT ALL privileges ON WLREPORT.* TO 'worklight'@'Worklight-host' IDENTIFIED BY 'password';
GRANT ALL privileges ON WLREPORT.* TO 'worklight'@'localhost' IDENTIFIED BY 'password';
Flush privileges;

```

3. Replace *Worklight-host* with the name of the host on which IBM Worklight runs.

*Creating the Oracle databases:*

This section explains the procedures used to create the Oracle databases.

### About this task

The <configureDatabase> Ant task can create the databases or users and schemas inside an existing database 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 or users and schemas for you. When you manually create the databases or users, you can use database names, user names, and a password of your choosing. Note that lowercase characters in Oracle user names can lead to trouble.

### Procedure

1. If you do not already have a database named ORCL, use the Oracle Database Configuration Assistant (DBCA) and follow the steps in the wizard to create a new database named ORCL:
  - a. Use global database name *ORCL\_your\_domain*, and system identifier (SID) ORCL.
  - 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 procedure, accepting the default values.
2. Create database users either by using Oracle Database Control, or by using the Oracle SQLPlus command-line interpreter.
  - Using Oracle Database Control.
    - a. Create the user for the runtime database:
      - 1) Connect as SYSDBA.
      - 2) Go to the **Users** page: click **Server**, then **Users** in the **Security** section.
      - 3) Create a user, for example, named WORKLIGHT. If you want multiple IBM Worklight Servers to connect to the same general-purpose database you created in step 1, use a different user name for each connection. Each database user has a separate default schema.
      - 4) Assign 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**
- b. Repeat step "a" to create a user, for example, named WORKLIGHTREPORTS for the IBM Worklight report database.
- Using the Oracle SQLPlus command-line interpreter.

The commands in the following example create a user named worklight for all three databases:

```
CONNECT system/<system_password>@ORCL
CREATE USER WORKLIGHT IDENTIFIED BY password;
GRANT CONNECT, RESOURCE, CREATE VIEW TO WORKLIGHT;
DISCONNECT;
```

```
CONNECT system/<system_password>@ORCL
CREATE USER WORKLIGHTREPORTS IDENTIFIED BY password;
GRANT CONNECT, RESOURCE, CREATE VIEW TO WORKLIGHTREPORTS;
DISCONNECT;
```

### Creating and configuring the databases with Ant tasks:

Use Ant tasks to create and configure the IBM Worklight databases as detailed here.

An Ant task is supplied that ensures that a Worklight or WorklightReports database is present and that it is operational. The task:

- Creates the database, if it does not yet exist.
- Ensures that the database is accessible by the specified user, granting the required access rights if necessary.
- Ensures that the database has a schema with the given name. It creates the schema if necessary. (DB2 and Apache Derby only)
- Ensures that the database has the required tables. It creates the tables if the database or schema is empty, or upgrades the database contents if it finds tables from a previous version of Worklight.

To start the Ant task, you need an Ant XML file with one or more invocations of the <configuredatabase> task. If you want to start the Ant task from a computer on which Worklight Server is not installed, you must copy the file <WorklightInstallDir>/WorklightServer/worklight-ant.jar to that computer.

Sample Ant XML files are presented in the following sections.

### Prerequisite steps

Before doing these steps, it is necessary that:

- Worklight Server is installed.
- Apache Ant is installed.
- If you plan to use a database management system other than Apache Derby, the database management system must be installed on some database server (possibly the same machine, possibly a different machine), and that database server is running.
- If you do not have database administrator permissions on the database management system, the steps in section "Optional creation of databases before you use the Ant tasks" on page 666 must be completed.



## To create Apache Derby databases

**Note:** The Apache Derby database is provided in the Worklight Server distribution, but is not suggested for use in production environments. In these environments, an IBM DB2, MySQL, or Oracle database is more common and more appropriate. The following Ant XML examples are provided in case you want to install the Apache Derby database in a test environment.

If you want to have two different databases, for example to implement the situation that is described in “Setting up your Apache Derby databases manually” on page 682, the configuration looks similar to the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<project basedir="." default="all">
  <taskdef resource="com/worklight/ant/defaults.properties">
    <classpath>
      <pathelement location="/opt/IBM/Worklight/WorklightServer/worklight-ant.jar"/>
    </classpath>
  </taskdef>

  <target name="all">

    <configuredatabase kind="Worklight">
      <derby database="WRKLGHT" datadir="/var/ibm/Worklight/derby"/>
    </configuredatabase>

    <configuredatabase kind="WorklightReports">
      <derby database="WLREPORT" datadir="/var/ibm/Worklight/derby"/>
    </configuredatabase>

  </target>
</project>
```

If you want to have a single database with different schemas:

```
<?xml version="1.0" encoding="UTF-8"?>
<project basedir="." default="all">
  <taskdef resource="com/worklight/ant/defaults.properties">
    <classpath>
      <pathelement location="/opt/IBM/Worklight/WorklightServer/worklight-ant.jar"/>
    </classpath>
  </taskdef>

  <target name="all">

    <configuredatabase kind="Worklight">
      <derby database="WorklightProduction" datadir="/var/databases/derby" schema="WL60"/>
    </configuredatabase>

    <configuredatabase kind="WorklightReports">
      <derby database="WorklightProduction" datadir="/var/databases/derby" schema="WL60REP"/>
    </configuredatabase>

  </target>
</project>
```

## To create DB2 databases

If you want to have two different databases, for example to implement the situation that is described in “Setting up your DB2 databases manually” on page 678, the configuration can look like the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<project basedir="." default="all">
  <taskdef resource="com/worklight/ant/defaults.properties">
```



```

<classpath>
  <pathelement location="/opt/IBM/Worklight/WorklightServer/worklight-ant.jar"/>
</classpath>
</taskdef>

<target name="all">

  <configuredatabase kind="Worklight">
    <db2 database="WRKLGHT" server="proddb.example.com"
      user="wl6admin" password="wl6pass">
      <dba user="db2inst1" password="db2IsFun"/>
    </db2>
    <driverclasspath>
      <fileset dir="/opt/database-drivers/db2-9.7">
        <include name="db2jcc4.jar"/>
        <include name="db2jcc_license_*.jar"/>
      </fileset>
    </driverclasspath>
  </configuredatabase>

  <configuredatabase kind="WorklightReports">
    <db2 database="WLREPORT" server="proddb.example.com"
      user="wl6admin" password="wl6pass">
      <dba user="db2inst1" password="db2IsFun"/>
    </db2>
    <driverclasspath>
      <fileset dir="/opt/database-drivers/db2-9.7">
        <include name="db2jcc4.jar"/>
        <include name="db2jcc_license_*.jar"/>
      </fileset>
    </driverclasspath>
  </configuredatabase>

</target>
</project>

```

If you want to have a single database with two schemas, each owned by a different user:

```

<?xml version="1.0" encoding="UTF-8"?>
<project basedir="." default="all">
  <taskdef resource="com/worklight/ant/defaults.properties">
    <classpath>
      <pathelement location="/opt/IBM/Worklight/WorklightServer/worklight-ant.jar"/>
    </classpath>
  </taskdef>

  <target name="all">

    <configuredatabase kind="Worklight">
      <db2 database="PROD" server="proddb.example.com"
        user="wladmin" password="wl6pass">
        <dba user="db2inst1" password="db2IsFun"/>
      </db2>
      <driverclasspath>
        <fileset dir="/opt/database-drivers/db2-9.7">
          <include name="db2jcc4.jar"/>
          <include name="db2jcc_license_*.jar"/>
        </fileset>
      </driverclasspath>
    </configuredatabase>

    <configuredatabase kind="WorklightReports">
      <db2 database="PROD" server="proddb.example.com"
        user="wlreport" password="wl6pass">
        <dba user="db2inst1" password="db2IsFun"/>
      </db2>
    </configuredatabase>
  </target>
</project>

```

```

        <driverclasspath>
        <fileset dir="/opt/database-drivers/db2-9.7">
        <include name="db2jcc4.jar"/>
        <include name="db2jcc_license_*.jar"/>
        </fileset>
        </driverclasspath>
    </configuredatabase>

</target>
</project>

```

To create a single database with two schemas, which are owned by the same user:

```

<?xml version="1.0" encoding="UTF-8"?>
<project basedir="." default="all">
    <taskdef resource="com/worklight/ant/defaults.properties">
        <classpath>
            <pathelement location="/opt/IBM/Worklight/WorklightServer/worklight-ant.jar"/>
        </classpath>
    </taskdef>

    <target name="all">

        <configuredatabase kind="Worklight">
            <db2 database="PROD" server="proddb.example.com"
                user="wl6admin" password="wl6pass" schema="WL60">
                <dba user="db2inst1" password="db2IsFun"/>
            </db2>
            <driverclasspath>
                <fileset dir="/opt/database-drivers/db2-9.7">
                <include name="db2jcc4.jar"/>
                <include name="db2jcc_license_*.jar"/>
                </fileset>
            </driverclasspath>
        </configuredatabase>

        <configuredatabase kind="WorklightReports">
            <db2 database="PROD" server="proddb.example.com"
                user="wl6admin" password="wl6pass" schema="WL60REP">
                <dba user="db2inst1" password="db2IsFun"/>
            </db2>
            <driverclasspath>
                <fileset dir="/opt/database-drivers/db2-9.7">
                <include name="db2jcc4.jar"/>
                <include name="db2jcc_license_*.jar"/>
                </fileset>
            </driverclasspath>
        </configuredatabase>

    </target>
</project>

```

### To create MySQL databases

With MySQL, you must have two different databases. The configuration looks similar to the following example:

```

<?xml version="1.0" encoding="UTF-8"?>
<project basedir="." default="all">
    <taskdef resource="com/worklight/ant/defaults.properties">
        <classpath>
            <pathelement location="/opt/IBM/Worklight/WorklightServer/worklight-ant.jar"/>
        </classpath>
    </taskdef>

    <target name="all">

```

```

<configuredatabase kind="Worklight">
  <mysql database="WRKLGHT" server="proddb.example.com"
    user="wl6admin" password="wl6pass">
    <dba user="root" password="UnGuessable"/>
    <client hostname="prodserver.example.com"/>
  </mysql>
  <driverclasspath>
    <pathelement location="/opt/database-drivers/mysql/mysql-connector-java-5.1.25-bin.jar"/>
  </driverclasspath>
</configuredatabase>

<configuredatabase kind="WorklightReports">
  <mysql database="WLREPORT" server="proddb.example.com"
    user="wl6admin" password="wl6pass">
    <dba user="root" password="UnGuessable"/>
    <client hostname="prodserver.example.com"/>
  </mysql>
  <driverclasspath>
    <pathelement location="/opt/database-drivers/mysql/mysql-connector-java-5.1.25-bin.jar"/>
  </driverclasspath>
</configuredatabase>

</target>
</project>

```

### To create Oracle databases

You may want to have two different users and schemas in the same database, for example to implement the situation that is described in “Setting up your Oracle databases manually” on page 690. In this case, the configuration can look like the following example:

```

<?xml version="1.0" encoding="UTF-8"?>
<project basedir="." default="all">
  <taskdef resource="com/worklight/ant/defaults.properties">
    <classpath>
      <pathelement location="/opt/IBM/Worklight/WorklightServer/worklight-ant.jar"/>
    </classpath>
  </taskdef>

  <target name="all">

    <configuredatabase kind="Worklight">
      <oracle database="ORCL" server="proddb.example.com"
        user="WL60MAIN" password="wl6pass"
        SYSTEMPassword="Passw0rd">
        <dba user="oracle" password="delphi"/>
      </oracle>
      <driverclasspath>
        <pathelement location="/opt/databases-drivers/oracle-11.1/ojdbc6.jar"/>
      </driverclasspath>
    </configuredatabase>

    <configuredatabase kind="WorklightReports">
      <oracle database="ORCL" server="proddb.example.com"
        user="WL60REPT" password="wl6pass"
        systemPassword="Passw0rd">
        <dba user="oracle" password="delphi"/>
      </oracle>
      <driverclasspath>
        <pathelement location="/opt/databases-drivers/oracle-11.1/ojdbc6.jar"/>
      </driverclasspath>
    </configuredatabase>

  </target>
</project>

```

If you want to have two different databases (the way that IBM Worklight 5.x created the databases by default), the configuration can look like the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<project basedir="." default="all">
  <taskdef resource="com/worklight/ant/defaults.properties">
    <classpath>
      <pathelement location="/opt/IBM/Worklight/WorklightServer/worklight-ant.jar"/>
    </classpath>
  </taskdef>

  <target name="all">

    <configuredatabase kind="Worklight">
      <oracle database="WRKLGHT" server="proddb.example.com"
        user="SCOTT" password="tiger"
        SYSPassword="Passw0rd" SYSTEMPassword="Passw0rd">
        <dba user="oracle" password="delphi"/>
      </oracle>
      <driverclasspath>
        <pathelement location="/opt/databases-drivers/oracle-11.1/ojdbc6.jar"/>
      </driverclasspath>
    </configuredatabase>

    <configuredatabase kind="WorklightReports">
      <oracle database="WLREPORT" server="proddb.example.com"
        user="SCOTT" password="tiger"
        sysPassword="Passw0rd" systemPassword="Passw0rd">
        <dba user="oracle" password="delphi"/>
      </oracle>
      <driverclasspath>
        <pathelement location="/opt/databases-drivers/oracle-11.1/ojdbc6.jar"/>
      </driverclasspath>
    </configuredatabase>

  </target>
</project>
```

Ant **configuredatabase** task reference:

Reference information for the Ant <configuredatabase> task.

The <configuredatabase> task has the following attributes:

Table 152. Attributes for the <configuredatabase> Ant task

Attribute	Description	Required	Default
<b>kind</b>	Type of database ( <b>Worklight</b> or <b>WorklightReports</b> )	Yes	none

It supports the following elements:

Table 153. Inner elements for the <configuredatabase> Ant attribute

Element	Description	Count
<b>derby</b>	Parameters for Derby	0..1
<b>db2</b>	Parameters for DB2	0..1
<b>mysql</b>	Parameters for MySQL	0..1
<b>oracle</b>	Parameters for Oracle	0..1
<b>driverclasspath</b>	JDBC driver class path	0..1

## For the configuration of Apache Derby databases

The element <derby> has the following attributes:

Table 154. Attributes for the <derby> element

Attribute	Description	Required	Default
<b>database</b>	Database name	No	WRKLGHT or WLREPORT, depending on kind
<b>datadir</b>	Directory that contains the databases	Yes	none
<b>schema</b>	Schema name	No	WORKLIGHT

## For the configuration of DB2 databases

The element <db2> has the following attributes:

Table 155. Attributes for the <db2> element

Attribute	Description	Required	Default
<b>database</b>	Database name	No	WRKLGHT or WLREPORT, depending on kind
<b>server</b>	Host name of database server	Yes	none
<b>port</b>	Port on database server	No	50000
<b>user</b>	User name for accessing database	Yes	none
<b>password</b>	Password for accessing database	Yes	none
<b>instance</b>	DB2 instance name	No	Depends on server
<b>schema</b>	Schema name	No	Depends on user

For more information about DB2 user accounts, see DB2 security model overview.

The <db2> element also supports an inner element <dba> that specifies database administrator credentials. This element has the following attributes:

Table 156. Attributes for the <dba> element for DB2 databases

Attribute	Description	Required	Default
<b>user</b>	User name for accessing database	Yes	none
<b>password</b>	Password or accessing database	Yes	none

The user that is specified in a <dba> element must have either the DB2 privilege SYSADM or SYSCTRL. For more information, see Authorities overview.

The <driverclasspath> element must contain a DB2 JDBC driver JAR file and an associated license JAR file. You can download DB2 JDBC drivers from DB2 JDBC Driver Versions, or you can fetch the db2jcc4.jar file and its associated db2jcc\_license\_\*.jar files from the directory DB2\_INSTALL\_DIR/java on the DB2 server.

You cannot specify details of the table allocations, such as the table space, through the Ant task. To control the table space, you must use the manual instructions in section “Configuring the DB2 databases manually” on page 678.

### For the configuration of MySQL databases

The element <mysql> has the following attributes:

Table 157. Attributes for the <mysql> element

Attribute	Description	Required	Default
<b>database</b>	Database name	No	WRKLGHT or WLREPORT, depending on kind
<b>server</b>	Host name of database server	Yes	none
<b>port</b>	Port on database server	No	3306
<b>user</b>	User name for accessing database	Yes	none
<b>password</b>	Password for accessing database	Yes	none

For more information about MySQL user accounts, see MySQL User Account Management.

The <mysql> element also supports an inner element <dba> that specifies database administrator credentials. This element has the following attributes:

Table 158. Attributes for the <dba> element for MySQL databases

Attribute	Description	Required	Default
<b>user</b>	User name for accessing database	Yes	none
<b>password</b>	Password or accessing database	Yes	none

The user that is specified in a <dba> element must be a MySQL superuser account. For more information, see Securing the Initial MySQL Accounts.

The <mysql> element also supports inner elements <client> that each specifies a client computer or a wildcard for client computers. These computers are allowed to connect to the database. This element has the following attributes:

Table 159. Attributes for the <client> element for MySQL databases

Attribute	Description	Required	Default
<b>hostname</b>	Symbolic host name, IP address, or template with % as a placeholder	Yes	none

For more details on the hostname syntax, see Specifying Account Names.

The <driverclasspath> element must contain a MySQL Connector/J JAR file. You can download it from Download Connector/J.

### For the configuration of Oracle databases

The element <oracle> has the following attributes:

Table 160. Attributes for the <oracle> element

Attribute	Description	Required	Default
<b>database</b>	Database name	No	ORCL
<b>server</b>	Host name of database server	Yes	none
<b>port</b>	Port on database server	No	1521
<b>user</b>	User name for accessing database	Yes	none
<b>password</b>	Password for accessing database	Yes	none
<b>sysPassword</b>	Password for the user SYS	Yes, if the database does not yet exist	none
<b>systemPassword</b>	Password for the user SYSTEM	△Yes, if the database or the user does not yet exist	none

For more information about Oracle user accounts, see Overview of Authentication Methods.

The <oracle> element also supports an inner element <dba> that specifies database administrator credentials. This element has the following attributes:

Table 161. Attributes for the <dba> element for Oracle databases

Attribute	Description	Required	Default
<b>user</b>	User name for accessing database	Yes	none
<b>password</b>	Password or accessing database	Yes	none

The <driverclasspath> element must contain an Oracle JDBC driver JAR file. You can download Oracle JDBC drivers from JDBC, SQLJ, Oracle JPublisher and Universal Connection Pool (UCP).

You cannot specify details of the table allocations, such as the table space, through the Ant task. To control the table space, you can create the user account manually and assign it a default table space, before invoking the Ant task. To control other details, you must use the manual instructions in section “Configuring the Oracle databases manually” on page 690.



## Creating and configuring the databases manually:

You can manually create and configure the IBM Worklight databases as detailed here.

### *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 666
2. Create the tables in the databases. This step is described in “Setting up your DB2 databases manually”
3. Perform the application server-specific setup as listed below.

### *Setting up your DB2 databases manually:*

You can set up the database manually instead of using the Ant tasks. The following topic explains how to set up the DB2 database manually.

#### **About this task**

Set up your DB2 database by creating the database schema. The procedure below creates the schemas for WRKLGHT and WLREPORT in different databases, but it is possible to group them in the same database. In this case, skip step 5.

#### **Procedure**

1. Create a system user, **worklight**, in a DB2 admin group such as **DB2USERS**, by using the appropriate commands for your operating system. Give it the password **password**. 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.

**Note:** If you want multiple IBM Worklight Servers to connect to the same database, use a different user name for each connection. Each database user has a separate default schema.

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 `~/sql1lib/bin` and enter `./db2`.
3. 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 previously created. If you defined a different user name, replace **worklight** accordingly.

4. Invoke **db2** with the following commands to create the **WRKLGHT** tables:

```
db2 CONNECT TO WRKLGHT USER worklight USING password
db2 SET CURRENT SCHEMA = 'WRKSCHM'
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 previously created, and **password** after **USING** is this user's password. If you 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.

5. 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
```

6. Invoke DB2 with the following commands to create the **WLREPORT** tables:

```
db2 CONNECT TO WLREPORT USER worklight USING password
db2 SET CURRENT SCHEMA = 'WLRESCHM'
db2 -vf <worklight_install_dir>/WorklightServer/databases/create-worklightreports-db2.sql -t
```

*Configuring Liberty Profile for DB2 manually:*

If you want to manually set up and configure your DB2 database with WebSphere Application Server Liberty Profile, use the following procedure.

### 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 (`worklightServer` may be replaced in this path by the name of your server) 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 Worklight Server project database -->
<dataSource jndiName="worklight/jdbc/WorklightDS" transactional="false">
  <jdbcDriver libraryRef="DB2Lib"/>
  <properties.db2.jcc databaseName="WRKLGHT" currentSchema="WRKSCHM"
    serverName="db2server" portNumber="50000"
    user="worklight" password="password"/>
```

```
</dataSource>
```

```
<!-- Declare the Worklight Server reports database -->  
<dataSource jndiName="worklight/jdbc/WorklightReportsDS" transactional="false">  
  <jdbcDriver libraryRef="DB2Lib"/>  
  <properties.db2.jcc databaseName="WLREPORT" currentSchema="WLRESCHM"  
    serverName="db2server" portNumber="50000"  
    user="worklight" password="password"/>  
</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 **password** after **password=** is this user's password. If you have defined either a different user name, or a different password, or both, replace these entries 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.

The `jndiName` attributes must depend on the context root that you select for the Worklight Server application, following the instructions in "Configuring the WebSphere Liberty Profile manually" on page 710. If the context root is `/app_context`, use `jndiName="app_context/jdbc/WorklightDS"` and `jndiName="app_context/jdbc/WorklightReportsDS"` respectively.

*Configuring WebSphere Application Server for DB2 manually:*

If you want to manually set up and configure your DB2 database with WebSphere Application Server, use the following procedure.

#### 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/6.0/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/6.0/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**.
  - e. Click **Next**.
  - f. Enter properties for the data source: 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**.
  - l. In **Resources > JDBC > Data sources**, select the new data source.
  - m. Click **WebSphere Application Server data source properties**.
  - n. Select the **Non-transactional data source** check box.
  - o. Click **OK**.
  - p. Click **Save**.
  - q. Click **Custom properties** for the datasource, select property **currentSchema**, and set the value to the schema used to create the datasource tables (WRKSCHM and WLRESCHM in this example).
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**.
  - h. Click **Save**.
  - i. In **Resources > JDBC > Data sources**, select the new data source.
  - j. Click **WebSphere Application Server data source properties**.
  - k. Select the **Non-transactional data source** check box.
  - l. Click **OK**.
  - m. Click **Save**.
  - n. Click **Custom properties** for the datasource, select property **currentSchema**, and set the value to the schema used to create the datasource tables (WRKSCHM and WLRESCHM in this example).
5. Test the data source connection by selecting each **Data Source** and clicking **Test Connection**.

*Configuring Apache Tomcat for DB2 manually:*

If you want to manually set up and configure your DB2 database with Apache Tomcat server, use the following procedure.

## 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"
  name="jdbc/WorklightDS"
  username="worklight"
  password="password"
  type="javax.sql.DataSource"
  url="jdbc:db2://server:50000/WRKLGHT:currentSchema=WRKSCHM;"/>
<Resource auth="Container"
  driverClassName="com.ibm.db2.jcc.DB2Driver"
  name="jdbc/WorklightReportsDS"
  username="worklight"
  password="password"
  type="javax.sql.DataSource"
  url="jdbc:db2://server:50000/WLREPORT:currentSchema=WLRESCHM;"/>
...
</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 **password** 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.

*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.

### 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 683
  - "Configuring WebSphere Application Server for Derby manually" on page 684
  - "Configuring Apache Tomcat for Derby manually" on page 686

*Setting up your Apache Derby databases manually:*

You can set up the database manually instead of using the Ant tasks. The following topic explains how to set up the Apache Derby database manually.

## About this task

Set up your Apache Derby database by creating the database schema.

### Procedure

1. 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 10.8.

**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.

2. 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';
quit;
```

*Configuring Liberty Profile for Derby manually:*

If you want to manually set up and configure your Apache Derby database with WebSphere Application Server Liberty Profile, use the following procedure.

## About this task

Complete the Apache Derby database setup procedure before continuing.

### Procedure

Configure the data source in the `$LIBERTY_HOME/usr/servers/worklightServer/server.xml` file (`worklightServer` may be replaced in this path by the name of your server) as follows:

```
<!-- Declare the jar files for Derby access through JDBC. -->
<library id="derbyLib">
  <fileset dir="C:/Drivers/derby" includes="derby.jar" />
</library>
```

```
<!-- Declare the Worklight Server project database -->
<dataSource jndiName="worklight/jdbc/WorklightDS" transactional="false" 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>
```

```
<!-- Declare the Worklight Server reports database -->
<dataSource jndiName="worklight/jdbc/WorklightReportsDS" transactional="false" statementCacheSize=
  <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>
```



The `jndiName` attributes must depend on the context root that you select for the Worklight Server application, following the instructions in “Configuring the WebSphere Liberty Profile manually” on page 710. If the context root is `/app_context`, use `jndiName="app_context/jdbc/WorklightDS"` and `jndiName="app_context/jdbc/WorklightReportsDS"` respectively.

*Configuring WebSphere Application Server for Derby manually:*

If you want to manually set up and configure your Apache Derby database for Application Center with WebSphere Application Server, use the following procedure.

### 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/6.0/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**.
  - b. Set the scope to **Node level**.
  - c. Click **New**.
  - d. Set **Database type** to **User-defined**.
  - e. Set **Class Implementation name** to `org.apache.derby.jdbc.EmbeddedConnectionPoolDataSource40`.
  - f. Set **Name** to **Worklight - Derby JDBC Provider**.
  - g. Set **Description** to **Derby JDBC provider for Worklight**.
  - h. Click **Next**.
  - i. Set the **Class path** to `WAS_INSTALL_DIR/optionalLibraries/IBM/Worklight/6.0/derby/derby.jar`.
  - j. Click **Finish**.
3. Create the data source for the IBM Worklight database.
  - a. In the WebSphere Application Server console, click **Resources > JDBC > Data sources**.
  - b. Set **Scope** to **Node level**.
  - c. Click **New**.
  - d. Set **Data source Name** to **Worklight Database**.
  - e. Set **JNDI name** to `jdbc/WorklightDS`.
  - f. Click **Next**.
  - g. Select the existing JDBC provider that is named **Worklight - Derby JDBC Provider**.
  - h. Click **Next**.
  - i. Click **Next**.
  - j. Click **Finish**.
  - k. Click **Save**.
  - l. In the table, click the **Worklight Database** datasource that you created.
  - m. Under **Additional Properties**, click **Custom properties**.



- n. Click **databaseName**.
  - o. Set **Value** to the path to the WRKLGHT database that is created by the configuredatabase ant task.
  - p. Click **OK**.
  - q. Click **Save**.
  - r. At the top of the page, click **Worklight Database**.
  - s. Under **Additional Properties**, click **WebSphere Application Server data source properties**.
  - t. Select **Non-transactional datasource**.
  - u. Click **OK**.
  - v. Click **Save**.
  - w. In the table, select the **Worklight Database** datasource that you created.
  - x. Click **test connection** (only if you are not on the console of a WAS Deployment Manager).
4. Set up the data source for the IBM Worklight report database.
    - a. In the WebSphere Application Server console, click **Resources > JDBC > Data sources**.
    - b. Set **Scope** to **Node level**.
    - c. Click **New**.
    - d. Set **Data source name** to **Worklight Reports Database**.
    - e. Set **JNDI name** to **jdbc/WorklightReportsDS**.
    - f. Click **Next**.
    - g. Select the existing JDBC provider that is named **Worklight - Derby JDBC Provider**.
    - h. Click **Next**.
    - i. Click **Next**.
    - j. Click **Finish**.
    - k. Click **Save**.
    - l. In the table, click the **Worklight Reports Database** datasource that you created.
    - m. Under **Additional properties**, click **Custom properties**.
    - n. Click **databaseName**.
    - o. Set **Value** to the path to the WLREPORT database that is created by the configuredatabase ant task.
    - p. Click **OK**.
    - q. Click **Save**.
    - r. At the top of the page, click **Worklight Reports Database**.
    - s. Under **Additional Properties**, click **WebSphere Application Server data source properties**.
    - t. Select **Non-transactional datasource**.
    - u. Click **OK**.
    - v. Click **Save**.
    - w. In the table, select the **Worklight Reports Database** datasource that you created.
    - x. Click **test connection** (only if you are not on the console of a WAS Deployment Manager).

*Configuring Apache Tomcat for Derby manually:*

If you want to manually set up and configure your Apache Derby database with the Apache Tomcat server, use the following procedure.

### **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"
  username="WORKLIGHT"
  password=""
  type="javax.sql.DataSource"
  url="jdbc:derby:DERBY_DATABASES_DIR/WRKLGHT"/>
<Resource auth="Container"
  driverClassName="org.apache.derby.jdbc.EmbeddedDriver"
  name="jdbc/WorklightReportsDS"
  username="WORKLIGHT"
  password=""
  type="javax.sql.DataSource"
  url="jdbc:derby:DERBY_DATABASES_DIR/WLREPORT"/>
...
</Context>
```

*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 667
2. Create the tables in the databases. This step is described in “Setting up your MySQL databases manually”
3. Perform the application server-specific setup as listed below.

*Setting up your MySQL databases manually:*

You can set up the database manually instead of using the Ant tasks. The following topic explains how to set up the MySQL database manually.

### **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 'password';
GRANT ALL privileges ON WRKLGHT.* TO 'worklight'@'localhost' IDENTIFIED BY 'password';
Flush privileges;
CREATE DATABASE WLREPORT CHARACTER SET utf8 COLLATE utf8_general_ci;
GRANT ALL privileges ON WLREPORT.* TO 'worklight'@'Worklight-host' IDENTIFIED BY 'password';
GRANT ALL privileges ON WLREPORT.* TO 'worklight'@'localhost' IDENTIFIED BY 'password';
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;
```

Where **worklight** before the @ sign is the user name, **password** after **IDENTIFIED BY** is its password, and **Worklight-host** is the name of the host on which IBM Worklight runs.

2. Add the following property to your MySQL option file: `max_allowed_packet=16M`  
For more details about `max_allowed_packet`, see the MySQL documentation, section Packet Too Large.  
For more information about option files, see the MySQL documentation at MySQL.

*Configuring Liberty Profile for MySQL manually:*

If you want to manually set up and configure your MySQL database with WebSphere Application Server Liberty Profile, use the following procedure.

#### 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/usr/servers/worklightServer/server.xml` file (`worklightServer` may be replaced in this path by the name of your server) 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 Worklight Server project database -->
<dataSource jndiName="worklight/jdbc/WorklightDS" transactional="false">
  <jdbcDriver libraryRef="MySQLLib"/>
  <properties databaseName="WRKLGHT"
    serverName="mysqlserver" portNumber="3306"
    user="worklight" password="password"/>
</dataSource>

<!-- Declare the Worklight Server reports database -->
<dataSource jndiName="worklight/jdbc/WorklightReportsDS" transactional="false">
  <jdbcDriver libraryRef="MySQLLib"/>
```

```

    <properties databaseName="WLREPORT"
      serverName="mysqlserver" portNumber="3306"
      user="worklight" password="password"/>
  </dataSource>

```

where **worklight** after **user=** is the user name, **password** 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).

The `jndiName` attributes must depend on the context root that you select for the Worklight Server application, following the instructions in "Configuring the WebSphere Liberty Profile manually" on page 710. If the context root is `/app_context`, use `jndiName="app_context/jdbc/WorklightDS"` and `jndiName="app_context/jdbc/WorklightReportsDS"` respectively.

*Configuring WebSphere Application Server for MySQL manually:*

If you want to manually set up and configure your MySQL database for Application Center with WebSphere Application Server, use the following procedure.

### 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. 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.
  - 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. 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`.
  - 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. 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 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
  5. Set the WAS custom properties of each new data source.
    - a. In **Resources > JDBC > Data sources**, select the new data source.
    - b. Click **WebSphere Application Server data source properties**.
    - c. Select the **Non-transactional data source** checkbox.
    - d. Click **OK**.
    - e. Click **Save**.

*Configuring Apache Tomcat for MySQL manually:*

If you want to manually set up and configure your MySQL database with the Apache Tomcat server, use the following procedure.

#### **About this task**

Complete the MySQL database setup procedure before continuing.

#### **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="password"
  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="password"
```

```

        driverClassName="com.mysql.jdbc.Driver"
        url="jdbc:mysql://server:3306/WLREPORT"/>
        ...
    </Context>

```

*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 668
2. Create the tables in the databases. This step is described in “Setting up your Oracle databases manually”
3. Perform the application server-specific setup as listed below.

*Setting up your Oracle databases manually:*

You can set up the database manually instead of using the Ant tasks. The following topic explains how to set up the Oracle database manually.

#### **About this task**

Complete the following procedure to set up your Oracle databases.

#### **Procedure**

1. Ensure that you have at least one Oracle database. In many Oracle installations, the default database has the SID (name) ORCL. For best results, the character set of the database should be set to **Unicode (AL32UTF8)**.
2. Create the user WORKLIGHT, either by using Oracle Database Control, or by using the Oracle SQLPlus command-line interpreter.

Create the user for the IBM Worklight database/schema, 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 WORKLIGHTREPORTS for the IBM Worklight reports database/schema and a user APPCENTER for the IBM Application Center database/schema.

To create the two users by using Oracle SQLPlus, enter the following commands:

```

CONNECT system/system_password@ORCL
CREATE USER WORKLIGHT IDENTIFIED BY WORKLIGHT_password;
GRANT CONNECT, RESOURCE, CREATE VIEW TO WORKLIGHT;
DISCONNECT;
CONNECT system/system_password@ORCL
CREATE USER WORKLIGHTREPORTS IDENTIFIED BY WORKLIGHTREPORTS_password;
GRANT CONNECT, RESOURCE, CREATE VIEW TO WORKLIGHTREPORTS;
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 by running the create-worklight-oracle.sql file:

```

CONNECT WORKLIGHT/<WORKLIGHT_password>@ORCL
@<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 by running the create-worklightreports-oracle.sql file:

```

CONNECT WORKLIGHTREPORTS/<WORKLIGHTREPORTS_password>@ORCL
@<worklight_install_dir>/WorklightServer/databases/create-worklightreports-oracle.sql
DISCONNECT;

```

4. 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:*

If you want to manually set up and configure your Oracle database with WebSphere Application Server Liberty Profile, use the following procedure.

### 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 (worklightServer may be replaced in this path by the name of your server) 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 Worklight Server project database -->
<dataSource jndiName="worklight/jdbc/WorklightDS" transactional="false">
  <jdbcDriver libraryRef="OracleLib"/>
  <properties.oracle driverType="thin" databaseName="ORCL"
    serverName="oserver" portNumber="1521"
    user="WORKLIGHT" password="WORKLIGHT_password"/>
</dataSource>

```



```

<!-- Declare the Worklight Server reports database -->
<dataSource jndiName="worklight/jdbc/WorklightReportsDS" transactional="false">
  <jdbcDriver libraryRef="OracleLib"/>
  <properties.oracle driverType="thin" databaseName="ORCL"
    serverName="oserver" portNumber="1521"
    user="WORKLIGHTREPORTS" password="WORKLIGHTREPORTS_password"/>
</dataSource>

```

Where **oserver** is the host name of your Oracle server (for example, localhost, if it is on the same machine).

The `jndiName` attributes must depend on the context root that you select for the Worklight Server application, following the instructions in “Configuring the WebSphere Liberty Profile manually” on page 710. If the context root is `/app_context`, use `jndiName="app_context/jdbc/WorklightDS"` and `jndiName="app_context/jdbc/WorklightReportsDS"` respectively.

*Configuring WebSphere Application Server for Oracle manually:*

If you want to manually set up and configure your DB2 database with WebSphere Application Server, use the following procedure.

**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 162. 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**.
  - 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/ORCL`, 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_password`.
  - m. Click **OK** and save the changes.
  - n. In **Resources > JDBC > Data sources**, select the new data source.
  - o. Click **WebSphere Application Server data source properties**.
  - p. Check the **Non-transactional data source** check box.
  - q. Click **OK**.
  - r. Click **Save**.
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 user name `WORKLIGHTREPORTS` and its corresponding password.

*Configuring Apache Tomcat for Oracle manually:*

If you want to manually set up and configure your Oracle database with the Apache Tomcat server, use the following procedure.

#### 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"
  driverClassName="oracle.jdbc.driver.OracleDriver"
  url="jdbc:oracle:thin:@oserver:1521/ORCL"
  username="WORKLIGHT"
  password="WORKLIGHT_password"/>
<Resource name="jdbc/WorklightReportsDS"
  auth="Container"
  type="javax.sql.DataSource"
  driverClassName="oracle.jdbc.driver.OracleDriver"
  url="jdbc:oracle:thin:@oserver:1521/ORCL"
  username="WORKLIGHTREPORTS"
  password="WORKLIGHTREPORTS_password"/>
...
</Context>
```

Where **oserver** is the host name of your Oracle server (for example, localhost, if it is on the same machine).

## Deploying a project WAR file and configuring the application server

You can deploy a project WAR file to an application server, and configure the application server, as detailed here.

### Deploying a project WAR file and configuring the application server with Ant tasks:

Use Ant tasks to deploy the project WAR file to an application server, and configure data sources, properties, and database drivers for use by IBM Worklight as detailed here.

An Ant task is supplied that can configure an application server for a Worklight project. This configured application includes the Worklight console. The input for this configuration is the Worklight project WAR file. The task:

- Installs the project WAR file as an application in the application server.
- Configures the data sources for IBM Worklight and for the IBM Worklight reports.
- Configures Worklight configuration properties (through JNDI).
- Installs the database drivers and the Worklight Server runtime library (`worklight-jee-library.jar`) in the application server.

To start the Ant task, you need an Ant XML file with one or more invocations of the `<configureapplicationserver>` task. If you want to start the Ant task from a computer on which Worklight Server is not installed, you must copy the files `<WorklightInstallDir>/WorklightServer/worklight-ant.jar` and `<WorklightInstallDir>/WorklightServer/worklight-jee-library.jar` to this computer.

Before starting the `<configureapplicationserver>` task, you must create and configure the databases. See “Creating and configuring the databases with Ant tasks” on page 669 for details.

**Note:** The `<database>` elements that are passed to `<configureapplicationserver>` must be consistent with the `<configuredatabase>` invocations that were used when configuring the databases.

Another Ant task is supplied that removes the configuration that was done by an earlier `<configureapplicationserver>` invocation. This Ant task is named `<unconfigureapplicationserver>` and takes the same parameters as the `<configureapplicationserver>` invocation.

Sample Ant XML files are presented in the following sections.

### Sample Ant XML file for WebSphere Application Server Liberty Profile with Derby databases

**Note:** The Apache Derby database is provided in the IBM Worklight distribution, but is not suggested for use in production environments. In these environments, an IBM DB2, MySQL, or Oracle database is more common and more appropriate. The following Ant XML examples are provided in case you want to install the Apache Derby database in a test application server environment.

```
<?xml version="1.0" encoding="UTF-8"?>
<project basedir="." default="install">
  <taskdef resource="com/worklight/ant/defaults.properties">
```

```

    <classpath>
      <pathelement location="/opt/IBM/Worklight/WorklightServer/worklight-ant.jar"/>
    </classpath>
  </taskdef>

  <target name="databases">
    <configuredatabase kind="Worklight">
      <derby database="WRKLGHT" datadir="/var/ibm/Worklight/derby"/>
    </configuredatabase>
    <configuredatabase kind="WorklightReports">
      <derby database="WLREPORT" datadir="/var/ibm/Worklight/derby"/>
    </configuredatabase>
  </target>

  <target name="install">
    <configureapplicationserver shortcutsDir="/tmp/shortcuts">
      <project warfile="/n/download/testWorklight.war"/>

      <!-- Here you can define values which override the
           default values of Worklight configuration properties -->
      <property name="serverSessionTimeout" value="10"/>

      <applicationserver>
        <websphereapplicationserver installDir="/n/java/webservers/was-liberty-8.5-express"
                                   profile="Liberty">
          <server name="server1"/>
        </websphereapplicationserver>
      </applicationserver>
      <database kind="Worklight">
        <derby database="WRKLGHT" datadir="/var/ibm/Worklight/derby"/>
      </database>
      <database kind="WorklightReports">
        <derby database="WLREPORT" datadir="/var/ibm/Worklight/derby"/>
      </database>
    </configureapplicationserver>
  </target>

  <target name="uninstall">
    <unconfigureapplicationserver shortcutsDir="/tmp/shortcuts">
      <project warfile="/n/download/testWorklight.war"/>
      <property name="serverSessionTimeout" value="10"/>
      <applicationserver>
        <websphereapplicationserver installDir="/n/java/webservers/was-liberty-8.5-express"
                                   profile="Liberty">
          <server name="server1"/>
        </websphereapplicationserver>
      </applicationserver>
      <database kind="Worklight">
        <derby database="WRKLGHT" datadir="/var/ibm/Worklight/derby"/>
      </database>
      <database kind="WorklightReports">
        <derby database="WLREPORT" datadir="/var/ibm/Worklight/derby"/>
      </database>
    </unconfigureapplicationserver>
  </target>
</project>

```

### Sample Ant XML file for an Apache Tomcat server with MySQL databases

```

<?xml version="1.0" encoding="UTF-8"?>
<project basedir="." default="install">
  <taskdef resource="com/worklight/ant/defaults.properties">
    <classpath>
      <pathelement location="/opt/IBM/Worklight/WorklightServer/worklight-ant.jar"/>
    </classpath>
  </taskdef>

```

```

<target name="databases">
  <configuredatabase kind="Worklight">
    <mysql database="WRKLGHT" server="proddb.example.com" user="w16test" password="w16pass">
      <dba user="root" password="UnGuessable"/>
      <client hostname="localhost"/>
      <client hostname="prodtomcat.example.com"/>
    </mysql>
    <driverclasspath>
      <pathelement location="/opt/database-drivers/mysql/mysql-connector-java-5.1.25-bin.jar"/>
    </driverclasspath>
  </configuredatabase>
  <configuredatabase kind="WorklightReports">
    <mysql database="WLREPORT" server="proddb.example.com" user="w16test" password="w16pass">
      <dba user="root" password="UnGuessable"/>
      <client hostname="localhost"/>
      <client hostname="prodtomcat.example.com"/>
    </mysql>
    <driverclasspath>
      <pathelement location="/opt/database-drivers/mysql/mysql-connector-java-5.1.25-bin.jar"/>
    </driverclasspath>
  </configuredatabase>
</target>

<target name="install">
  <configureapplicationserver shortcutsDir="/tmp/shortcuts">
    <project warfile="/n/download/testWorklight.war"/>

    <!-- Here you can define values which override the
         default values of Worklight configuration properties -->
    <property name="serverSessionTimeout" value="10"/>

    <applicationserver>
      <tomcat installdir="/n/java/webservers/tomcat-7.0.23"/>
    </applicationserver>
    <database kind="Worklight">
      <mysql database="WRKLGHT" server="proddb.example.com" user="w16test" password="w16pass"/>
      <driverclasspath>
        <pathelement location="/opt/database-drivers/mysql/mysql-connector-java-5.1.25-bin.jar"/>
      </driverclasspath>
    </database>
    <database kind="WorklightReports">
      <mysql database="WLREPORT" server="proddb.example.com" user="w16test" password="w16pass"/>
      <driverclasspath>
        <pathelement location="/opt/database-drivers/mysql/mysql-connector-java-5.1.25-bin.jar"/>
      </driverclasspath>
    </database>
  </configureapplicationserver>
</target>

<target name="uninstall">
  <unconfigureapplicationserver shortcutsDir="/tmp/shortcuts">
    <project warfile="/n/download/testWorklight.war"/>
    <property name="serverSessionTimeout" value="10"/>
    <applicationserver>
      <tomcat installdir="/n/java/webservers/tomcat-7.0.23"/>
    </applicationserver>
    <database kind="Worklight">
      <mysql database="WRKLGHT" server="proddb.example.com" user="w16test" password="w16pass"/>
      <driverclasspath>
        <pathelement location="/opt/database-drivers/mysql/mysql-connector-java-5.1.25-bin.jar"/>
      </driverclasspath>
    </database>
    <database kind="WorklightReports">
      <mysql database="WLREPORT" server="proddb.example.com" user="w16test" password="w16pass"/>
      <driverclasspath>
        <pathelement location="/opt/database-drivers/mysql/mysql-connector-java-5.1.25-bin.jar"/>
      </driverclasspath>
    </database>
  </unconfigureapplicationserver>
</target>

```

```

        </database>
    </unconfigureapplicationserver>
</target>
</project>

```

### Sample Ant XML file for a WebSphere Application Server Full Profile with DB2 databases

```

<?xml version="1.0" encoding="UTF-8"?>
<project basedir="." default="install">
  <taskdef resource="com/worklight/ant/defaults.properties">
    <classpath>
      <pathelement location="/opt/IBM/Worklight/WorklightServer/worklight-ant.jar"/>
    </classpath>
  </taskdef>

  <target name="databases">
    <configuredatabase kind="Worklight">
      <db2 database="WRKLGHT" server="proddb.example.com" user="w16test" password="w16pass">
        <dba user="db2inst1" password="db2IsFun"/>
      </db2>
      <driverclasspath>
        <fileset dir="/opt/database-drivers/db2-9.7">
          <include name="db2jcc4.jar"/>
          <include name="db2jcc_license_*.jar"/>
        </fileset>
      </driverclasspath>
    </configuredatabase>
    <configuredatabase kind="WorklightReports">
      <db2 database="WLREPORT" server="proddb.example.com" user="w16test" password="w16pass">
        <dba user="db2inst1" password="db2IsFun"/>
      </db2>
      <driverclasspath>
        <fileset dir="/opt/database-drivers/db2-9.7">
          <include name="db2jcc4.jar"/>
          <include name="db2jcc_license_*.jar"/>
        </fileset>
      </driverclasspath>
    </configuredatabase>
  </target>

  <target name="install">
    <configureapplicationserver shortcutsDir="/tmp/shortcuts">
      <project warfile="/n/download/testWorklight.war" libraryfile="/n/download/worklight-jee-lib

      <!-- Here you can define values which override the
           default values of Worklight configuration properties -->
      <property name="serverSessionTimeout" value="10"/>

      <applicationserver>
        <websphereapplicationserver installDir="/n/java/webservers/was-8.0-express"
                                   profile="AppSrv01"
                                   user="admin" password="admin">
          <server name="server1"/>
        </websphereapplicationserver>
      </applicationserver>
      <database kind="Worklight">
        <db2 database="WRKLGHT" server="proddb.example.com" user="w16test" password="w16pass"/>
        <driverclasspath>
          <fileset dir="/opt/database-drivers/db2-9.7">
            <include name="db2jcc4.jar"/>
            <include name="db2jcc_license_*.jar"/>
          </fileset>
        </driverclasspath>
      </database>
      <database kind="WorklightReports">
        <db2 database="WLREPORT" server="proddb.example.com" user="w16test" password="w16pass"/>

```

```

        <driverclasspath>
        <fileset dir="/opt/database-drivers/db2-9.7">
        <include name="db2jcc4.jar"/>
        <include name="db2jcc_license_*.jar"/>
        </fileset>
        </driverclasspath>
    </database>
</configureapplicationserver>
</target>

<target name="uninstall">
    <unconfigureapplicationserver shortcutsDir="/tmp/shortcuts">
    <project warfile="/n/download/testWorklight.war" libraryfile="/n/download/worklight-jee-library">
    <property name="serverSessionTimeout" value="10"/>
    <applicationserver>
    <websphereapplicationserver installDir="/n/java/webservers/was-8.0-express"
        profile="AppSrv01"
        user="admin" password="admin">
        <server name="server1"/>
    </websphereapplicationserver>
    </applicationserver>
    <database kind="Worklight">
    <db2 database="WRKLGHT" server="proddb.example.com" user="wl6test" password="wl6pass"/>
    <driverclasspath>
    <fileset dir="/opt/database-drivers/db2-9.7">
    <include name="db2jcc4.jar"/>
    <include name="db2jcc_license_*.jar"/>
    </fileset>
    </driverclasspath>
    </database>
    <database kind="WorklightReports">
    <db2 database="WLREPORT" server="proddb.example.com" user="wl6test" password="wl6pass"/>
    <driverclasspath>
    <fileset dir="/opt/database-drivers/db2-9.7">
    <include name="db2jcc4.jar"/>
    <include name="db2jcc_license_*.jar"/>
    </fileset>
    </driverclasspath>
    </database>
    </unconfigureapplicationserver>
</target>
</project>

```

*Ant **configureapplicationserver** task reference:*

Reference information for the Ant `<configureapplicationserver>` and `<unconfigureapplicationserver>` tasks.

The `<configureapplicationserver>` and `<unconfigureapplicationserver>` tasks have the following attributes:

*Table 163. Attributes for the `<configureapplicationserver>` and `<unconfigureapplicationserver>` Ant tasks*

Attribute	Description	Required	Default
<b>contextroot</b>	Common prefix of path of URLs to the application (context root)	No	/worklight
<b>id</b>	Distinguishes different deployments	No	empty
<b>shortcutsDir</b>	Directory where to place shortcuts	No	none

The `contextroot` and `id` attributes distinguish different Worklight projects. By default, when a project is created in Worklight Studio V6.0.0 and higher, its context



root is the name of the project; the default value of /worklight was chosen to facilitate backwards compatibility with Worklight V5.x applications.

In WebSphere Application Server Liberty profile and in Tomcat environments, the contextroot parameter is sufficient for this purpose. In WebSphere Application Server Full profile environments, the id attribute is used instead.

The shortcutsDir attribute specifies where to place shortcuts to the Worklight console. If this attribute is set, three files can be added to this directory:

- A file worklight-console.url. This file is a Windows shortcut. It opens the Worklight console in a browser.
- A file worklight-console.sh. This file is a UNIX shell script that opens the Worklight console in a browser.
- A file worklight-console.html. This file is a web page for testing Worklight console deployments in WebSphere Application Server Network Deployment environments.

The <configureapplicationserver> and <unconfigureapplicationserver> tasks support the following elements:

Table 164. Inner elements for the <configureapplicationserver> and <unconfigureapplicationserver> Ant tasks

Element	Description	Count
<b>project</b>	Project	1
<b>property</b>	Properties	0..∞
<b>applicationserver</b>	Application server	1
<b>reports</b>	Reports	0..1
<b>database</b>	Databases	2

The element <project> specifies details about the project to deploy into the application server. It has the following attributes:

Table 165. Attributes for the <project> element

Attribute	Description	Required	Default
<b>warfile</b>	Project WAR file	Yes	none
<b>libraryfile</b>	File name of worklight-jee-library.jar	No	next to worklight-ant.jar

The warfile is created through the <war-builder> Ant task. See “Building a project WAR file with Ant” on page 665.

The element <property> specifies a deployment property that is to be defined in the application server. It has the following attributes:

Table 166. Attributes for the <property> element

Attribute	Description	Required	Default
<b>name</b>	Name of the property	Yes	none
<b>value</b>	Value for the property	Yes	none

For general information about Worklight properties, or for a list of properties that can be set,, see “Configuration of IBM Worklight applications on the server” on page 714.

The element <applicationserver> describes the application server into which to deploy the Worklight application. It is a container for one of the following elements:

Table 167. Inner elements for the <applicationserver> element

Element	Description	Count
<b>websphereapplicationserver</b> or <b>was</b>	Parameters for WebSphere Application Server	0..1
<b>tomcat</b>	Parameters for Apache Tomcat	0..1

The element <websphereapplicationserver> (or <was> in its short form) denotes a WebSphere Application Server, version 7.0 or newer. WebSphere Application Server Full Profile (Express, Base, and Network Deployment) are supported, as is Liberty Profile (Core). Liberty Profile Network Deployment is not yet supported. The element <websphereapplicationserver> has the following attributes:

Table 168. Attributes for the <websphereapplicationserver> or <was> element

Attribute	Description	Required	Default
<b>installdir</b>	WebSphere Application Server installation directory.	Yes	none
<b>profile</b>	WebSphere Application Server profile, or Liberty	Yes	none
<b>user</b>	WebSphere Application Server administrator name	Yes, except for Liberty	none
<b>password</b>	WebSphere Application Server administrator password	Yes, except for Liberty	none

It supports the following elements for single-server deployment:

Table 169. Inner elements for the <was> element (single-server deployment)

Element	Description	Count
<b>server</b>	A single server	0..1

The element <server>, used in this context, has the following attributes:

Table 170. Inner elements for the <server> element (single-server deployment)

Attribute	Description	Required	Default
<b>name</b>	Server name	Yes	none

It supports the following elements for Network Deployment:

Table 171. Inner elements for the <was> element (network deployment)

Element	Description	Count
<b>cell</b>	The entire cell	0..1
<b>cluster</b>	All servers of a cluster	0..1
<b>node</b>	All servers in a node, excluding clusters	0..1
<b>server</b>	A single server	0..1

The element <cell> has no attributes.

The element <cluster> has the following attributes:

Table 172. Attributes for the <cluster> element (network deployment)

Attribute	Description	Required	Default
<b>name</b>	Cluster name	Yes	none

The element <node> has the following attributes:

Table 173. Attributes for the <node> element (network deployment)

Attribute	Description	Required	Default
<b>name</b>	Node name	Yes	none

The element <server>, used in a Network Deployment context, has the following attributes:

Table 174. Attributes for the <server> element (network deployment)

Attribute	Description	Required	Default
<b>nodeName</b>	Node name	Yes	none
<b>serverName</b>	Server name	Yes	none

The element <tomcat> denotes an Apache Tomcat server. It has the following attributes:

Table 175. Attributes of the <tomcat> element

Attribute	Description	Required	Default
<b>installDir</b>	Apache Tomcat installation directory.	Yes	none

The element <reports> specifies a set of reports (BIRT \*.rptdesign files) to instantiate so that they can access the Worklight reports database.

The element <reports> has the following attributes:

Table 176. Attributes of the <reports> element

Attribute	Description	Required	Default
<b>toDir</b>	Destination directory	Yes	none

It supports the following elements:

Table 177. Inner elements for the <reports> element

Element	Description	Count
<b>fileset</b>	Set of files to copy and process	0..∞

A <reports> element without any inner <fileset> element instantiates all report templates provided in the WorklightServer/report-templates/ directory in the Worklight Server distribution.

The element <database> specifies the information that is needed to access a particular database. Two databases must be declared: <database kind="Worklight"> and <database kind="WorklightReports">. The element <database> is specified like the <configuredatabase> task, except that it does not have the elements <dba> and <client>. It might, however, have <property> elements. The element <database> has the following attributes:

Table 178. Attributes of the <database> element

Attribute	Description	Required	Default
<b>kind</b>	Type of database ( <i>Worklight</i> or <i>WorklightReports</i> )	Yes	none

It supports the following elements:

Table 179. Inner elements for the <database> element

Element	Description	Count
<b>derby</b>	Parameters for Derby	0..1
<b>db2</b>	Parameters for DB2	0..1
<b>mysql</b>	Parameters for MySQL	0..1
<b>oracle</b>	Parameters for Oracle	0..1
<b>driverclasspath</b>	JDBC driver class path	0..1

### To specify an Apache Derby database

The element <derby> has the following attributes:

Table 180. Attributes of the <derby> element

Attribute	Description	Required	Default
<b>database</b>	Database name	No	<i>WRKLIGHT</i> or <i>WLREPORT</i> , depending on kind
<b>datadir</b>	Directory that contains the databases	Yes	none
<b>schema</b>	Schema name	No	<i>WORKLIGHT</i>

It supports the following elements:

Table 181. Inner elements for the <derby> element

Element	Description	Count
<b>property</b>	Data source property or JDBC connection property	0..∞

For the available properties, see the documentation for class **EmbeddedDataSource40** at Class EmbeddedDataSource40. See also the documentation for class **EmbeddedConnectionPoolDataSource40** at Class EmbeddedConnectionPoolDataSource40.

For the available properties for a Liberty server, see the documentation for **properties.derby.embedded** at Liberty profile: Configuration elements in the server.xml file.

A <driverclasspath> element is not needed in this case, when the worklight-ant.jar is used within the installation directory of Worklight.

### To specify a DB2 database

The element <db2> has the following attributes:

Table 182. Attributes of the <db2> element

Attribute	Description	Required	Default
<b>database</b>	Database name	No	WRKLGHT or WLREPORT, depending on kind
<b>server</b>	Host name of database server	Yes	none
<b>port</b>	Port on database server	No	50000
<b>user</b>	User name for accessing database	Yes	none
<b>password</b>	Password for accessing database	Yes	none
<b>schema</b>	Schema name	No	Depends on user

For more information about DB2 user accounts, see DB2 security model overview.

It supports the following elements:

Table 183. Inner elements for the <db2> element

Element	Description	Count
<b>property</b>	Data source property or JDBC connection property	0..∞

For the available properties, see Properties for the IBM Data Server Driver for JDBC and SQLJ.

For the available properties for a Liberty server, see the **properties.db2.jcc** section at Liberty profile: Configuration elements in the server.xml file.

The <driverclasspath> element must contain a DB2 JDBC driver JAR file and an associated license JAR file. You can download DB2 JDBC drivers from DB2 JDBC Driver Versions.

### To specify a MySQL database

The element <mysql> has the following attributes:

Table 184. Attributes of the <mysql> element

Attribute	Description	Required	Default
<b>database</b>	Database name	No	WRKLGHT or WLREPORT, depending on kind
<b>server</b>	Host name of database server	Yes	none
<b>port</b>	Port on database server	No	3306
<b>user</b>	User name for accessing database	Yes	none
<b>password</b>	Password for accessing database	Yes	none

A URL can also be specified instead of database, server, and port. The alternative list of attributes is as follows:

Table 185. Alternative elements for the <mysql> element

Attribute	Description	Required	Default
<b>url</b>	Database connection URL	Yes	none
<b>user</b>	User name for accessing database	Yes	none
<b>password</b>	Password for accessing database	Yes	none

For more information about MySQL user accounts, see MySQL User Account Management.

It supports the following elements:

Table 186. Inner elements for the <mysql> element

Element	Description	Count
<b>property</b>	Data source property or JDBC connection property	0..∞

For the available properties, see the documentation at Driver/Datasource Class Names, URL Syntax and Configuration Properties for Connector/J.

For the available properties for a Liberty server, see the **properties** section at Liberty profile: Configuration elements in the server.xml file.

The <driverclasspath> element must contain a MySQL Connector/J JAR file. You can download it from Download Connector/J.

## To specify an Oracle database

The element `<oracle>` has the following attributes:

Table 187. Attributes of the `<oracle>` element

Attribute	Description	Required	Default
<b>database</b>	Database name	No	ORCL
<b>server</b>	Host name of database server	Yes	none
<b>port</b>	Port on database server	No	1521
<b>user</b>	User name for accessing database	Yes	none
<b>password</b>	Password for accessing database	Yes	none

A URL can also be specified instead of database, server, and port. The alternative list of attributes is as follows:

Table 188. Alternative attributes of the `<oracle>` element

Attribute	Description	Required	Default
<b>url</b>	Database connection URL	Yes	none
<b>user</b>	User name for accessing database	Yes	none
<b>password</b>	Password for accessing database	Yes	none

For more information about Oracle user accounts, see [Overview of Authentication Methods](#).

For details on Oracle database connection URLs, see the [Database URLs and Database Specifiers](#) section at [Data Sources and URLs](#).

It supports the following elements:

Table 189. Inner elements for the `<oracle>` element

Element	Description	Count
<b>property</b>	Data source property or JDBC connection property	0..∞

For the available properties, see the [Data Sources and URLs](#) section at [Data Sources and URLs](#).

For the available properties for a Liberty server, see the [properties.oracle](#) section at Liberty profile: Configuration elements in the `server.xml` file.

The `<driverclasspath>` element must contain an Oracle JDBC driver JAR file. You can download Oracle JDBC drivers from JDBC, SQLJ, Oracle JPublisher and Universal Connection Pool (UCP).

The `<property>` element, which can be used inside `<derby>`, `<db2>`, `<mysql>`, or `<oracle>` elements, has the following attributes:



Table 190. Attributes for the <property> element in a database-specific element

Attribute	Description	Required	Default
<b>name</b>	Name of the property	Yes	none
<b>type</b>	Java type of the property's values (usually java.lang.String/Integer/Boolean)	No	java.lang.String
<b>value</b>	Value for the property	Yes	none

Sample configuration files:

IBM Worklight includes a number of sample configuration files to help you get started with the Ant tasks.

The easiest way to get started with the <configureapplicationserver> and <configuredatabase> Ant tasks is by working with the sample configuration files provided in the WorklightServer/configuration-samples/ directory of the Worklight Server distribution.

### Step 1

Pick the appropriate sample configuration file. The following files are provided

Table 191. Sample configuration files provided with IBM Worklight

Sample	Derby	DB2	MySQL	Oracle
WebSphere Application Server Liberty profile	configure-liberty-derby.xml	configure-liberty-db2.xml	configure-liberty-mysql.xml	configure-liberty-oracle.xml
WebSphere Application Server Full profile, single-server	configure-was-derby.xml	configure-was-db2.xml	configure-was-mysql.xml	configure-was-oracle.xml
WebSphere Application Server Network Deployment	n/a	configure-wasnd-db2.xml	configure-wasnd-mysql.xml	configure-wasnd-oracle.xml
Apache Tomcat	configure-tomcat-derby.xml	configure-tomcat-db2.xml	configure-tomcat-mysql.xml	configure-tomcat-oracle.xml

### Step 2

Change the file access rights of the sample file to be as restrictive as possible. Step 3 requires that you supply some passwords. If you must prevent other users on the same computer from learning these passwords, you must remove the read permissions of the file for users other than yourself. You can use a command, such as the following examples:

- On UNIX:  

```
chmod 600 configure-file.xml
```

- On Windows:  
`cacls configure-file.xml /P Administrators:F %USERDOMAIN%\%USERNAME%:F`

### Step 3

Similarly, if the server is a WebSphere Application Server Liberty Profile or Apache Tomcat server, and the server is meant to be started only from your user account, you must also remove the read permissions for users other than yourself from the following file:

- For WAS Liberty profile: `wlp/usr/servers/<server>/server.xml`
- For Apache Tomcat: `conf/server.xml`

### Step 4

Replace the placeholder values for the properties at the top of the file.

**Note:** The following special characters need to be escaped when used in values in Ant XML scripts:

- The dollar sign (\$) must be written as \$\$, unless you explicitly want to reference an Ant variable through the syntax `${variable}`, as described in Properties in the *Apache Ant Manual*.
- The ampersand character (&) must be written as `&amp;`, unless you explicitly want to reference an XML entity.
- Double quotation marks (") must be written as `&quot;`, except when inside a string enclosed in single quotation marks.

### Step 5

In the `<configureapplicationserver>` and `<unconfigureapplicationserver>` invocations (in target `install` and `uninstall`), define Worklight properties. For a list of properties that can be set, see “Configuration of IBM Worklight applications on the server” on page 714. In production, you must often define the following specific properties:

- `publicWorkLightHostname`
- `publicWorkLightProtocol`
- `publicWorkLightPort`

### Step 6

Run the command:

```
ant -f configure-file.xml databases
```

This command ensures that the designated databases exist and contain the required tables for Worklight.

### Step 7

Run the command:

```
ant -f configure-file.xml install
```

This command installs your Worklight project as a `.war` file into the application server.

To reverse Step 6, run the command:

```
ant -f configure-file.xml uninstall
```

This command uninstalls the Worklight project .war file.

At least for WebSphere Application Server, it is a good idea to keep the modified `configure-file.xml` for later use when you install updates of the Worklight project's .war file. This file makes it possible to redeploy an updated .war file with the same Worklight properties. If you use the WebSphere Application Server's administrative console to update the .war file, all properties that are configured for this web application are lost.

*Configuring multiple IBM Worklight projects:*

The Ant tasks that are used to configure multiple projects on a single server are documented in this section.

It is possible to install different IBM Worklight projects WAR files in the same application server, and have them operate in parallel and independently. This configuration is possible even for IBM Worklight projects that use different versions of IBM Worklight. For example, several IBM Worklight V6.0.x projects and at most one IBM Worklight V5.0.6 project can be used at the same time in the same application server.

Note that each IBM Worklight project presents its own Worklight Console, and that uploading an app or an adapter to one of the Worklight Consoles has no effect on the other Worklight Consoles.

If you use this configuration, some constraints must be respected:

- Each IBM Worklight project configuration must use a different IBM Worklight database or schema, and each must use its own IBM Worklight reports database or schema.
- If the application server is WebSphere Application Server Liberty Profile, each IBM Worklight project must use a different `contextroot` attribute and have a different base name for the .war file. But you can rename a .war file before you install it. The `id` attribute is not used in this case.
- If the application server is WebSphere Application Server Full Profile or WebSphere Application Server Network Deployment, each IBM Worklight project must use a different `id` attribute. Different deployments with the same `contextroot` attribute are possible, if they are deployed to separate sets of servers (for example, to different clusters or to different nodes).
- If the application server is Tomcat, each IBM Worklight project must use a different `contextroot` attribute. The `id` attribute is not used in this case. In addition, the versions of the JDBC drivers must be suitable for all declared data sources of the particular database type.

*Configuring WebSphere Application Server Network Deployment servers:*

Specific considerations when configuring WebSphere Application Server Network Deployment servers through Ant tasks are documented in this section.

To install a Worklight project into a set of WebSphere Application Server Network Deployment servers, run the `<configureapplicationserver>` Ant task on the computer where the deployment manager is running.

## Procedure

1. Specify a database type other than Apache Derby. IBM Worklight supports Apache Derby only in embedded mode, and this choice is incompatible with deployment through WebSphere Application Server Network Deployment.
2. As value of the profile attribute, specify the deployment manager profile.  
**Attention:** Do not specify an application server profile and then a single managed server. Doing so causes the deployment manager to overwrite the configuration of the server. This is true whether you install on the computer on which the deployment manager is running or on a different computer.
3. Specify an inner element, depending on where you want the Worklight project to be installed. The following table lists the available elements:

Table 192. Inner elements of <was> for network deployment

Element	Explanation
cell	Install the Worklight project into all application servers of the cell.
cluster	Install the Worklight project into all application servers of the specified cluster.
node	Install the Worklight project into all application servers of the specified node that are not in a cluster.
server	Install the Worklight project into the specified server, which is not in a cluster.

4. After starting the <configureapplicationserver> Ant task, restart the affected servers:
  - You must restart the servers that were running and on which the Worklight project application was 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.

## Results

The configuration has no effect outside the set of servers in the specified scope. The JDBC providers, JDBC data sources, and shared libraries are defined with the specified scope. The entities that have a cell-wide scope (the applications and, for DB2, the authentication alias) use the specified id attribute as a suffix in their name; it makes their name unique. So, you can install IBM Worklight Server in different configurations or even different versions of IBM Worklight Server, in different clusters of the same cell.

**Note:** Because the JDBC driver is installed only in the specified set of application servers, the **Test connection** button for the JDBC data sources in the WebSphere Application Server administration console of the deployment manager might not work.

## Adding a server to a cluster

When you add a server to a cluster that has a Worklight project installed on it, you must repeat some configuration manually. For each affected server, add a specific web container custom property:

1. Click **Servers > Server Types > Application Servers**, and select the server.
2. Click **Web Container Settings > Web container**.

3. Click **Custom properties**.
4. Click **New**.
5. Enter the property values listed in the following table:

*Table 193. Web container custom property values*

Property	Value
Name	com.ibm.ws.webcontainer.invokeFlushAfterService
Value	false
Description	See <a href="http://www.ibm.com/support/docview.wss?uid=swg1PM50111">http://www.ibm.com/support/docview.wss?uid=swg1PM50111</a> .

6. Click **OK**.
7. Click **Save**.

### Deploying a project WAR file and configuring the application server manually:

The procedure to manually deploy your app to an application server depends on the type of application server being configured, as detailed here.

These manual instructions assume that you are familiar with your application server.

**Note:** Using the Ant task to deploy the project WAR file and configure the application server is more reliable than installing and configuring manually, and should be used whenever possible.

#### *Configuring the WebSphere Liberty Profile manually:*

To configure WebSphere Application Server Liberty Profile manually, you must modify the `server.xml` file.

#### **About this task**

In addition to modifications for the databases that are described in “Creating and configuring the databases manually” on page 678, you must make the following modifications to the `server.xml` file.

**Note:** In the following procedure, when the example uses `worklight.war`, it must be the name of your Worklight project, for example, `myProject.war`.

#### **Procedure**

1. In the installation directory of Liberty, open the *user data* directory.  
If the installation directory for Liberty contains a `etc/server.env` file, and if this file defines a `WLP_USER_DIR` variable, then the *user data* directory is the value of this variable. Otherwise, it is the `usr` directory in the installation directory of Liberty.
2. Copy the IBM Worklight JAR file into the `shared/resources/lib/` directory that is in the *user data* directory.

If there is no `etc/server.env` file in the installation directory for Liberty, enter the following commands, according to your operating system:

- On UNIX and Linux:

```
mkdir -p WLP_DIR/usr/shared/resources/lib
cp WL_INSTALL_DIR/WorklightServer/worklight-jee-library.jar WLP_DIR/usr/shared/resources/lib
```

- On Windows:

```
mkdir WLP_DIR\usr\shared\resources\lib
copy /B WL_INSTALL_DIR\WorklightServer\worklight-jee-library.jar WLP_DIR\usr\shared\resources\lib
```

3. 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>
```

4. Add the following declarations in the `<server>` element, for the Worklight run time and the Worklight Console:

```
<!-- Declare the 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 Worklight Server application. -->
<webContainer invokeFlushAfterService="false"/>
```

This declaration installs the Worklight Server application with the context root `/worklight`. If you want to assign a different context root `/app_context`, start the declaration with one of the following code snippets:

```
<application id="app_context" name="app_context" location="worklight.war" type="war">
```

or:

```
<application id="worklight" name="worklight" location="worklight.war" context-root="/app_context">
```

For more information about the context root, see WebSphere Application Server documentation about Deploying application to the Liberty Profile.

*Configuring WebSphere Application Server manually:*

To configure WebSphere Application Server manually, you must configure variables, custom properties, and class loader policies.

### Before you begin

These instructions assume that you already have a standalone 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. 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`.
  4. Create the IBM Worklight JDBC data source and provider. See the instructions for the appropriate DBMS in "Creating and configuring the databases manually" on page 678
  5. 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 194. Web container custom property values

Property	Value
<b>Name</b>	com.ibm.ws.webcontainer.invokeFlushAfterService
<b>Value</b>	false
<b>Description</b>	See <a href="http://www.ibm.com/support/docview.wss?uid=swg1PM50111">http://www.ibm.com/support/docview.wss?uid=swg1PM50111</a>

- f. Click **OK**.
  - g. Click **Save**.
6. Install an IBM Worklight project WAR file.
 

**Note:** In the following procedure, when the example uses `worklight.war`, it should be the name of your Worklight project, for example, `myProject.war`.

    - 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**.
7. (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**.
    - 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.
  8. 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**.
  9. 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).

*Configuring Apache Tomcat manually:*

To configure Apache Tomcat manually, you must copy JAR and WAR files to Tomcat, add database drivers, edit the `server.xml` file, and then start Tomcat.

### Procedure

1. Copy the IBM Worklight 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 “Creating and configuring the databases manually” on page 678.
3. Copy the Worklight project WAR file to the Tomcat web application directory, `TOMCAT_HOME/webapps`, thereby renaming it according to the required context root. For example:
  - If the context root is `/worklight`, rename it to `worklight.war`.
  - If the context root is `/`, rename it to `ROOT.war`.
4. Edit `TOMCAT_HOME/conf/server.xml` to declare the context and properties of the Worklight application:

```
<!-- Declare the IBM Worklight Console application. -->  
<Context path="/worklight" docBase="worklight"/>
```

Here, make sure that the `path` and `docBase` attributes are both consistent with the WAR file name. That is, if the WAR file name is `worklight.war`, set the `path` to `"/worklight"` and the `docBase` to `"worklight"`. Whereas if the WAR file name is `ROOT.war`, set the `path` to `"` and the `docBase` to `"ROOT"`.

5. Start Tomcat.

## Configuration of IBM Worklight applications on the server

You can configure each IBM Worklight application by specifying a set of configuration parameters on the server side.

IBM Worklight application configuration parameters configure the database, push notifications, the use of SSL to secure communications between the server and the client application, and other settings.

When you develop an IBM Worklight application, you use the `worklight.properties` file to specify most of the configuration parameters. This file is in the `server/conf` folder in the project structure in Worklight Studio. You use the `worklight.properties` file during development to test a particular configuration. For example, if you want to use the analytics features during development, you might set the `wl.analytics.url` property to a valid URL in the `worklight.properties` file.

When your IBM Worklight project is built by Worklight Studio, the project WAR file that is created in the project `bin` folder contains the configuration that is specified in the `worklight.properties` file. The values for the parameters that are specified in the `worklight.properties` file then define the default configuration of your application.

When you deploy your project (your WAR file) to the production or test environment, your configuration is certain to be different from the development environment. It is easy to modify the configuration to fit the new environment because the project WAR file defines JNDI environment entries for each parameter that can be configured in a production environment. You leave the values in the `worklight.properties` file unchanged; instead, you specify the configuration during the deployment to the application server.

See “Configuring an IBM Worklight project in production by using JNDI environment entries” on page 723 to learn about the JNDI environment entries that you can specify in a production environment. Properties that are relevant only in development environments are not available as JNDI entries.

Within the `worklight.properties` file, you can define some application-specific configuration properties; for example, to configure the URL of a service that is called from the mobile application and the URL is different in production, development, and test environments. See “Declaring and using application-specific configuration properties” on page 723 to learn how to create such configuration properties.

### Configuring the Worklight Server location

You can configure the Worklight Server location by specifying configuration properties.

In production, you must configure your server location, since in most cases, production servers sit behind a reverse proxy; therefore, their machine IP address (which is the default value of `publicWorkLightHostname`) is not used for accessing them from the outside world.

To configure the Worklight Server location, set the values of the following properties:

*Table 195. Worklight Server location properties*

Property name	Description
<code>publicWorkLightHostname</code>	<p>The IP address or host name of the computer running IBM Worklight.</p> <p>If the Worklight Server is behind a reverse proxy, the value is the IP address or host name of the reverse proxy.</p> <p>This property must be identical for nodes within the same cluster.</p> <p>Default: IP address of current server.</p>

Table 195. Worklight Server location properties (continued)

Property name	Description
<b>publicWorkLightPort</b>	<p>The port for accessing the Worklight Server.</p> <p>If the Worklight Server is behind a reverse proxy, the value is the port for accessing the reverse proxy.</p> <p>This property must be identical for nodes within the same cluster.</p> <p>Default: 10080.</p> <p>The &lt;configureApplicationServer&gt; Ant task sets a default value that depends on the application server.</p>
<b>publicWorkLightProtocol</b>	<p>The protocol for accessing the Worklight Server.</p> <p>The valid values are HTTP and HTTPS. If the Worklight Server is behind a reverse proxy, the value is the protocol for accessing the reverse proxy.</p> <p>This property must be identical for nodes within the same cluster.</p> <p>Default: HTTP.</p> <p>The &lt;configureApplicationServer&gt; Ant task sets a default value that depends on the application server.</p>

For descriptions of other configuration properties, see “Configuring an IBM Worklight project in production by using JNDI environment entries” on page 723

For information about how to specify configuration properties, see “Configuration of IBM Worklight applications on the server” on page 714.

### IBM Worklight database setup for development mode

IBM Worklight uses defaults to access the IBM Worklight database. When you work in a development environment and use JDBC to connect to a database, you can use a set of property keys to change the settings.

#### Attention:

This method of declaring data sources is deprecated in a production environment and is only suitable when working in a development environment and using JDBC for database connections. To configure data sources when working in a production environment, see “Creating and configuring the databases manually” on page 678.

Property keys and values for JDBC-based properties

Property Key	Property Value
<b>wl.db.url</b>	JDBC path to IBM Worklight database.
<b>wl.db.username</b>	IBM Worklight database user name. Default: Worklight

Property keys and values for JDBC-based properties

Property Key	Property Value
<b>wl.db.password</b>	IBM Worklight database password. Default: Worklight
<b>wl.db.driver</b>	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
<b>wl.reports.db.url (*)</b>	JDBC path to IBM Worklight Reports database  Default: refers to IBM Worklight database
<b>wl.reports.db.username (*)</b>	IBM Worklight Reports database user name.  Default: refers to IBM Worklight database
<b>wl.reports.db.password (*)</b>	IBM Worklight Reports database password  Default: refers to IBM Worklight database

**Note:** (\*) By default all IBM Worklight report mechanisms use a single reports database. The reports database is set to be the same as the IBM Worklight database. For information about how this default setting can be changed, see “Using raw data reports” on page 861.

### Testing the Worklight Console login screen

When you work in a development environment, you can test the Worklight Console login screen by defining user credentials that are required to access it.

The user credential settings that you define to test the Worklight Console login screen can be encrypted as described in “Storing properties in encrypted format” on page 721.

Property Key	Property Value
<b>console.username</b>	Name of the user that can access the Console
<b>console.password</b>	User password

In addition to defining these two properties, it is also required to uncomment several sections in the authenticationConfig.xml file:

```
<!-- Uncomment the next element to protect the worklight console and the first section in security
<staticResources>
  <!--
    <resource id="worklightConsole" securityTest="WorklightConsole">
      <urlPatterns>/console*</urlPatterns>
    </resource>
  -->
</staticResources>
<securityTests>
```

```

<!--
<customSecurityTest name="WorklightConsole">
  <test realm="WorklightConsole" isInternalUserID="true"/>
</customSecurityTest>
...
...
-->
</securityTests>

```

Follow the instructions in the file about how to configure it. The authenticationConfig.xml file is located under *<Worklight Root Directory>\server\conf*. This file is described in “The authentication configuration file” on page 424.

**Note:** Setting user credentials in this way is not a suitable method for protecting Worklight Console access in a production environment. Use an alternative method such as LDAP instead.

### Push notification settings

When working with push notifications, you must use the correct proxy settings. For Android, you use GCM proxy settings, and for iOS, you use APNS proxy settings. SMS has its own set of proxy settings.

GCM proxy settings	Value
<code>push.gcm.proxy.enabled</code>	Shows whether Google GCM must be accessed through a proxy. Can be either true or false. The default is false.
<code>push.gcm.proxy.protocol</code>	GCM proxy protocol. Can be either http or https.
<code>push.gcm.proxy.host</code>	GCM proxy host.
<code>push.gcm.proxy.port</code>	GCM proxy port. Use -1 for the default port.
<code>push.gcm.proxy.user</code>	Proxy user name, if the proxy requires authentication. An empty user name means no authentication.
<code>push.gcm.proxy.password</code>	Proxy password, if the proxy requires authentication.

APNS proxy settings	Value
<code>push.apns.proxy.enabled</code>	Shows whether APNS must be accessed through a proxy. Can be either true or false. The default is false.
<code>push.apns.proxy.type</code>	APNS proxy type. Must be SOCKS.
<code>push.apns.proxy.host</code>	APNS proxy host.
<code>push.apns.proxy.port</code>	APNS proxy port.

SMS proxy settings	Value
<code>push.sms.proxy.enabled</code>	Can be either true or false. Use true to send SMS notifications through proxy.
<code>push.sms.proxy.protocol</code>	SMS proxy protocol. Can be either http or https.
<code>push.sms.proxy.host</code>	SMS proxy host name.

SMS proxy settings	Value
<code>push.sms.proxy.port</code>	SMS proxy port. Use -1 for the default port.
<code>push.sms.proxy.user</code>	Proxy user name, for authentication. An empty user name means no authentication.
<code>push.sms.proxy.password</code>	Proxy password, if the proxy requires authentication.

## Analytics

Analytics properties files contain the parameters for how IBM Worklight creates activity logs and sends them to a server for analysis.

You can modify how the Worklight Server forwards analytics data to the IWAP server by editing the following properties files.

Table 196.

Property Key	Property Value
<code>wl.analytics.logs.forward</code>	A Boolean value (true or false) that indicates whether to send messages that are logged by the Worklight Server to the IWAP server. If this value is true, all logs that are specified in <code>com.worklight</code> settings are forwarded to the IWAP server. The default value is true. This setting is only supported on IBM Worklight production servers. It is not supported on the IBM Worklight Studio development environment. (Optional.)
<code>wl.analytics.url</code>	The url to which analytics data are sent. Typically, this url is for the IWAP server that comes with the IBM Worklight installation. The URL is of the form <code>&lt;protocol&gt;://&lt;iwapURL&gt;/iwap/v1/events/_bulk</code> Example: <code>http://myServer.austin.ibm.com/iwap/v1/events/_bulk</code> <b>Note:</b> The <b>Analytics</b> tab is displayed in the IBM Worklight console only if the user enters a valid link for <code>wl.analytics.url</code> in the <code>worklight.properties</code> file. If the properties file contains no URL, the <b>Analytics</b> tab is not present.
<code>wl.analytics.username</code>	The basic authorized user name for the URL entered in <code>wl.analytics.url</code> . (Optional.)
<code>wl.analytics.password</code>	The basic authorized password for the URL entered in <code>wl.analytics.url</code> . (Optional.)
<code>wl.analytics.queues</code>	Sets the maximum number of queues that the Worklight Server can create to hold analytics data before it sends the data to the server. When this number of queues is reached, the Worklight Server rejects any new analytics data until the current data finishes processing. The default value is 20 (Optional).



Table 196. (continued)

Property Key	Property Value
<b>wl.analytics.queue.size</b>	Sets the number of individual analytics data that each queue can hold. The total number of analytics data that the server can hold at one time before it begins to drop data is (wl.analytics.queues * wl.analytics.queue.size). The default value is 10 (Optional).

## 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 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 property keys listed in the following table:

Table 197. JNDI environment entries for SSL certificate keystore

Property name	Description
<b>ssl.keystore.path</b>	Path to the keystore relative to the server folder in the Worklight Project; for example: conf/my-cert.jks.
<b>ssl.keystore.type</b>	Type of keystore file. Valid values are jks or pkcs12.
<b>ssl.keystore.password</b>	Password to the keystore file.

For descriptions of other IBM Worklight configuration properties, see “Configuring an IBM Worklight project in production by using JNDI environment entries” on page 723

For information about how to specify IBM Worklight configuration properties, see “Configuration of IBM Worklight applications on the server” on page 714.

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 364.

If you use SSL with mutual authentication between the 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 368.

**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.

## 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
<code>serverSessionTimeout</code>	Client inactivity timeout, after which the IBM Worklight session is invalidated.  Default is 10 minutes.
<code>bitly.username</code>	User name for accessing the bit.ly API for creating a shortened URL for mobile web apps through IBM Worklight Console.
<code>bitly.apikey</code>	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. The encryption facility included with IBM Worklight uses the 128-bit symmetric-key algorithm defined by the AES specification.

### 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=TYakEHRba3rIU7pNjxtDxoAdqijKIET7cy4mCr0iaEj0rY080DK00yqR`

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 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

You can encrypt IBM Worklight properties using the 128-bit symmetric-key algorithm defined by the AES specification. To encrypt 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.

The input file for the encryption is called `secret.properties` and contains the following data:

```
worklight_enc_password=abc123
certificate.password=certificatepwd123
wl.db.password=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=dR41nMQDaNEQyLQ17b2RmpdE99HKpqaSJ6mce0uJgaY\=
wl.db.password.enc=6boxojGZsUNTxw00GgI6dg\=\=
```

## Obsolete properties

Some properties are no longer required.

Table 198. Categories and list of obsolete properties

Category	Properties
Proxy settings	<code>proxy.enabled</code> , <code>proxy.nonProxyHosts</code> , <code>proxy.host</code> , <code>proxy.port</code> , <code>proxy.username</code> , <code>proxy.password</code> , <code>https.proxy.host</code> , <code>https.proxy.port</code>
Public resource server settings	<code>publicResourceServer.deployDestination</code> , <code>publicResourceServer.host</code> , <code>publicResourceServer.port</code> , <code>publicResourceServer.filesRootDir</code>
Environments	<code>environment.netvibes</code> , <code>environment.iphone</code> , <code>environment.embedded</code> , <code>environment.air</code> , <code>environment.android</code> , <code>environment.blackberry</code>
Certificate settings	<code>certificate.certificatesDirPath</code> , <code>certificate.keyStoreFilePath</code> , <code>certificate.keyAlias</code> , <code>certificate.keyStorePassword</code> , <code>certificate.keyAliasPassword</code> , <code>certificate.PFXFilePath</code> , <code>certificate.password</code> , <code>certificate.DERFilePath</code> , <code>certificate.P7BFilePath</code> , <code>vista.linux.oss1signcodeFilepath</code>
Push notification settings	<code>push.apns.certificatePassword</code> , <code>push.gcm.senderID</code> , <code>push.gcm.senderPassword</code>
Miscellaneous settings	<code>devmode</code> , <code>guid</code> , <code>wlclientTimeout</code> , <code>backend.request.timeout</code> , <code>reports.produceReports</code> , <code>wl.db.initialSize</code> , <code>wl.db.maxActive</code> , <code>wl.db.maxIdle</code> , <code>wl.db.testOnBorrow</code> , <code>wl.db.autodd1</code>
Tomcat settings	<code>local.bindAddress</code> , <code>local.httpPort</code>

Table 198. Categories and list of obsolete properties (continued)

Category	Properties
Console security settings	<code>console.username</code> , <code>console.password</code>

## Declaring and using application-specific configuration properties

Use the `${propertyName}` notation to reuse application-specific properties that are declared in the `worklight.properties` file.

You can declare application-specific properties in the `worklight.properties` file and reuse the value of those properties within the various IBM Worklight configuration files by using the `${propertyName}` notation. For example, you can insert the notation in an adapter (`adapter.xml`) file or in the authentication configuration (`authenticationConfig.xml`) file.

Here is an example for declaring a data source and reusing it in an adapter:

1. In the `worklight.properties` file, define a new (custom) property:  
`my.adapter.db.jndi.name=jdbc/MyAdapterDS`
2. You can then include a property declaration in the `adapter.xml` file:

```
<wl:adapter>
...
<connectivity>
  <connectionPolicy xsi:type="sql:SQLConnectionPolicy">
    <dataSourceJNDIName>
      ${my.adapter.db.jndi.name}
    </dataSourceJNDIName>
  </connectionPolicy>
...

```

Such properties are exposed as JNDI entries (see “Configuring an IBM Worklight project in production by using JNDI environment entries”) for the project WAR file. In this example, the JNDI name of the adapter data source becomes parametric and can be changed from the server configuration files.

## Configuring an IBM Worklight project in production by using JNDI environment entries

When you deploy an IBM Worklight project to a Worklight Server, the project’s WAR file can be configured with JNDI environment entries that cover all the properties that you can set in a production environment. You set the JNDI environment entries either by editing the deployer Ant task configuration XML file, or by configuring the server’s environment entries through the administration console (WebSphere Application Server) or the `server.xml` file (WebSphere Application Server Liberty Profile, Apache Tomcat).

### About this task

Many of the IBM Worklight configuration properties must have different values when the project is deployed to different environments. For example, the configuration properties that allow the specification of the Worklight Server public URL (that is, `publicWorkLightHostname`, `publicWorkLightPort`, and `publicWorkLightProtocol`) are properties that might be different when the Worklight project is deployed to a staging server or to a production server. You can configure the project WAR file through JNDI environment entries.

**Note:** Some of the properties are only relevant in a development environment and are not available as JNDI entries.

**Note:** To encrypt JNDI properties that are listed in the following table, as described in “Storing properties in encrypted format” on page 721, you must first define the property with the .enc suffix in the `worklight.properties` file that is packaged in the WAR file of the IBM Worklight project. It is then possible to override the encrypted value by using a JNDI property.

The following table lists the IBM Worklight properties that are always available as JNDI entries:

*Table 199. IBM Worklight properties available as JNDI entries*

Property name	Description
<code>publicWorkLightHostname</code>	<p>The IP address or host name of the computer that is running IBM Worklight.</p> <p>If the Worklight Server is behind a reverse proxy, the value is the IP address or host name of the reverse proxy.</p> <p>This property must be identical for nodes within the same cluster.</p> <p>Default: IP address of current server.</p>
<code>publicWorkLightPort</code>	<p>The port for accessing the Worklight Server.</p> <p>If the Worklight Server is behind a reverse proxy, the value is the port for accessing the reverse proxy.</p> <p>This property must be identical for nodes within the same cluster.</p> <p>Default: 10080.</p> <p>The <code>&lt;configureApplicationServer&gt;</code> Ant task sets a default value that depends on the application server.</p>
<code>publicWorkLightProtocol</code>	<p>The protocol for accessing the Worklight Server.</p> <p>The valid values are HTTP and HTTPS. If the Worklight Server is behind a reverse proxy, the value is the protocol for accessing the reverse proxy.</p> <p>This property must be identical for nodes within the same cluster.</p> <p>Default: HTTP.</p> <p>The <code>&lt;configureApplicationServer&gt;</code> Ant task sets a default value that depends on the application server.</p>
<code>serverSessionTimeout</code>	Idle session timeout in minutes. Default: 10.
<code>reports.exportRawData</code>	Whether reporting is activated (true or false). Default: false.
<code>push.gcm.proxy.host</code>	GCM proxy host. A negative value means default port.
<code>push.gcm.proxy.port</code>	GCM proxy port. Use -1 for the default port. Default: -1.

Table 199. IBM Worklight properties available as JNDI entries (continued)

Property name	Description
<code>push.gcm.proxy.protocol</code>	Either http or https.
<code>push.gcm.proxy.user</code>	Proxy user name, if the proxy requires authentication. Empty user name means no authentication.
<code>push.gcm.proxy.password</code>	Proxy password, if the proxy requires authentication.
<code>push.apns.proxy.enabled</code>	Indicates whether APNS must be accessed through a proxy. Default: false.
<code>push.sms.proxy.enabled</code>	Indicates whether push SMS proxy is enabled. Default: false.
<code>push.apns.proxy.host</code>	APNS proxy host.
<code>push.apns.proxy.port</code>	APNS proxy port.
<code>push.sms.proxy.protocol</code>	Push SMS proxy protocol.
<code>push.sms.proxy.host</code>	Push SMS proxy host.
<code>push.sms.proxy.port</code>	Push SMS proxy port.
<code>push.sms.proxy.user</code>	Push SMS proxy user.
<code>push.sms.proxy.password</code>	Push SMS proxy password.
<code>wl.ca.keystore.path</code>	Path to the keystore relative to the server folder in the Worklight Project; for example: <code>conf/my-cert.jks</code> .
<code>wl.ca.keystore.type</code>	Type of keystore file. Valid values are jks or pkcs12.
<code>wl.ca.keystore.password</code>	Password to the keystore file.
<code>wl.ca.key.alias</code>	Alias of the entry where the private key and certificate are stored in the keystore.
<code>wl.ca.key.alias.password</code>	Password to the alias in the keystore.
<code>ssl.keystore.path</code>	SSL certificate keystore location. Default: <code>conf/default.keystore</code> .
<code>ssl.keystore.type</code>	SSL certificate keystore type. Valid keystore types: jks or PKCS12. Default: jks.
<code>ssl.keystore.password</code>	SSL certificate keystore password. Default: <code>worklight</code> .
<code>cluster.data.synchronization.taskFrequencyInSeconds</code>	Applications and adapters cluster data synchronization interval. Default: 2.
<code>deployables.cleanup.taskFrequencyInSeconds</code>	Deployable folder cleanup task interval (in seconds). Default: 86400.
<code>sso.cleanup.taskFrequencyInSeconds</code>	Interval (seconds) for a cleanup task that cleans the database of orphaned and expired single-sign-on login contexts. Default: 5



Table 199. IBM Worklight properties available as JNDI entries (continued)

Property name	Description
<b>wl.analytics.logs.forward</b>	Boolean value (true or false) that indicates whether to send all com.worklight.* logs to the IWAP server. If this value is true, all logs that are specified in com.worklight settings are forwarded to the IWAP server. The default value is true. This setting is only supported on IBM Worklight production servers. It is not supported on the Worklight Studio development environment.
<b>wl.analytics.url</b>	URL to which analytics data are sent. Typically, this URL is for the IWAP server that comes with the IBM Worklight installation. The URL is of the form [protocol]://[iwapURL]/iwap/v1/events/[eventName]. Currently, the only event name that is used is <code>_bulk</code> . Example: <code>http://myServer.austin.ibm.com/iwap/v1/events/_bulk</code> .
<b>wl.analytics.username</b>	Basic authorized user name for the URL entered in <b>wl.analytics.url</b> . <b>Note:</b> The username is only required if you configured the IBM HTTP Server (IHS) in your IWAP installation to use SSL.
<b>wl.analytics.password</b>	Basic authorized password for the URL entered in <b>wl.analytics.url</b> . <b>Note:</b> The password is only required if you configured the IBM HTTP Server (IHS) in your IWAP installation to use SSL.
<b>wl.analytics.queues</b>	Sets the maximum number of queues that Worklight Server can create to hold analytics data before it sends the data to the server. When all the queues are full, Worklight Server quietly discards any new analytics data until the current data finishes processing. Default: 20.
<b>wl.analytics.queue.size</b>	Number of individual analytics data that each queue can hold. The number that represents the total amount of analytics data that the server can hold at one time before it begins to drop data is ( <b>wl.analytics.queues</b> * <b>wl.analytics.queue.size</b> ). Default: 10.

Custom user properties that are defined in the `worklight.properties` file are exposed as well.

The properties: **wl.db.\*** and **wl.reports.db.\*** are not available as JNDI environment entries because they are intended for use only during the development phase.

#### Configuring with the Ant task

When you deploy and configure the project with the Ant task (as described in “Deploying a project WAR file and configuring the application server with Ant



tasks" on page 694), it is possible to set values for Worklight configuration properties inside the <configureapplicationserver> tag. For example:

```
<configureapplicationserver shortcutsDir="${shortcuts.dir}">
  <property name="serverSessionTimeout" value="30"/>
  <property name="publicWorkLightHostname" value="www.example.com"/>
  <property name="publicWorkLightPort" value="80"/>
  <property name="publicWorkLightProtocol" value="http"/>
</configureapplicationserver>
```

### Manually configuring on the server

In some cases, when you do not want to or cannot redeploy the application, it is also possible to set values for Worklight configuration properties manually on the server configuration files (or console). This procedure is what the Ant task does behind the scenes. The manual configuration method is less recommended because in some cases (for example, when upgrading or redeploying), the application server might forget the configuration and the administrator must reconfigure it.

## Procedure

Complete the following tasks, depending on which application server is used:

- WebSphere Liberty Profile:

Insert the following declarations in the server.xml file:

```
<application id="worklight" name="worklight" location="worklight.war"
  type="war" context-root="/app_context_path">
</application>
<jndiEntry value="9080" jndiName="app_context_path/publicWorkLightPort"/>
<jndiEntry value="www.example.com" jndiName="app_context_path/publicWorkLightHostname"/>
```

The context path (in the previous example: **app\_context\_path**) connects between the JNDI entry and a specific IBM Worklight application. If multiple Worklight applications exist on the same server, the context path prefix enables you to define specific JNDI entries for each application. Typically, **app\_context\_path** is simply **"worklight"**.

- Apache Tomcat:

Insert the following declarations in the server.xml file:

```
<Context docBase="app_context_path" path="/app_context_path">
  <Environment name="publicWorkLightPort" override="false"
    type="java.lang.String" value="9080"/>
  <Environment name="publicWorkLightHostname" override="false"
    type="java.lang.String" value="www.example.com"/>
</Context>
```

**Note:** On Apache Tomcat, `override="false"` is mandatory.

With Apache Tomcat, the context path prefix is not needed because the JNDI entries are defined inside the <Context> element of an application.

- WebSphere Application Server:

1. In the administration console, navigate to **Applications > Application Types > WebSphere enterprise applications > Worklight > Environment entries for Web modules**
2. In the **Value** fields, enter values appropriate to your circumstances. See Figure 70 on page 728

[Enterprise Applications](#) > [Worklight](#) > Environment entries for Web modules

Environment entries for Web modules

Configure values for environment entries in web modules.

Web module	URI	Name	Type	Description	Value
Worklight	TestApp.war.WEB-INF/web.xml	publicWorkLightHostname	String	[REQUIRED] 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.	<input type="text"/>
Worklight	TestApp.war.WEB-INF/web.xml	publicWorkLightPort	String	[REQUIRED] 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.	<input type="text"/>
Worklight	TestApp.war.WEB-INF/web.xml	publicWorkLightProtocol	String	[REQUIRED] 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.	<input type="text"/>
Worklight	TestApp.war.WEB-INF/web.xml	serverSessionTimeout	String	[OPTIONAL] Idle session timeout in minutes Default is 10.	<input type="text"/>
Worklight	TestApp.war.WEB-INF/web.xml	reports.exportRawData	String	[OPTIONAL] Is reports active (true/false) default is false.	<input type="text"/>
Worklight	TestApp.war.WEB-INF/web.xml	push.gcm.proxy.enabled	String	[OPTIONAL] Shows whether Google GCM must be accessed through a proxy. Default is false.	<input type="text"/>
Worklight	TestApp.war.WEB-INF/web.xml	push.gcm.proxy.host	String	[OPTIONAL] GCM proxy host. Negative value means default port.	<input type="text"/>
Worklight	TestApp.war.WEB-INF/web.xml	push.gcm.proxy.port	String	[OPTIONAL] GCM proxy port. Use -1 for the default port. Default is -1.	<input type="text"/>
Worklight	TestApp.war.WEB-INF/web.xml	push.gcm.proxy.protocol	String	[OPTIONAL] Can be either http or https.	<input type="text"/>
Worklight	TestApp.war.WEB-INF/web.xml	push.gcm.proxy.user	String	[OPTIONAL] Proxy user name, if the proxy requires authentication. Empty user name means no authentication.	<input type="text"/>
Worklight	TestApp.war.WEB-INF/web.xml	push.gcm.proxy.password	String	[OPTIONAL] Proxy password, if the proxy requires authentication.	<input type="text"/>

Figure 70. Setting JNDI environment entries on WebSphere Application Server

**Related concepts:**

“Configuration of IBM Worklight applications on the server” on page 714  
 You can configure each IBM Worklight application by specifying a set of configuration parameters on the server side.

**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.

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 200. SMSConfig.xml elements and subelements

Element	Element Value
<b>gateway</b>	<p>Mandatory. The &lt;gateway&gt; element is the root element of the SMS gateway definition. It includes 6 attributes:</p> <ul style="list-style-type: none"> <li>• hostname</li> <li>• id</li> <li>• port</li> <li>• programName</li> <li>• toParamName</li> <li>• textParamName</li> </ul> <p>These attributes are described in Table 201</p>
<b>parameter</b>	<p>Optional. The &lt;parameter&gt; subelement is dependent on the SMS gateway. Each SMS gateway may have its own set of parameters. The number of &lt;parameter&gt; 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 &lt;parameter&gt; subelements.</p> <p>Each &lt;parameter&gt; subelement has the following attributes:</p> <ul style="list-style-type: none"> <li>• name</li> <li>• value</li> </ul>

Table 201. <gateway> element attributes

Attribute	Attribute Value
<b>hostname</b>	Mandatory. The host name of the configured SMS gateway.
<b>id</b>	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.
<b>port</b>	Optional. The port number of the SMS gateway. The default value is 80.
<b>programName</b>	<p>Optional. The name of the program that the SMS gateway expects. For example, if the SMS gateway expects the following URI:</p> <p>http://&lt;hostname&gt;:port/sendsms</p> <p>then <b>programName</b>="sendsms"</p>
<b>toParamName</b>	Optional. The name that is used by the SMS gateway to specify the destination mobile phone number. The default value is <i>to</i> . The destination mobile phone number is sent as a name-value pair when SMS notifications are sent; that is, <i>toParamName</i> = <i>destination mobile phone number</i> .

Table 201. <gateway> element attributes (continued)

Attribute	Attribute Value
textParamName	Optional. The name that is used by the SMS gateway to specify the SMS message text. The default value is <i>text</i> .

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/XMLSchema-instance" xsi:schemaLocation="http://www.worklight.com/sms/config sms-config.xsd">
  <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>
```

## Ant tasks for building and deploying applications and adapters

A set of Ant tasks is supplied with Worklight Server. The tasks in this section are used to build and deploy your applications, adapters, and projects.

IBM Worklight provides a set of Ant tasks that help you build and deploy adapters and applications to your IBM Worklight 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.

Apache Ant is required to run these tasks. The minimum supported version of Ant is listed in “System requirements for using IBM Worklight” on page 8.

### Building and deploying an application

The Ant tasks used for building and deploying an IBM Worklight application are documented in this section.

The following sections document examples of Ant XML files used to build and deploy applications.

#### Building an application

The Ant task for building an application has the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<project basedir="." default="target-name">
  <taskdef resource="com/worklight/ant/defaults.properties">
    <classpath>
      <pathelement location="WL_INSTALL_DIR/WorklightServer/worklight-ant.jar"/>
    </classpath>
  </taskdef>
  <target name="target-name">
    <app-builder
      worklightServerHost="http://server-address:port"
```

```

        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 worklightServerHost attribute (mandatory) specifies the full URL of your Worklight Server.
- 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 outputFolder attribute specifies the folder to which the resulting .wlap 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

**Note:** As a prerequisite step, before using this Ant task you must have deployed the application's corresponding Worklight project. For more information, see "Deploying an IBM Worklight project" on page 665.

The Ant task for deploying an application has the following structure:

```

<?xml version="1.0" encoding="UTF-8"?>
<project basedir="." default="target-name">
  <taskdef resource="com/worklight/ant/defaults.properties">
    <classpath>
      <pathelement location="WL_INSTALL_DIR/WorklightServer/worklight-ant.jar" />
    </classpath>
  </taskdef>
  <target name="target-name">
    <app-deployer deployable="app.wlap"
      worklightServerHost="http://server-address:port/contextroot"
      userName="username" password="password" />
  </target>
</project>

```

The <app-deployer> element has the following attributes:

- The worklightServerHost attribute (mandatory) specifies the full URL of your Worklight Server.
- The deployable attribute (mandatory) contains the .wlap file to deploy.
- The userName attribute (optional) contains the console user name, and is used if the console is protected with a form-based authenticator.
- The password attribute (optional) contains the console password, and is used if the console is protected with a form-based authenticator.

If you must deploy more than one .wlap file, add an <app-deployer> element for each file.

## Building and deploying adapters

The Ant tasks used for building and deploying IBM Worklight adapters are documented in this section.

The following sections document examples of Ant XML files used to build and deploy adapters.

### Building an adapter

The Ant task for building an adapter has the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<project basedir="." default="target-name">
  <taskdef resource="com/worklight/ant/defaults.properties">
    <classpath>
      <pathelement location="WL_INSTALL_DIR/WorklightServer/worklight-ant.jar"/>
    </classpath>
  </taskdef>
  <target name="target-name">
    <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

**Note:** As a prerequisite step, before using this Ant task you must have deployed the adapter's corresponding Worklight project. For more information, see "Deploying an IBM Worklight project" on page 665.

The Ant task for deploying an adapter has the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<project basedir="." default="target-name">
  <taskdef resource="com/worklight/ant/defaults.properties">
    <classpath>
      <pathelement location="WL_INSTALL_DIR/WorklightServer/worklight-ant.jar" />
    </classpath>
  </taskdef>
  <target name="target-name">
    <adapter-deployer deployable="myAdapter.adapter"
      worklightserverhost="http://server-address:port/contextroot"
      userName="username" password="password" />
  </target>
</project>
```

The <adapter-deployer> element has the following attributes:



- The `worklightserverhost` attribute (mandatory) specifies the full URL of your Worklight Server.
- The `deployable` attribute (mandatory) specifies the `.adapter` file to deploy.
- The `userName` attribute (optional) contains the console user name, and is used if the console is protected with a form-based authenticator.
- The `password` attribute (optional) contains the console password, and is used if the console is protected with a form-based authenticator.

If you must deploy more than one `.adapter` file, add an `<adapter-deployer>` element for each file.

## Deploying applications and adapters to Worklight Server

You can deploy customer-specific content (apps and adapters) only after the project WAR file is deployed and the server is started.

### About this task

Customer-specific content includes applications that must be served by Worklight Server and their underlying integration adapters. You can create apps and adapters by building them in Worklight Studio, or with the Ant tasks provided with IBM Worklight to build them. The result of the build action is files with extension `.wlap` and `.adapter` respectively.

There are two ways to deploy applications and adapters to Worklight Console:

- Use Ant tasks provided with IBM Worklight, and described in “Ant tasks for building and deploying applications and adapters” on page 730 and “Ant tasks for deploying a project WAR file and configuring an application server” on page 666.
- Use Worklight Console to manually deploy apps and adapters as described below.

Worklight Console opens in a “Catalog” page that enables you to work with apps and adapters.

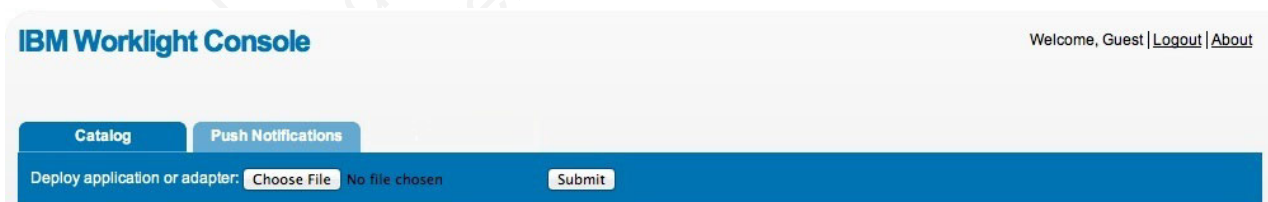


Figure 71. IBM Worklight Catalog

To deploy an adapter:

### 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.



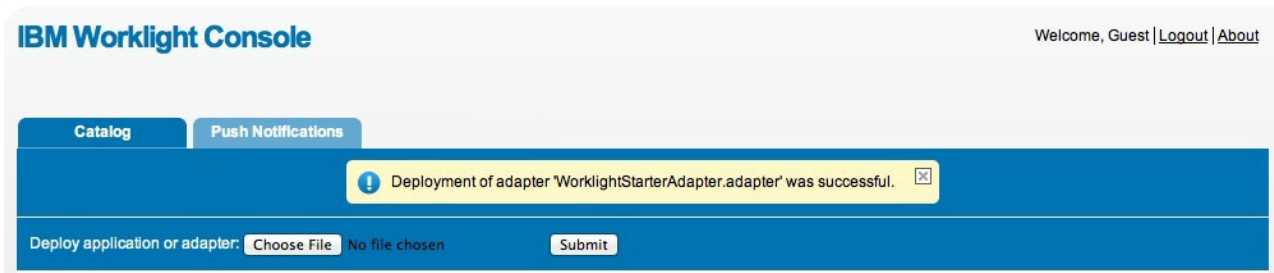


Figure 72. IBM Worklight adapter deployment success or failure message

As a result, the details of the deployed adapter are added to the catalog:

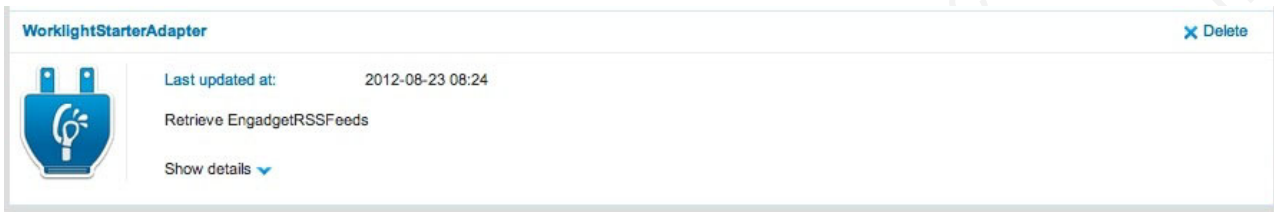


Figure 73. Deployed adapter details

3. Click **Show details** to view connectivity details for the adapter and the list of procedures it exposes.

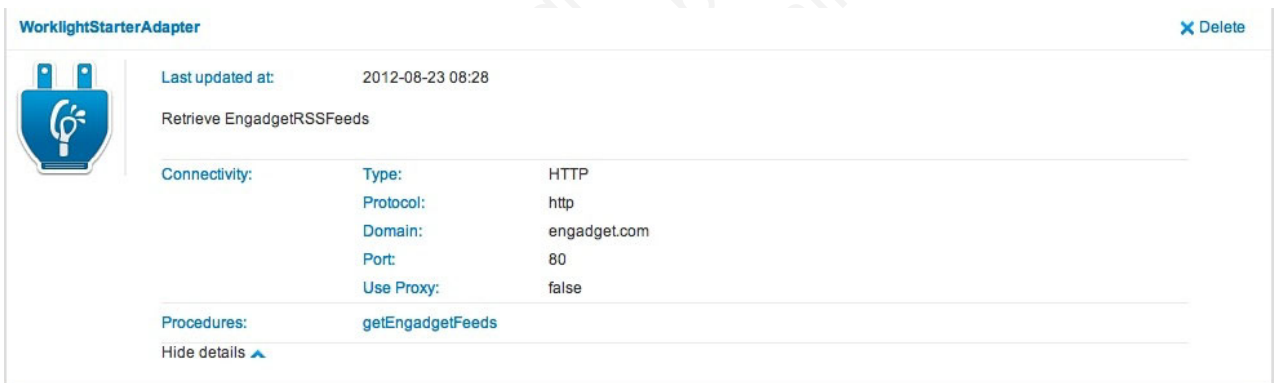


Figure 74. Adapter connectivity details

4. Repeat steps 1 – 3 for each adapter.
- To deploy an application:
5. In the catalog page, click **Browse**, then navigate to the `.wlap` file and select it.



Figure 75. Browsing the catalog page to find the `.wlap` file

6. Click **Submit**. A message is displayed indicating whether the deployment action succeeded or failed.

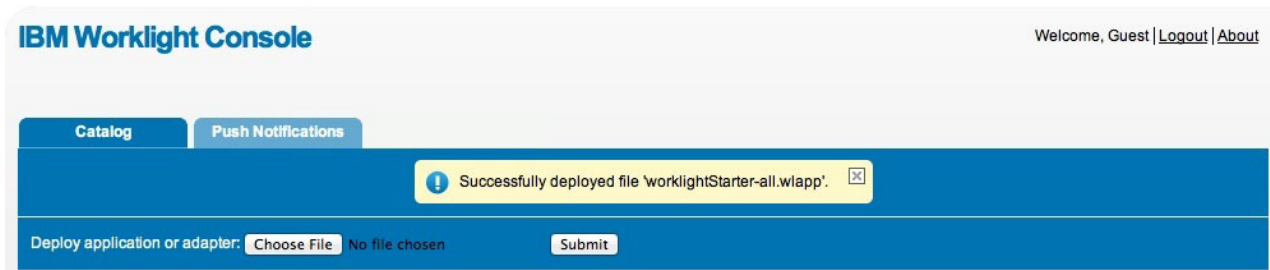


Figure 76. Deployment success or failure message

As a result, the details of the deployed application are added to the catalog.

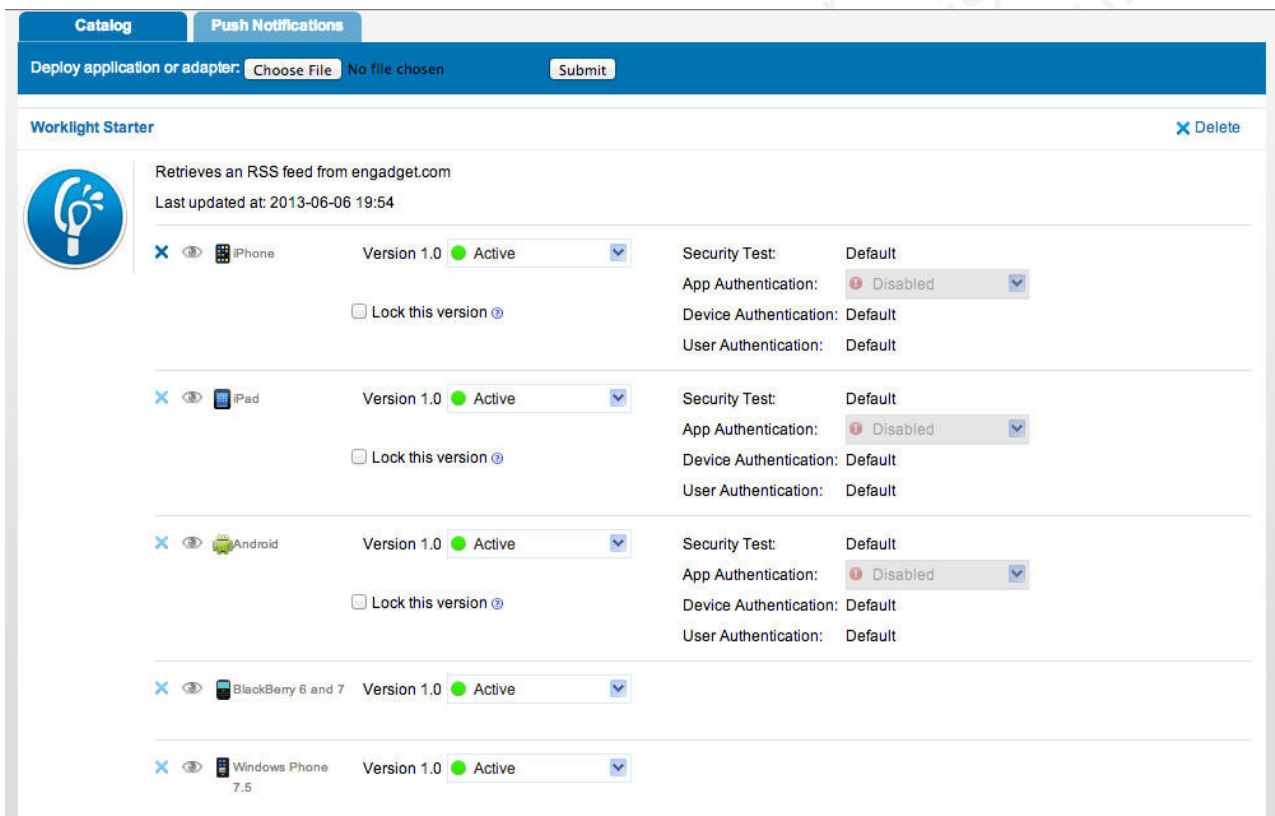


Figure 77. Details of the deployed application

7. Repeat steps 1 and 2 for each app.

## Administering adapters and apps in 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 console:

## Procedure

1. Open a browser and enter the following URL: `http://my-host:10080/my-context-root/console` 10080 is the default port. You might be using a different value.
2. If your 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.



## 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.

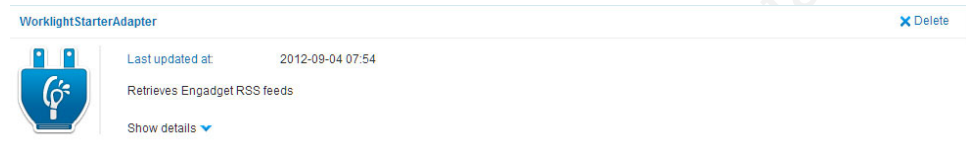
## Deploying adapters

Deploy an adapter from the console.

### Procedure

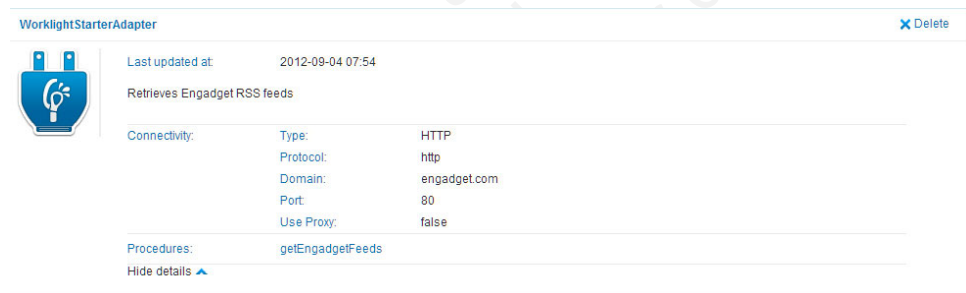
To deploy an adapter:

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.



## High availability

High availability is provided through clustering, the ability to provide multiple Worklight Servers acting together.

Multiple Worklight Servers enable horizontal scaling of the software as well as the prevention of a single point of failure.

### Clustering

The 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 project 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 project 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.

### 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 a 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 a 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 Worklight Servers.
2. Install the Worklight Server.
3. Apply the project WAR.

## Firewalls

Firewalls can be configured at various layers of the IBM Worklight architecture.

Firewalls in front of a 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.

## 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 202. 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.

#### 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.

---

## 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 IBM Worklight in combination with IBM 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 IBM Worklight runtime and artifacts support for IBM 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.

## Installing 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-6.0.0.0.tgz` file in the `worklight_pattern_6.0.0.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.
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.

## Installing custom IBM Worklight database workload standards

You need to install custom workload standards for the IBM Worklight runtime database and the IBM Worklight reports database.

### Before you begin

Extract the `WLRTDB.zip` and `WLRPTDB.zip` files from the `worklight_pattern_6.0.0.offering.zip` file.

### Procedure

1. Log in to IBM PureApplication System with an account that has permission to create database workload standards.
2. In the Workload Console, navigate to **Catalog > Database Workload Standards**.
3. From the toolbar, click the plus (+) button. The Database Workload Standards window opens.
4. In the **Name** field, enter a name; for example, `WL_DB`.
5. From the **Workload type** list, select **Departmental Transactional**.
6. In the **Upload file (.zip)** field, select the `WLRTDB.zip` file you extracted from the `worklight_pattern_6.0.0.offering.zip` file.
  - a. Click the **Browse** button, navigate to the folder into which you extracted the `WLRTDB.zip` file, and then select the `WLRTDB.zip` file.
7. Click the **Save** button to save your custom runtime database workload standard.

8. Repeat the previous steps to upload the WLRPTDB.zip file to create the custom reports database workload standard.

### **What to do next**

You use the installed database workload standards in the process of creating an IBM Mobile Application Platform Pattern (see “Creating an IBM Mobile Application Platform Pattern” on page 743).

### **Installation of IBM Worklight PureApplication System Extension for Worklight Studio**

IBM Worklight PureApplication System Extension is included with Worklight Studio in both the IBM Worklight Enterprise Edition and IBM Worklight Consumer Edition. When Worklight Studio is installed in the Eclipse development environment, the IBM Worklight PureApplication System Extension is also installed.

For more information about installation and configuration of Worklight Studio, see Chapter 4, “Installing and configuring,” on page 35.

## **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 a Worklight Server, a runtime database, and a reports database, and uploading applications and adapters.

### Before you begin

This procedure involves uploading certain artifacts to IBM PureApplication System such as the Worklight Server. Before you begin, ensure that the artifacts are available for upload.

### Procedure

1. Create a Worklight Server.
  - a. If necessary, use the Worklight Studio PureApplication System Extension or the command line interface to package up the Worklight Server into an EAR file. For further information, see the following topics:
    - “Deploying an IBM Worklight project to PureApplication System” on page 747
    - “Building an IBM Worklight virtual application” on page 748
  - 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 203. Worklight Server component properties

Property	Description
Name	Name for the Worklight Server.
EAR file	IBM Worklight EAR file that contains the Worklight Server package to be uploaded.

2. Create IBM Worklight runtime and reports databases.
  - 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 to define the runtime database:

Table 204. Worklight runtime database component properties

Property	Description
Name	Name for the Worklight runtime database component; for example, WL Runtime DB.
Database name	Name for the runtime database; for example, WLRTIME.
Source	From the <b>Source</b> list, select <b>Apply a database workload standard</b> , and then click the database workload standard created for the IBM Worklight runtime database (see “Installing custom IBM Worklight database workload standards” on page 741).

- c. Repeat step 2a to create a reports database.

- d. Supply the following information in the fields provided to define the reports database:

Table 205. Worklight reports database component properties

Property	Description
<b>Name</b>	Name for the Worklight reports database component; for example, WL Reports DB.
<b>Database name</b>	Name for the runtime database; for example, WLRPT.
<b>Source</b>	From the <b>Source</b> list, select <b>Apply a database workload standard</b> , and then click the database workload standard created for the IBM Worklight reports database (see “Installing custom IBM Worklight database workload standards” on page 741).

3. Configure connectivity for the runtime and reports databases.
  - a. Drag a connection from the Worklight Server component to the runtime database component.
  - b. In the **Resource References of Data Source** field, select **jdbc/WorklightDS** for the IBM Worklight runtime database.
  - c. Drag a connection from the Worklight Server component to the reports database component.
  - d. In the **Resource References of Data Source** field, select **jdbc/WorklightReportsDS** for the IBM Worklight reports database.
4. Configure the Worklight Server.
  - a. From the Assets list, expand **Worklight Components**, and then drag and drop a Worklight Configuration component onto the canvas.
  - b. Create a link from the Worklight Server component to the Worklight configuration component.
  - c. Supply the following information in the fields provided:

Table 206. Worklight configuration component properties

Property	Description
<b>Name</b>	Name for the Worklight configuration component.
<b>Worklight Console Protection</b>	Select this check box to enable security protection for the Worklight console. Clear the check box to disable security protection.
<b>Worklight Console Username</b>	User name for Worklight console protection.
<b>Worklight Console Password</b>	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 207. 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 208. 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 Worklight Server component, and from the Worklight adapter component to the 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 IBM Worklight application. For more information, see “Integration with Tivoli Directory Server” on page 748

### 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 IBM 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 a User Registry Tivoli Directory Server component onto the canvas.
3. Supply the following information in the fields provided:

Table 209. 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 <b>baseDN</b> .
User filter	Effective only when the LDAP login module has the parameter <b>userFilter</b> .

Table 209. Tivoli Directory Server component properties (continued)

Property	Description
Group filter	Effective only when the LDAP login module has the parameter <b>groupFilter</b> .

### 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 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 Worklight Server component to the Connect Out component.

### Performing operations on running IBM Worklight Virtual Application Pattern instances

Use the PureApplication System Workload Console to perform management tasks on a running IBM Worklight Virtual Application Pattern 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**.
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 210. Operations on Virtual Application Pattern instances

Operation	Description
Worklight Application/Adapter	Install or update Worklight applications and adapters. Supported file types: <b>*.wlap</b> , <b>*.adapter</b> , <b>*.zip</b> .
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.

**Note:** IBM Worklight PureApplication System Extension for Worklight Studio copies the PureApplication configuration into the WorklightServerConfig folder under your eclipse workspace.

### Setting up PureApplication System preferences in 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:

Table 211. 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 Worklight Server location” on page 715.

### Procedure

1. In Eclipse, navigate to the Project Explorer view.
2. Right-click your 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 Worklight project, and then click **Fetch Worklight Console URL on IPAS**. A window opens displaying the Worklight Console URL.

### Integration with Tivoli Directory Server

To use Tivoli Directory Server as a user registry on PureApplication System for your IBM 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 IBM 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 IBM 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 an IBM 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_6.0.0.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="{wlpProjectDestDir}"
    artifactsFolder="{artifactsFolder}"
    elbHost="{elbHost}"/>
</target>
```

The following table describes the attributes.

Table 212. Ant task build attributes

Attributes	Description
<b>worklightWar</b>	Required. The Worklight Console WAR file including the full file path.
<b>destinationFolder</b>	Optional. Default value: <code>{projectfolder}/bin</code> .
<b>artifactsFolder</b>	Optional. Folder for adapters and applications.
<b>elbHost</b>	Optional. Host name for elastic load balancer.
<b>createVAPFlag</b>	Optional. Whether to generate a VAP .zip file. Default value: <code>true</code> .
<b>isConnectNewTDS</b>	Optional. Whether to connect a new Tivoli Directory Service.
<b>ldiffFile</b>	Optional. When <b>isConnectNewTDS</b> is true, you must set this attribute.
<b>ipasModel</b>	Optional. Default value is <code>W1500</code> ; in this case, it works on Intel. You can set its value to <code>W1700</code> ; in this case, IBM PureApplication System runs on Power® system.
<b>ipasHost</b>	Optional. The URL of PureApplication System. Required when <b>createVAPFlag</b> is true.
<b>username</b>	Optional. The user name that is required to access the PureApplication System console. Required when <b>createVAPFlag</b> is true.
<b>password</b>	Optional. The password that is required to access the PureApplication System console. Required when <b>createVAPFlag</b> is true.

## Deploying an IBM Worklight virtual application

You can use an Ant task to deploy an IBM 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_6.0.0.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 213. Ant task deployment attributes

Attributes	Description
<b>vapZipFile</b>	Required. Path to the zip file built by vap-builder.
<b>ipasHost</b>	Required. The URL of IBM PureApplication System.
<b>username</b>	Required. Username to access PureApplication System console.
<b>password</b>	Required. Password to access PureApplication System console.
<b>profileName</b>	Required. Profile name for deploying VAP.
<b>cloudGroupName</b>	Required. Cloud group name for deploying VAP.
<b>ipGroupName</b>	Required. IP group name for deploying VAP.
<b>priority</b>	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.

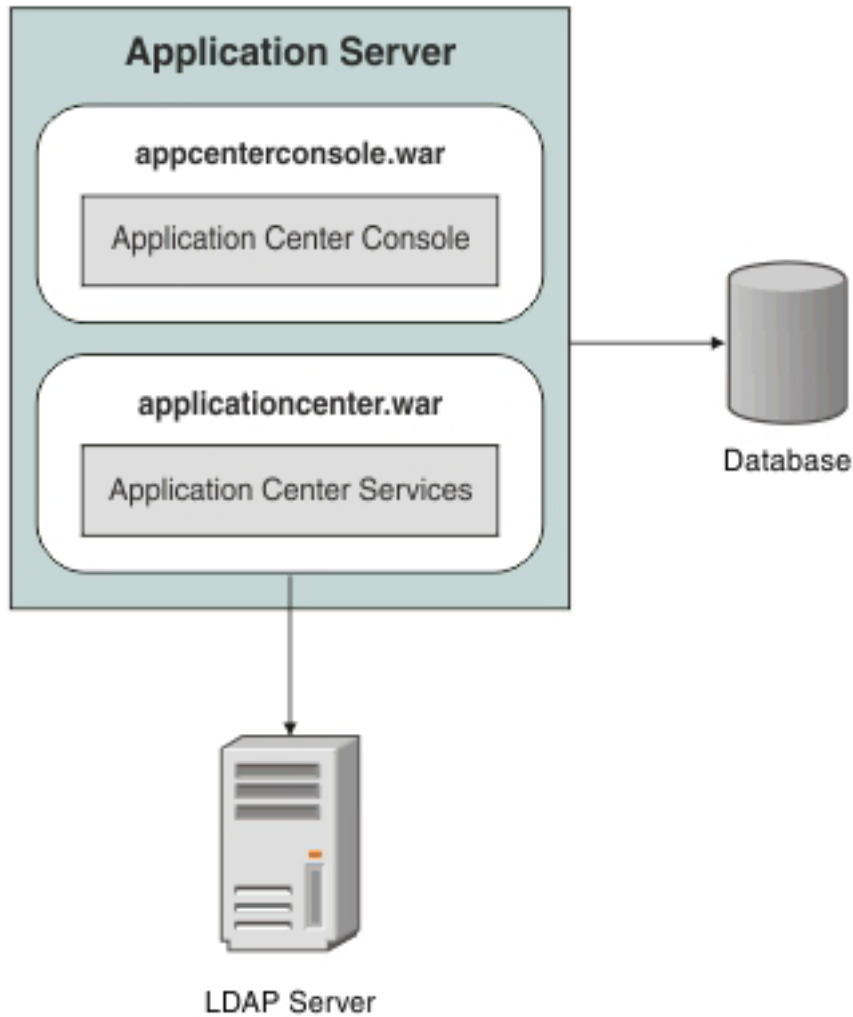


Figure 78. Typical operational model of the Application Center

**Related concepts:**

“Application Center” on page 769

Learn about the Application Center: what it is for, the different components and features, and how to use the console and the client.

“Configuring the Application Center after installation” on page 118

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 121

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.

### Before you begin

Install the Application Center. The Application Center is part of Worklight Server. You can install it through IBM Installation Manager or manually. See “Installing Worklight Server” on page 41.

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



---

## Chapter 9. Administering IBM Worklight applications

Run and maintain IBM Worklight applications in production.

---

### Administering IBM Worklight applications with Worklight Console

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.

### Direct updates of app versions to mobile devices

The 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 Worklight Server without changing its version, the 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.

Direct Update works only if the client-side artifacts of the application are built with the same version of Worklight Studio as the Worklight Server that delivers the direct updates. See "Migrate your Worklight projects" on page 192 for instructions about how to reenable direct update after an upgrade of the Worklight Server.

When the app connects to the Worklight Server, it starts downloading the newly deployed resources, as shown in the following figures.

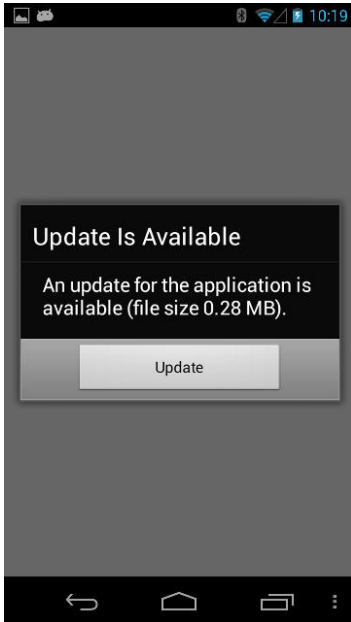


Figure 79. Update notice from Android

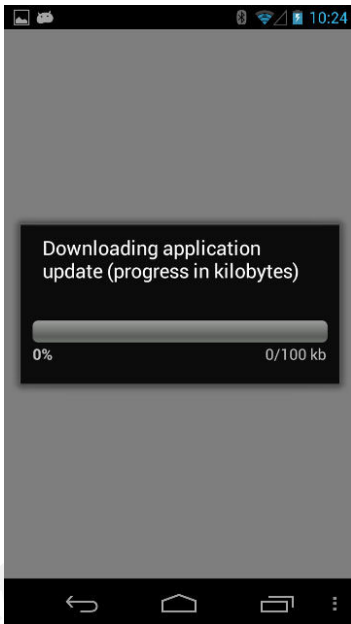


Figure 80. Downloading newly deployed resources to Android

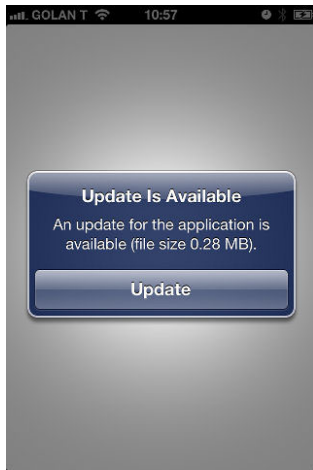


Figure 81. Update notice from iOS



Figure 82. 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.

When you redeploy a desktop app with a new version, the Worklight Server automatically pushes the app to the user's desktop. When the desktop app connects to the 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 only available for Adobe AIR applications.

## Locking an application

You can prevent developers or administrators from mistakenly updating an application, by locking it in Worklight Console.

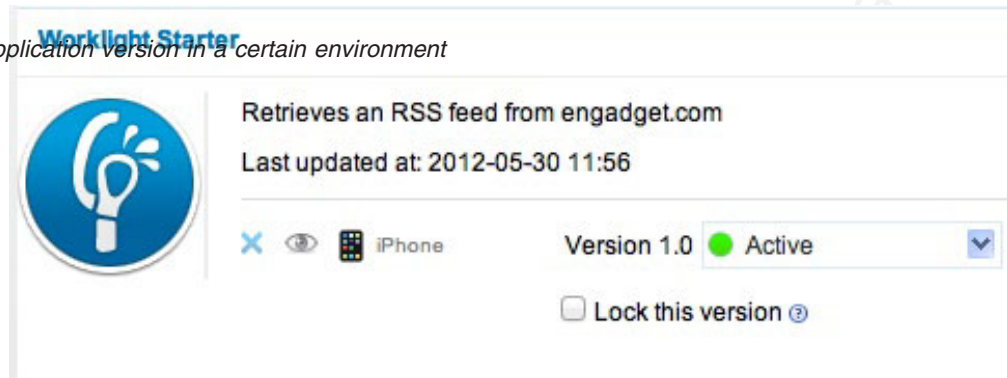
## About this task

You can lock applications for iPhone, iPad, and Android.

## 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 83. Locking an application version in a certain environment



## 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.

### Before you begin

If you need to use the Remote Disable feature with servers and clusters that experience heavy loads, consider enabling the Remote Disable cache. Enabling the cache can improve performance by reducing how frequently the database is checked to see if an app has been remotely disabled. By default, the cache is disabled. To enable and configure the cache, add the following lines to the Worklight project's `worklight.properties` file:

- `wl.remoteDisable.cache.enabled=true`
- `wl.remoteDisable.cache.refreshIntervalInSeconds=1`

The refresh interval determines how long (measured in seconds) values are kept in the cache before they are refreshed from the database. If you increase the interval, performance is improved as a result of fewer connections being made to the database, but you increase the duration before the remote disable state comes into effect. For example, if your infrastructure contains a cluster of four Worklight Servers and you set `wl.remoteDisable.cache.refreshIntervalInSeconds=1`, the database is accessed 4 times per second to check the remote disable state.

## 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.

## 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 84. Denying access to older application versions

The screenshot shows the Worklight Starter interface. At the top, it says "Worklight Starter" and "Retrieves an RSS feed from engadget.com". Below this, there's a header "Last deployed at: 2013-09-17 12:52". The main area lists three application versions for different platforms:

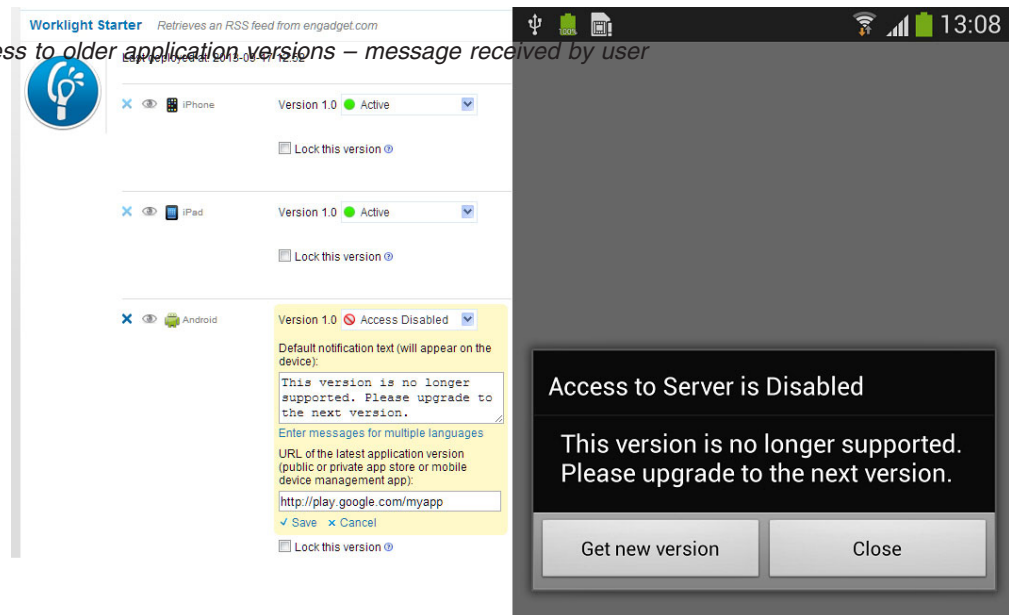
- iPhone:** Version 1.0, status Active. There is a checkbox for "Lock this version" which is unchecked.
- iPad:** Version 1.0, status Active. There is a checkbox for "Lock this version" which is unchecked.
- Android:** Version 1.0, status Access Disabled. This entry is highlighted in yellow. A dialog box is open for this version, showing a custom notification message: "This version is no longer supported. Please upgrade to the next version." and a URL for the latest version: "http://play.google.com/myapp". There is a checkbox for "Lock this version" which is unchecked.

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.

Figure 85. 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 Worklight Console in multiple languages" on page 761

You can set the deny and notification messages from 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.

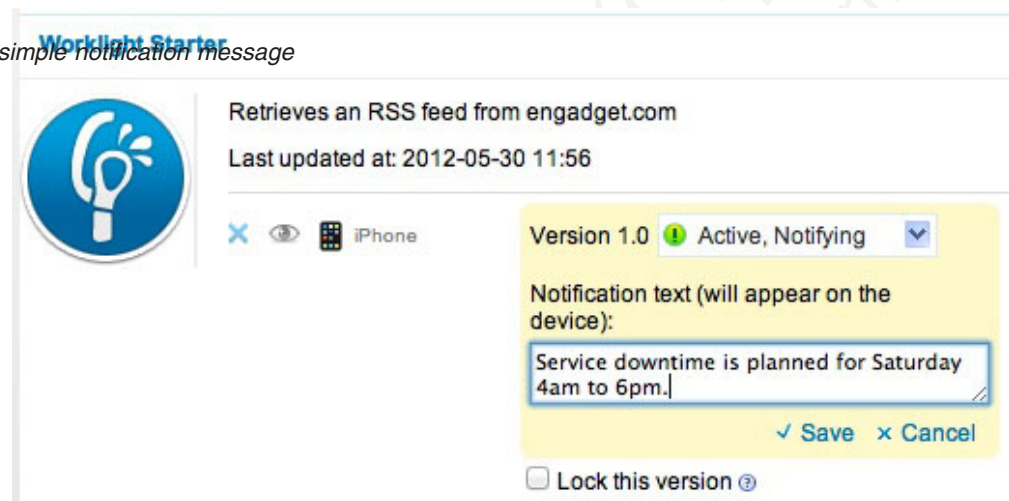
### About this task

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 86. Displaying a simple notification message



### 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 Worklight Console in multiple languages”  
You can set the deny and notification messages from 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 Worklight Console in multiple languages

You can set the deny and notification messages from 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 Worklight Console, select the status **Active**, **Notifying**, or **Disabled** in the list of application rules.
2. Click **Enter messages for multiple languages**.

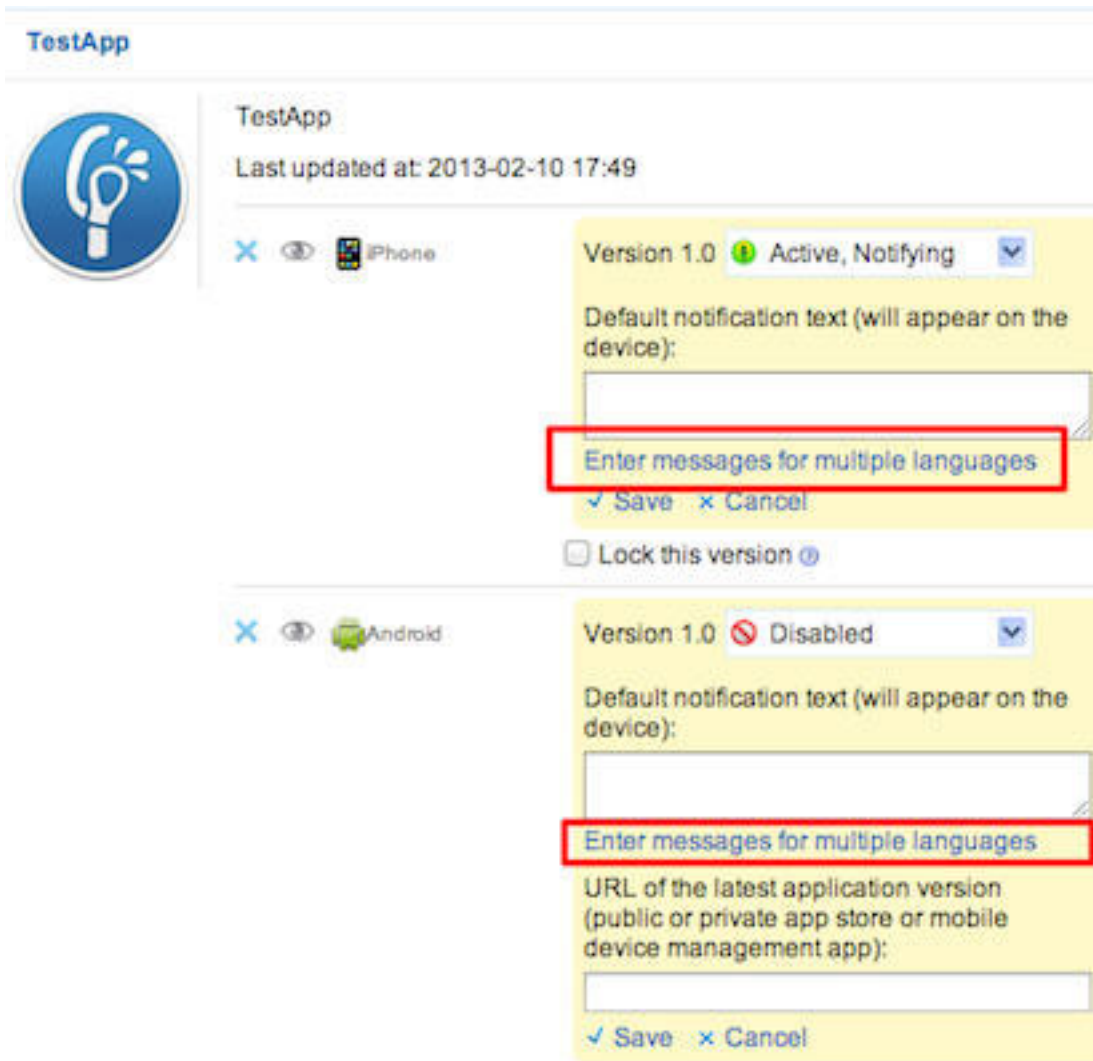


Figure 87. Defining the status of application rules in Worklight Console

3. In the **Messages for multiple languages** window that opens, notice that you can upload a CSV file.

Figure 88. 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 חסומה
ja,あなたのアプリケーションが無効になっていた
```

Figure 89. 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 90. View of your uploaded languages, and the default message with its translation

**Messages for multiple languages**

**Default Message**

your application is disabled (default)

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	
en	English
en-US	English United States
en-GB	English United Kingdom
ru	Russian
fr	French

Upload CSV

Clear

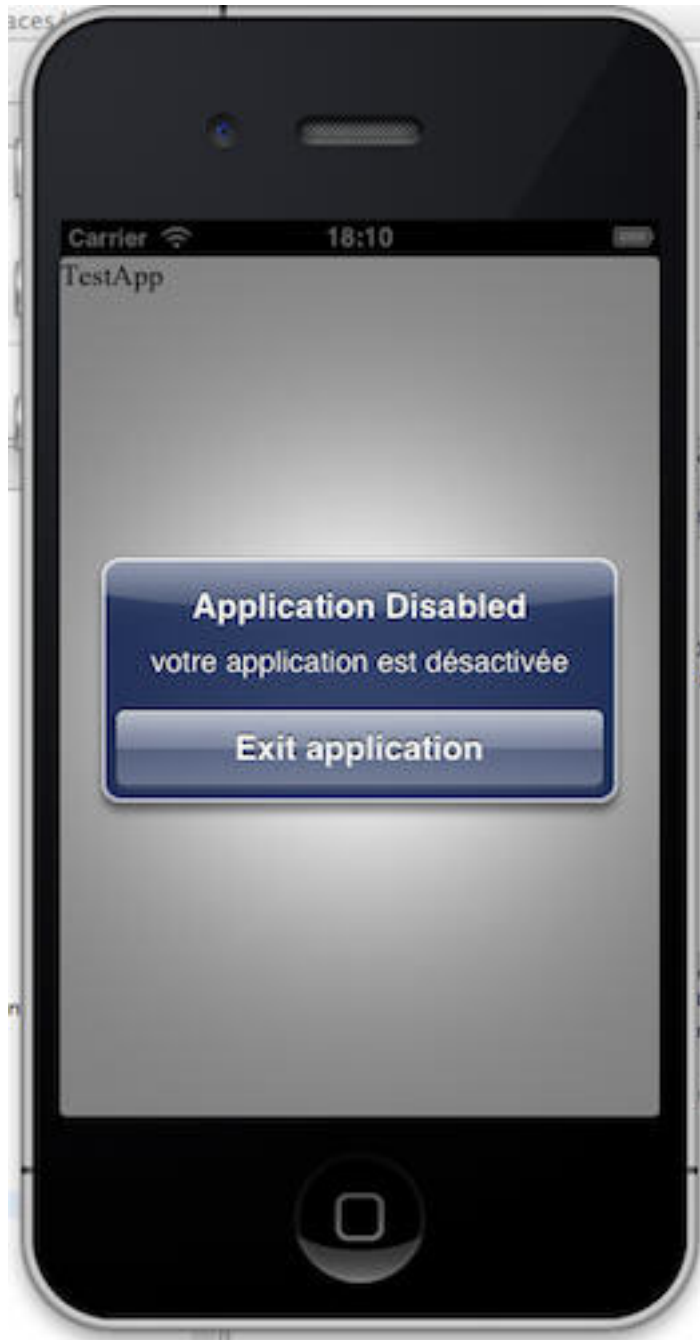
**Translation**

Your application is disabled in GB

Save Cancel

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 console.

**Note:** If you modified the default message, then the new default message shows.



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 91. Application Disabled message*

## Controlling authenticity testing for an app

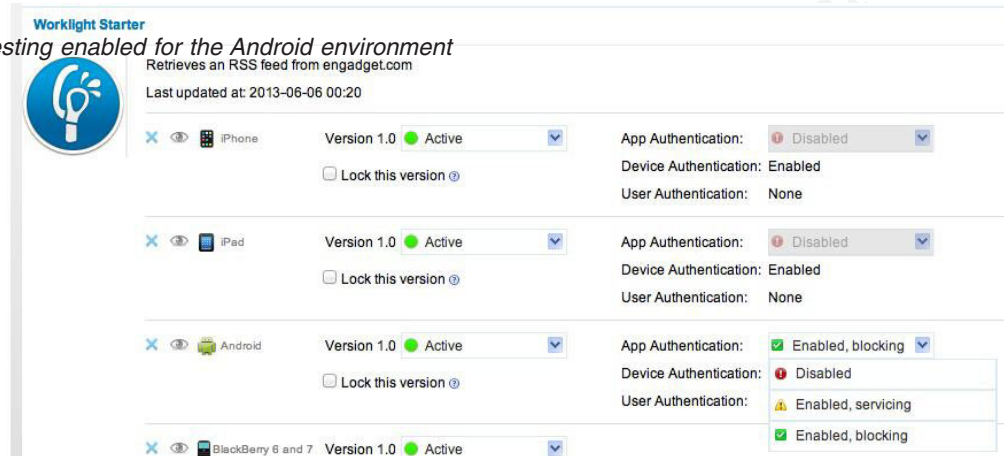
You can control authenticity testing for apps that connect to the Worklight Server.

When an app first connects to the 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 “IBM Worklight security framework” on page 417 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 92. Authenticity testing enabled for the Android environment



The menu has three options:

- **Disabled** – the Worklight Server does not test the authenticity of the app (despite the developer's settings).
- **Enabled, servicing** – the Worklight Server tests the authenticity of the app. If the app fails the test, the Worklight Server outputs an information message to the log but services the app.
- **Enabled, locking** – the Worklight Server tests the authenticity of the app. If the app fails the test, the 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.

## Administering push notifications with the Worklight Console

The Push Notifications page in the Worklight Console provides you with a quick view of the various entities in the push notification chain.

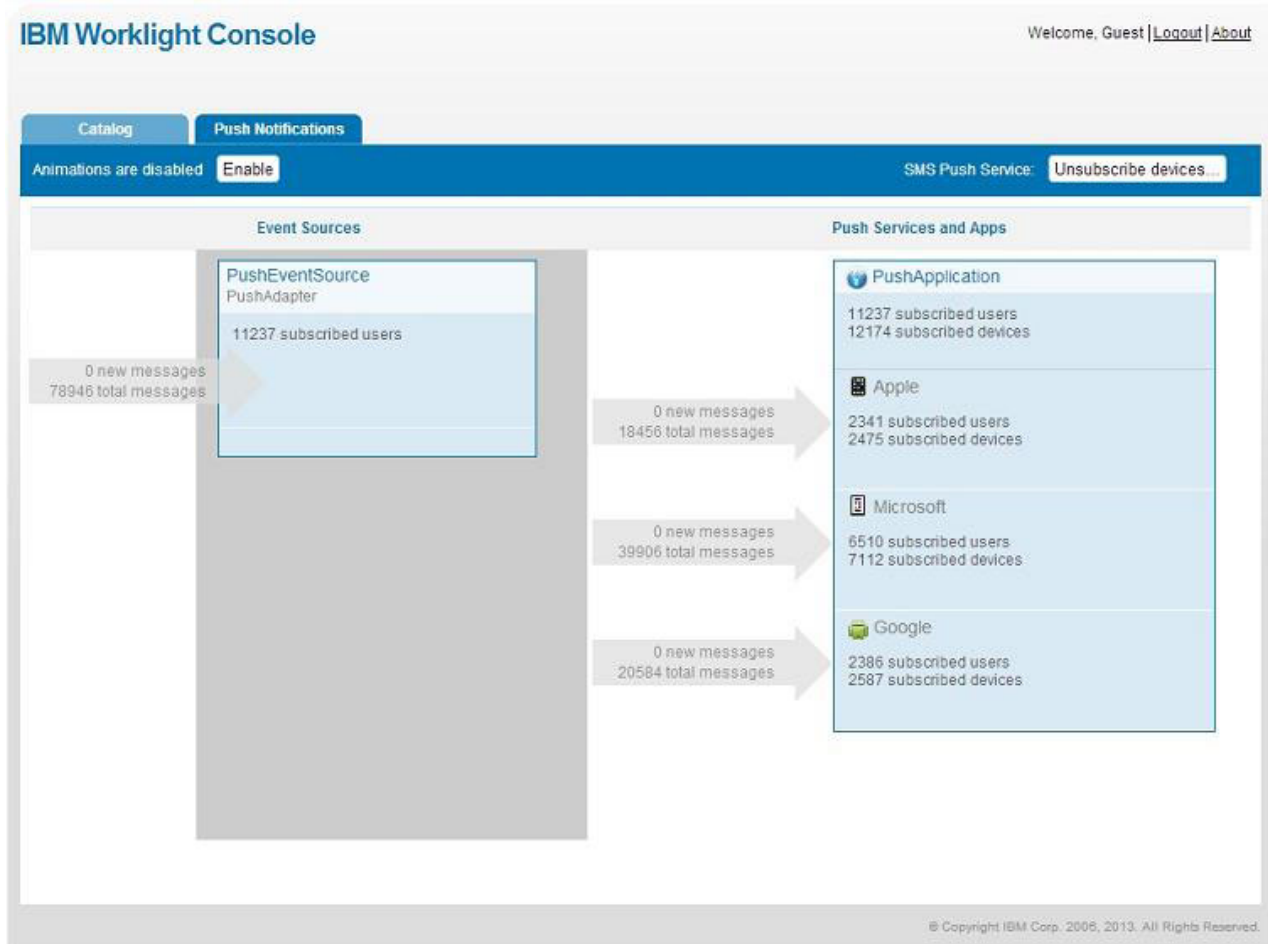


Figure 93. Push notifications in the Worklight Console

The left column displays the list of data sources that are configured in your 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.



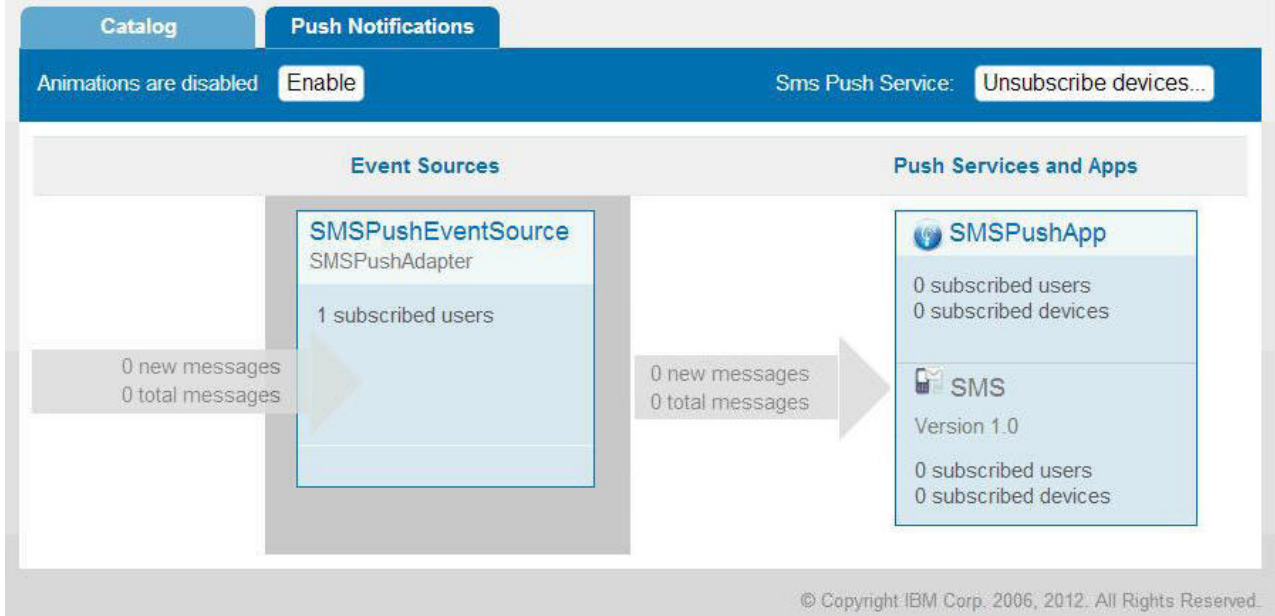


Figure 94. SMS push notifications in the Worklight Console

Administrators can forcibly unsubscribe existing SMS subscriptions by clicking **Unsubscribe devices**. The Unsubscribe SMS Devices window opens, and administrators can then enter 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 Worklight Console unsubscribes both subscriptions.



Figure 95. Unsubscribe existing SMS subscriptions



---

## Application Center

Learn about the Application Center: what it is for, the different components and features, and how to use the console and the client.

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.)

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.

### 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 782.

### 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 805.

The following figure shows an overview of the architecture.

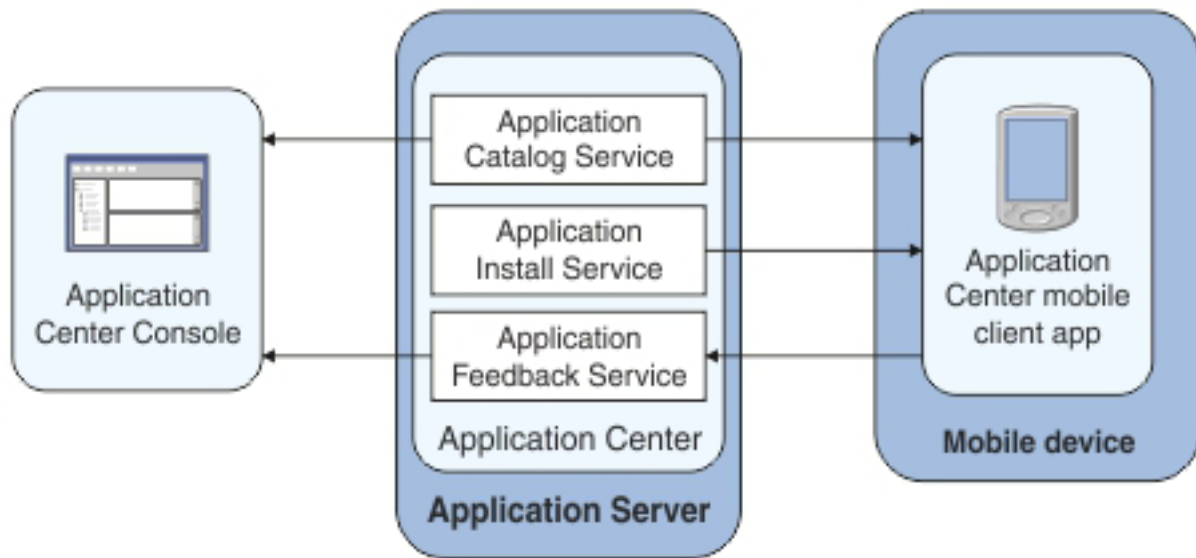


Figure 96. Architecture of the Application Center

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.

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 “Configuring the Application Center after installation” on page 118 or, if you are using LDAP authentication, “Managing users with LDAP” on page 121.

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/appcenterconsole/
```

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 782.

To install and run the mobile client on:

- Android operating system: see “Installing the client on an Android mobile device” on page 805
- iOS operating system: see “Installing the client on an iOS mobile device” on page 808
- BlackBerry OS 6 and OS 7: see “Installing the client on a BlackBerry mobile device” on page 809.

## 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.

The BlackBerry project **IBMAppCenterBlackBerry6** contains the version of the client for BlackBerry OS 6 and OS 7 devices. BlackBerry OS 10 is not supported by the current version of the Application Center.

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`. Push notifications are disabled. If you want to enable push notifications, you must rebuild the .apk file. See "Push notifications of application updates" on page 779 for more information about push notifications in the Application Center.

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."

### Importing and building the project (Android and iOS)

You must import the **IBMAppCenter** project into Worklight Studio and then build the project.

## About this task

Follow the normal procedure to import a project into Worklight Studio.

### Procedure

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 **IBMApCenter** project.
4. Select "IBMApCenter project".
5. Select "Copy projects into workspace". This selection creates a copy of the project in your workspace. On UNIX systems, the **IBMApCenter** project is read only at the original location. so copying projects into workspace avoids problems with file permissions.
6. Click **Finish** to import the **IBMApCenter** project into Worklight Studio.

### What to do next

Build the **IBMApCenter** project. The Worklight project contains a single application named AppCenter. Right-click the application and select **Run as > Build All and Deploy**.

Worklight Studio generates a native Android project in `IBMApCenter/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.

If you want to enable push notifications for application updates, you must first configure the Application Center client properties. See "Configuring push notifications for application updates" on page 779 for more information.

See "Developing hybrid applications for Android" on page 322 for more information about how you can create hybrid mobile applications with Worklight Studio.

### Android and iOS: for experts

You can customize features by editing a central property file and manipulating some other resources.

### Purpose

To customize features: several features are controlled by a central property file called `config.json` in the directory `IBMApCenter/apps/AppCenter/common/js/appcenter/`. If you want to change the default application behavior, you can adapt this property file before you build the project.



## Properties

This file contains the properties shown in the following table.

Table 214. Properties in the `config.js` file

Property	Description
<code>url</code>	The hardcoded address of the Application Center server. If this property is set, the address fields of the Login view are not displayed.
<code>defaultPort</code>	If the <code>url</code> property is null, this property prefills the <code>port</code> field of the Login view on a phone. This is a default value; the field can be edited by the user.
<code>defaultContext</code>	If the <code>url</code> property is null, this property prefills the <code>context</code> field of the Login view on a phone. This is a default value; the field can be edited by the user.
<code>ssl</code>	The default value of the SSL switch of the Login view.
<code>allowDowngrade</code>	This property indicates whether installation of older versions is authorized or not; an older version can be installed only if the operating system and version permit downgrade,
<code>showPreviousVersions</code>	This property indicates whether the device user can show the details of all the versions of applications or only details of the latest version.
<code>showInternalVersion</code>	This property indicates whether the internal version is shown or not. If the value is false, the internal version is shown only if no commercial version is set.
<code>listItemRenderer</code>	This property can have one of these values: <ul style="list-style-type: none"><li>• full, the default value; the application lists show application name, rating, and latest version.</li><li>• simple: the application lists show the application name only.</li></ul>
<code>listAverageRating</code>	This property can have one of these values: <ul style="list-style-type: none"><li>• latestVersion: the application lists show the average rating of the latest version of the application.</li><li>• allVersions: the application lists show the average rating of all versions of the application.</li></ul>
<code>gcmProjectId</code>	The Google API project ID (project name = <code>com.ibm.appcenter</code> ), which is required for Android push notifications; for example, 123456789012.

## Other resources

Other resources that are available are application icons, application name, splash screen images, icons, and translatable resources of the application.

### Application icons

Android: The file named `icon.png` in the `IBMAppCenter/apps/AppCenter/android/native/res/drawabledensity` directories; one directory exists for each density.

iOS: Files named `iconsize.png` in the `IBMAppCenter/apps/AppCenter/iphone/native/Resources` directories.



### Application name

Android: Edit the **app\_name** property in the `IBMAppCenter/apps/AppCenter/android/native/res/values/strings.xml` file.

iOS: Edit the **CFBundleDisplayName** key in the `IBMAppCenter/apps/AppCenter/iphone/native/IBMAppCenterAppCenterIphone-Info.plist` file.

### Splash screen images

Android: Edit the file named `splashimage.9.png` in the `IBMAppCenter/apps/AppCenter/android/native/res/drawable/density` directories; one directory exists for each density. This file is a patch 9 image.

iOS: Files named `Default-size.png` in the `IBMAppCenter/apps/AppCenter/iphone/native/Resources` directories.

### Icons (buttons, stars, and similar objects) of the application

`IBMAppCenter/apps/AppCenter/common/css/images`.

### Translatable resources of the application

`IBMAppCenter/apps/AppCenter/common/js/appcenter/nls/common.js`.

## Importing and building the project (BlackBerry)

You must import the BlackBerry project into the BlackBerry Eclipse IDE and then build the project.

### About this task

Follow the normal procedure to import a project into the BlackBerry Eclipse IDE.

### Procedure

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.

### What to do next

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, 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.

## BlackBerry: for experts

You can customize features by adapting a central property file and manipulating some other resources .

## Purpose

To customize features: look and feel and various features are controlled by a central property file called `appcenter.properties` in the directory `IBMApCenterBlackBerry6/src/main/resources`. If you want to disable or customize 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.

## Properties

This file contains the properties shown in the following table.

Table 215. Properties in the `appcenter.properties` file

Property	Description
<code>defaultServer</code>	The default value of the <b>server</b> field of the Login view. The field can be edited by the user.
<code>defaultPort</code>	The default value of the <b>port</b> field of the Login view. The field can be edited by the user.
<code>defaultContext</code>	The default value of the <b>context</b> field of the Login view. The field can be edited by the user.
<code>defaultUseSSL</code>	The default value of the SSL switch of the Login view.
<code>KeepLoginCredentialsTime</code>	The number of minutes the password remains valid after exiting the application. If set to 0, the user must log in again whenever the application starts. If set to -1, the login credentials are kept forever until the user explicitly logs out. If any other value is given and the user restarts the application within this time, it is not necessary to log in again.

Table 215. Properties in the appcenter.properties file (continued)

Property	Description
<b>allowVersionSelection</b>	This property indicates whether the user can show the details of all the versions of applications or only the details of the latest version. If disabled, it is only possible to install the latest version of the application. If enabled, it is possible to revert to any older version of the application.
<b>AdaptAppCatalogInfoLineToSorting</b>	This property indicates whether the rendering of the application list shows popularity or updates when sorting according to popularity and updates. Normally, the rendering shows version numbers. When this feature is enabled and you choose sorting according to the timestamps of popularity or updates, the rendering shows popularity or update timestamps instead of versions.

## Other resources

Other resources that are available are application icon, application name, icons, and translatable resources of the application.

### Application icon

IBMAppCenterBlackBerry6/src/main/resources/img/launchicon-144x144.png.

### Application name

Edit the IBMAppCenterBlackBerry6/BlackBerry\_App\_Descriptor.xml file. The key **title** is the application name.

### Icons (buttons, stars, and similar objects) of the application

IBMAppCenterBlackBerry6/src/main/resources/img/.

Depending on the color theme, either dark or light icons are chosen. For example, if the background is dark, light icons are chosen. Therefore, all icon file names have the suffix "light" or "dark". Several buttons can be disabled. To show the corresponding icon on a disabled button, some icons have the file name suffix "t50". The visual indicator of disabled buttons is implemented by adding 50% transparency to the icon.

### Translatable resources of the application

IBMAppCenterBlackBerry6/src/main/resources/com/ibm/appcenter/i18n/I18N.rrc.

## Deploying the mobile client in the Application Center

Deploy the different versions of the client application to 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 784 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 805, “Installing the client on an iOS mobile device” on page 808, or “Installing the client on a BlackBerry mobile device” on page 809.

## Push notifications of application updates

You can configure the Application Center client so that push notifications are sent to users when an update is available for an application in the store.

The Application Center administrator uses push notifications to automatically send a notification to any iOS or Android device where a specific application is installed when a new version of this application is available.

Push notifications are currently not available for the BlackBerry Application Center client.

### Push notification process

The first time that the Application Center client starts on a device, the user might be asked whether or not to accept incoming push notifications; that is the case for iOS mobile devices. The push notification feature does not work when the service is disabled on the mobile device. iOS and modern Android operating system versions offer a way to switch this service on or off on a per application basis. Refer to your device vendor to learn how to configure your mobile device for push notifications.

### Configuring push notifications for application updates

Configure the Application Center services to communicate with Google or Apple push notification servers.

#### Purpose

You must configure the credentials or certificates of the Application Center services to be able to communicate with third-party push notification servers. of application updates to mobile devices where the applications are installed.

### Configuring the server scheduler of the Application Center

The server scheduler is a background service that automatically starts and stops with the server. This scheduler is used to empty at regular intervals a stack that is automatically filled by administrator actions with push update messages to be sent. The default interval between sending two batches of push update messages is twelve hours. If this default value does not suit you, you can modify it by using the server environment variable *ibm.appcenter.push.schedule.period.hours*. This variable is used to define the elapsed time in hours between two batches of push messages.

Use a JNDI property to define this variable.

#### Example for Apache Tomcat server

Define this variable with a JNDI property in the *server.xml* file:

```
<Environment name="ibm.appcenter.push.schedule.period.hours" override="false" type="java.lang.Stri
```

For information about how to configure JNDI variables for WebSphere Application Server v8.5, see Using resource environment providers in WebSphere Application Server.

For information about how to configure JNDI variables for WebSphere Application Server Liberty Profile, see Using JNDI binding for constants from the server configuration files.

The remaining actions for setting up the push notification service depend on the vendor of the device where the target application is installed. See the following topics.

## Configuring the Application Center server for connection to Google Cloud Messaging

Enable Google Cloud Messaging (GCM) for your application.

### About this task

To enable Google Cloud Messaging (GCM) for an application, you must attach the GCM services to a developer Google account with the Google API enabled. See Getting Started with GCM for details.

### Procedure

1. If you do not have the appropriate Google account, go to Create a Google account and create one for the Application Center client.
2. Register this account by using the Google API in the Google API console. Registration creates a new default project that you can rename. The name you give to this GCM project is not related to your Android application package name. When the project is created, a GCM project ID is appended to the end of the project URL. You should record this trailing number as your project ID for future reference.
3. Enable the GCM service for your project; in the Google API console, click the **Services** tab on the left and enable the "Google Cloud Messaging for Android" service in the list of services.
4. Make sure that a Simple API Access Server key is available for your application communications.
  - a. Click the **API Access** vertical tab on the left of the console.
  - b. Create a Simple API Access Server key or, if a default key is already created, note the details of the default key. Two other kinds of key exist that are not of interest at this time.
  - c. Save the Simple API Access Server key for future use in your application communications through GCM. The key is about 40 characters long and is referred to as the Google API key that you will need later on the server side.
5. Enter the GCM project ID as a string resource property in the JavaScript project of the Application Center Android client; in the `IBMAppCenter/apps/AppCenter/common/js/appcenter/config.json` template file, modify this line with your own value:

```
gcmProjectId:""// Google API project (project name = com.ibm.appcenter) ID needed for Android pus
// example : 123456789012
```
6. Register the Google API key as a JNDI property for the Application Center server. The key name is : **ibm.appcenter.gcm.signature.googleapikey**. For example, you can configure this key for an Apache Tomcat server as a JNDI property in the `server.xml` file:

```
<Context docBase="AppCenterServices" path="/applicationcenter" reloadable="true" source="org.e
<Environment name="ibm.appcenter.gcm.signature.googleapikey" override="false" type="java.lang.
value="AIXaScCHg0VSGdgfOZKtzDJ44-oi0muUasMZvAs"/>
</Context>
```

The JNDI property must be defined in accordance with your application server requirements.

**Important:** If you use GCM with earlier versions of Android, you might need to pair your device with an existing Google account for GCM to work effectively. See GCM service: “It uses an existing connection for Google services. For pre-3.0 devices, this requires users to set up their Google account on their mobile devices. A Google account is not a requirement on devices running Android 4.0.4 or higher.”

## Configuring the Application Center server for connection to Apple Push Notification Services

Configure your iOS project for Apple Push Notification Services (APNs).

### About this task

You must be a registered Apple developer to successfully configure your iOS project with Apple Push Notification Services (APNs). In the company, the administrative role responsible for Apple development requests APNs enablement. The response to this request should provide you with an APNs-enabled provisioning profile for your iOS application bundle; that is, a string value that is defined in the configuration page of your Xcode project. This provisioning profile is used to generate a signature certificate file.

Two kinds of provisioning profile exist: development and production profiles, which address development and production environments respectively. Development profiles address Apple development APNs servers exclusively. Production profiles address Apple production APNs servers exclusively. These kinds of servers do not offer the same quality of service.

### Procedure

1. Obtain the APNs-enabled provisioning profile for the Application Center Xcode project. The result of your administrator's APNs enablement request is shown as a list accessible from <https://developer.apple.com/ios/my/bundles/index.action>. Each item in the list shows whether or not the profile has APNs capabilities. When you have the profile, you can download and install it in the Application Center client Xcode project directory by double-clicking the profile. The profile is then automatically installed in your keystore and Xcode project.
2. If you want to test or debug the Application Center on a device by launching it directly from Xcode, in the "Xcode Organizer" window, go to the "Provisioning Profiles" section and install the profile on your mobile device.
3. Create a signature certificate used by the Application Center services to secure communication with the APNs server. This server will use the certificate for purposes of signing each and every push request to the APNs server. This signature certificate is produced from your provisioning profile.
  - a. Open the "Keychain Access" utility and click the **My Certificates** category in the left pane.
  - b. Find the certificate you want to install and disclose its contents. You see both a certificate and a private key; for the Application Center, the certificate line contains the Application Center application bundle `com.ibm.imf.AppCenter`.



- c. Select **File > Export Items** to select both the certificate and the key and export them as a Personal Information Exchange (.p12) file. This .p12 file contains the private key required when the secure handshaking protocol is involved to communicate with the APNs server.
- d. Copy the .p12 certificate to the computer responsible for running the Application Center services and install it in the appropriate place. Both the certificate file and its password are needed to create the secure tunneling with the APNs server. You also require some information that indicates whether a development certificate or a production certificate is in play. A development provisioning profile produces a development certificate and a production profile gives a production certificate. The Application Center services web application uses JNDI properties to reference this secure data. The examples in the table show how the JNDI properties are defined in the server.xml file of the Apache Tomcat server.

Table 216. JNDI properties

JNDI Property	Type and description	Example for Apache Tomcat server
<code>ibm.appcenter.apns.p12.certificate.location</code>	defines the full path to the .p12 certificate.	Environment name=" <code>ibm.appcenter.apns.p12.certificate.location</code> " override="false" type="java.lang.String" value="/Users/someUser/someDirectory/apache-tomcat/conf/AppCenter/ibm.appcenter.apns.p12.certificate.location"/>
<code>ibm.appcenter.apns.p12.certificate.password</code>	defines the password needed to access the certificate.	Environment name=" <code>ibm.appcenter.apns.p12.certificate.password</code> " type="java.lang.String" value=" <code>this_is_a_secure_password</code> ">
<code>ibm.appcenter.apns.p12.certificate.isDevelopmentCertificate</code>	(identified as true or false) that defines whether or not the provisioning profile used to generate the authentication certificate was a development certificate.	Environment name=" <code>ibm.appcenter.apns.p12.certificate.isDevelopmentCertificate</code> " override="false" type="java.lang.String" value="true"/>

## The Application Center console

With the Application Center console, you can manage the repository of the Application Center and your applications.

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.



Figure 97. 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.

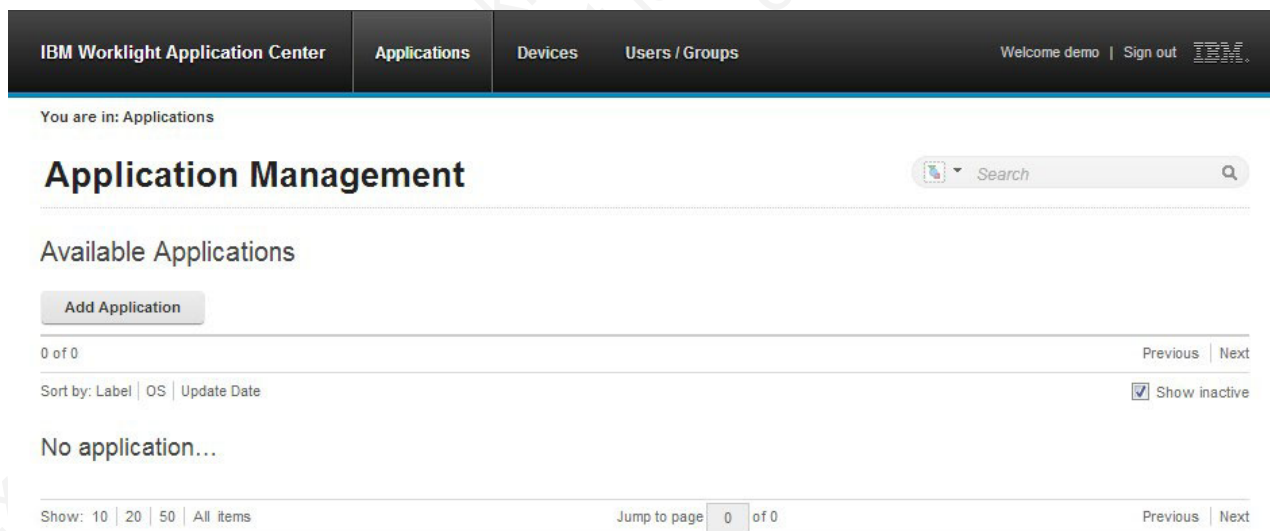
## Application Center installed on WebSphere Application Server Liberty Profile or on Apache Tomcat

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.



The screenshot shows the IBM Worklight Application Center interface. At the top, there is a navigation bar with tabs for 'Applications', 'Devices', and 'Users / Groups'. The 'Applications' tab is selected. Below the navigation bar, the page title is 'Application Management'. There is a search bar on the right. The main content area is titled 'Available Applications' and contains an 'Add Application' button. Below the button, it shows '0 of 0' items. There are sorting options: 'Sort by: Label | OS | Update Date' and a checkbox for 'Show inactive' which is checked. At the bottom, there are pagination controls: 'Show: 10 | 20 | 50 | All items', 'Jump to page: 0 of 0', and 'Previous | Next' links.

Figure 98. 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 for normal iOS applications.

The application file extension is zip for instrumented iOS applications for use in IBM Mobile Test Workbench for Worklight.

### BlackBerry

The application file extension is zip. This archive file must contain a file with extension jad and all related files with extension cod.

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**.

IBM Worklight Application Center | Applications | Devices | Users / Groups | Welcome demo | Sign out

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: User who has uploaded this application  
(2548 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

Instrumented: No  
Indicates whether this application is instrumented for IBM Mobile Test Workbench for Worklight

Previous Finish Cancel

Figure 99. 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.
- **Instrumented** indicates whether the uploaded application is an instrumented application for use in IBM Mobile Test Workbench for Worklight or a normal iOS application.

### 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

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 805.

### 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 785 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.

You are in: Applications > AppMan Sample



# AppMan Sample (iOS)

## version 1.0

- Properties
- Reviews

### Edit application properties

#### Application Details

Package, Version and Label must be set in the uploaded application package and cannot be modified afterwards.

Package:	AppMan <small>Identifies the application</small>
Internal Version:	1.0 <small>Internal version number used to compare versions</small>
Commercial Version:	<small>Version displayed on the mobile device</small>
* Label:	<input type="text" value="AppMan Sample"/> <small>Label of the application as defined by the developer</small>
Author:	demo <small>User who has uploaded this application</small>
Description:	<input type="text"/> <small>(2048 characters maximum)</small>
Recommended:	<input type="checkbox"/> <small>This application will be listed as a recommended application on the mobile device</small>
Installer:	<input type="checkbox"/> <small>Indicates whether this application is an installer</small>
Active:	<input checked="" type="checkbox"/> <small>An active application can be installed on a device</small>
Ready for production:	<input type="checkbox"/> <small>Indicates whether this application is ready for production</small>
Instrumented	No <small>Indicates whether this application is instrumented for IBM Mobile Test Workbench for Worklight</small>

#### Application File

Define an application file with an ipa extension for an iOS application to update the application.

Current:	appman.ipa
New file:	<input type="text"/> <input type="button" value="Browse..."/>

Figure 100. 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 reviews

In the Application Center console, you can see reviews about mobile application versions sent by users.

## About this task

Users of mobile applications can write a review, which includes a rating and a comment, and submit the review through the Application Center client. Reviews are available in the Application Center console and the client. Individual reviews are always associated with a particular version of an application.

## Procedure

To view reviews from mobile users or testers about an application version:


1. Select **Applications** to see the list of uploaded applications: Available Applications.
2. Select the version of the application.
3. In the menu, select **Reviews**.

The screenshot displays the IBM Worklight Application Center interface. At the top, there is a navigation bar with tabs for 'Applications', 'Devices', and 'Users / Groups'. The current page is titled 'AppMan Sample (iOS) version 1.0'. Below the title, there is a search bar and a 'Properties' menu with 'Reviews' selected. The main content area is titled 'Application Reviews' and shows an average rating of 4.5 stars. There are two reviews listed: one with a 5-star rating and the comment 'This is a great application!', and another with a 4-star rating and the comment 'Nice application!'. The interface includes pagination controls (Page 1 of 2) and sorting options (Sort by: Rating, Creation Date).

Figure 101. Reviews of application versions

The rating is an average of the ratings in all recorded reviews. It consists of one to five stars, where one star represents the lowest level of appreciation and five stars represent the highest level of appreciation. The client cannot send a zero star rating.

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 is part of the review and to view the details of the mobile device where the review is generated.

For example, the comment can give the reason for submitting the review, such as failure to install.

If you want to delete the review, 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.

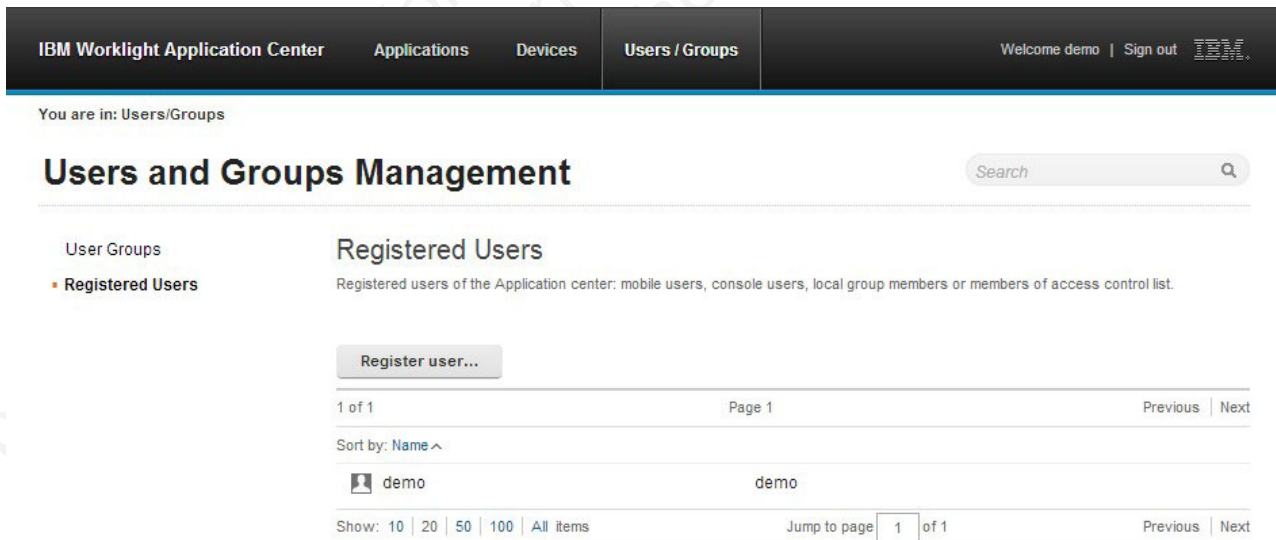
### Purpose

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



The screenshot shows the IBM Worklight Application Center interface. At the top, there is a navigation bar with tabs for 'Applications', 'Devices', and 'Users / Groups'. The 'Users / Groups' tab is selected. Below the navigation bar, there is a search bar and a 'Search' button. The main content area is titled 'Users and Groups Management' and contains a sidebar with 'User Groups' and 'Registered Users' (selected). The 'Registered Users' section displays a list of users, with one user named 'demo' visible. The interface includes a 'Register user...' button, a 'Sort by: Name' dropdown, and pagination controls showing '1 of 1' items and 'Page 1'.

Figure 102. 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.

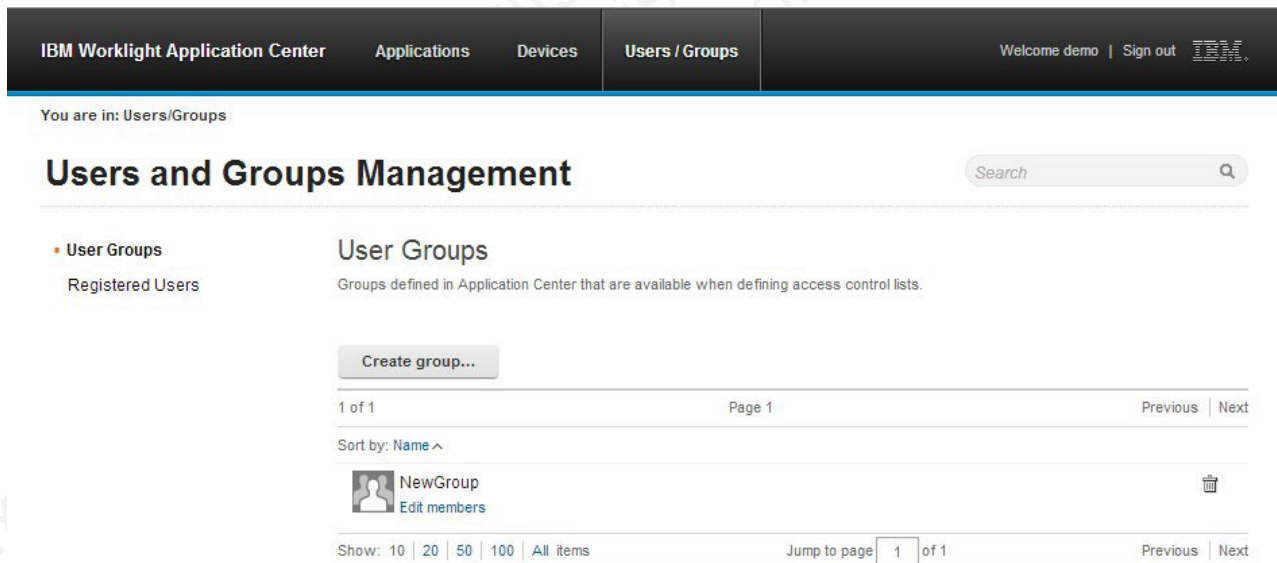


Figure 103. Local user groups

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.

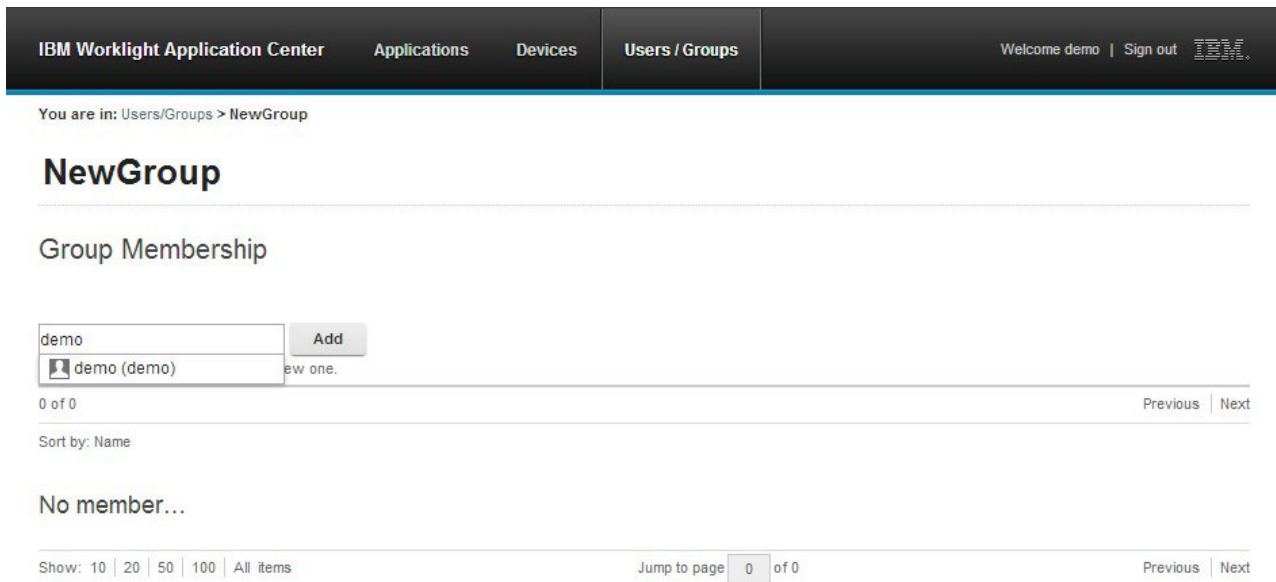


Figure 104. Managing group membership

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.

## Procedure

You can edit access control:

1. In Application Management under Available Applications, click the **unrestricted** or **restricted** state of Installation of an application.



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.

Figure 105. 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.

IBM Worklight Application Center Applications **Devices** Users / Groups Welcome demo | Sign out IBM


You are in: Devices

## 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 106. 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.

IBM Worklight Application Center Applications **Devices** Users / Groups Welcome demo | Sign out IBM

You are in: Devices > Demo's phone

## Demo's phone

Properties Installed Applications

### Edit Device Properties

\* Name:   
Name of the device

User Name: demo

Manufacturer: Apple

Model: iPhone 5

Operating System: iOS

Unique Identifier: myDeviceId

OK Apply Cancel

Figure 107. 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.



The screenshot shows the IBM Worklight Application Center interface. At the top, there are navigation tabs for 'Applications', 'Devices', and 'Users / Groups'. The current page is for a device named 'Demo's phone'. On the left, there is a sidebar with 'Properties' and 'Installed Applications' (selected). The main content area is titled 'Applications installed on device' and shows a table with one application: 'AppMan Sample 1.0' (iOS (AppMan)). The table has a search bar, sorting options (Label, OS, Update Date), and pagination controls (Page 1, Previous, Next).

Figure 108. 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 or deleting 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.

- `build.xml`: a sample ant script that you can use to upload a single file or a sequence of files to the Application Center.
- `acdeploytool.sh` and `acdeploytool.bat`: Simple scripts to call java with `applicationcenterdeploytool.jar`.

## 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
<code>-s</code>	<b>serverpath</b>	The path to the Application Center server.
<code>-c</code>	<b>context</b>	The context of the Application Center web application.
<code>-u</code>	<b>user</b>	The user credentials to access the Application Center.
<code>-p</code>	<b>password</b>	The password of the user.
<code>-d</code>	<b>description</b>	The description of the application to be uploaded.
<code>-l</code>	<b>label</b>	The fallback label. Normally the label is taken from the application descriptor stored in the file to be uploaded. If the application descriptor does not contain a label, the fallback label is used.
<code>-isActive</code>	true or false	The application is stored in the Application Center as an active or inactive application.
<code>-isInstaller</code>	true or false	The application is stored in the Application Center with the "installer" flag set appropriately.
<code>-isReadyForProduction</code>	true or false	The application is stored in the Application Center with the "ready-for-production" flag set appropriately.
<code>-isRecommended</code>	true or false	The application is stored in the Application Center with the "recommended" flag set appropriately.
<code>-e</code>		Shows the full exception stack trace on failure.

-f		Force uploading of applications, even if they exist already.
-y		Disable SSL security checking, which allows publishing on secured hosts without verification of the SSL certificate. Use of this flag is a security risk, but may be suitable for testing localhost with temporary self-signed SSL certificates.

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:9080 -c applicationcenter -u demo -p demopass
```

### Using the stand-alone tool to delete an application

To delete an application from the Application Center, 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 -delete [options] [files or applications]
```

You can pass any of the available options in the command line.

Option	Content indicated by	Description
-s	<b>serverpath</b>	The path to the Application Center server.
-c	<b>context</b>	The context of the Application Center web application.
-u	<b>user</b>	The user credentials to access the Application Center.
-p	<b>password</b>	The password of the user.
-y		Disable SSL security checking, which allows publishing on secured hosts without verification of the SSL certificate. Use of this flag is a security risk, but may be suitable for testing localhost with temporary self-signed SSL certificates.

You can specify files or the application package, operating system, and version. If files are specified, the package, operating system and version are determined

from the file and the corresponding application is deleted from the Application Center. If applications are specified, they must have one of the following formats:

`package@os@version`: This exact version is deleted from the Application Center. The version part must specify the “internal version”, not the “commercial version” of the application.

`package@os`: All versions of this application are deleted from the Application Center.

`package`: All versions of all operating systems of this application are deleted from the Application Center.

## Example

In this example, user **demo** has the password **demopassword**. Use this command line to delete the Android application `demo.HelloWorld` with internal version 3.

```
java com.ibm.appcenter.Upload -delete -s http://localhost:9080 -c applicationcenter -u demo -p demopassword
```

## Ant task for uploading or deleting an application

You can use the upload and delete tools 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 upload ant task is `com.ibm.appcenter.ant.UploadApps`. The **classname** of the delete ant task is `com.ibm.appcenter.ant.DeleteApps`.

Parameters of ant task	Description
<b>serverPath</b>	To connect to the Application Center. The default value is <code>http://localhost:9080</code> .
<b>context</b>	The context of the Application Center. The default value is <code>/applicationcenter</code> .
<b>loginUser</b>	The user name with permissions to upload an application.
<b>loginPass</b>	The password of the user with permissions to upload an application.
<b>forceOverwrite</b>	If set to true, the ant task attempts to overwrite applications in the Application Center when it uploads an application that is already present. This parameter is only available in the upload ant task.
<b>file</b>	The <code>.apk</code> or <code>.ipa</code> file to be uploaded to the Application Center or to be deleted from the Application Center. This parameter has no default value.
<b>fileset</b>	To upload or delete multiple files.
<b>application</b>	The package name of the application; this parameter is only available in the delete ant task.
<b>os</b>	The operating system of the application (Android, iOS, BlackBerry). This parameter is only available in the delete ant task.

<b>version</b>	The internal version of the application; this parameter is only available in the delete ant task. Do not use the commercial version here, because the commercial version is unsuitable to identify the version exactly.
----------------	---

## Example

An extended example can be found in the directory `ApplicationCenter/tools/build.xml`.

The following 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="${upload.file}"
    </target>
  <target name="upload.AllApps" description="Uploads all found APK and IPA files" depends="upload.App">
    <uploadapps serverPath="${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
```

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
<b>install.dir</b>	Defaults to ../../
<b>server.path</b>	The default value is http://localhost:9080.
<b>context.path</b>	The default value is applicationcenter.
<b>upload.file</b>	This property has no default value. It must include the exact file path.
<b>workspace.root</b>	Defaults to ../../
<b>login.user</b>	The default value is appcenteradmin.
<b>login.pass</b>	The default value is admin.
<b>force</b>	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.

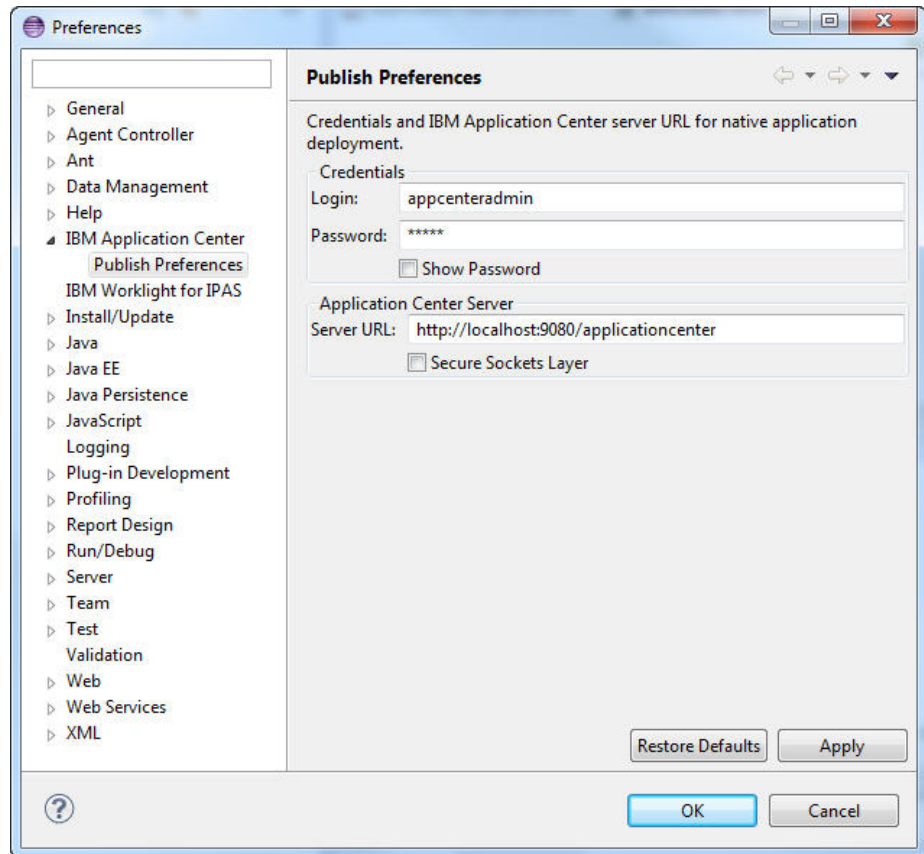
### About this task

You can deploy applications for Android, iOS, and BlackBerry operating systems to the Application Center directly from the IBM Worklight Studio IDE. In Worklight Studio, you can deploy Android application package (.apk) files, iOS application (.ipa) files, and BlackBerry (.zip) files that you choose from your file system. You can right-click an application (.apk, .ipa, or .zip) 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**.

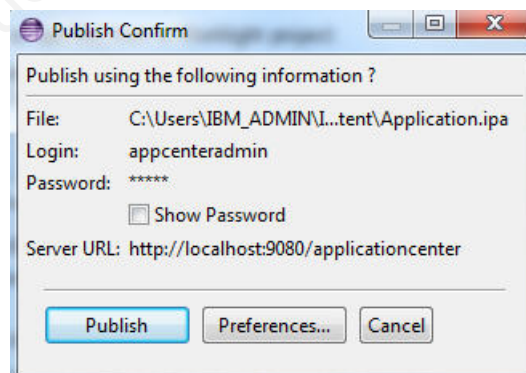


c. Specify the default publish preference settings for the Application Center:

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, .ipa, or .zip 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.

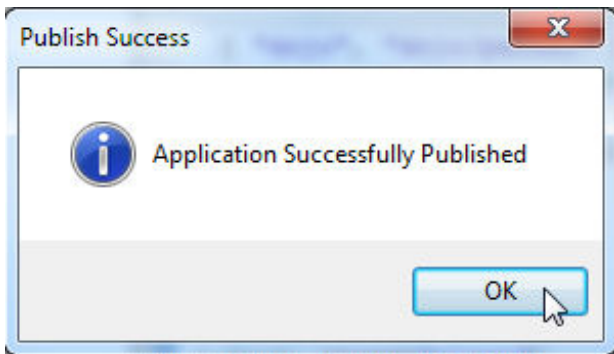


b. In the Publish Confirm dialog, choose one of the following options:

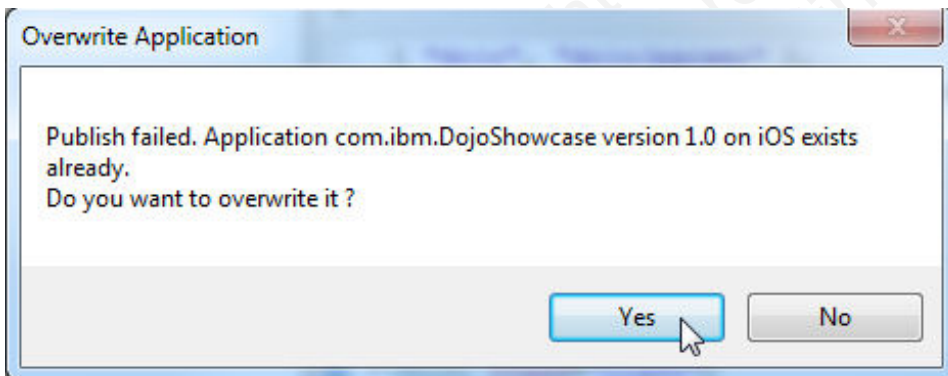


Option	Description
Publish the application by using the current preferences.	Click <b>Publish</b> .
Change any of the preferences before publishing the application.	Click <b>Preferences</b> to open the Publish Preferences page and edit the preference settings.

You receive confirmation when publication is successful.

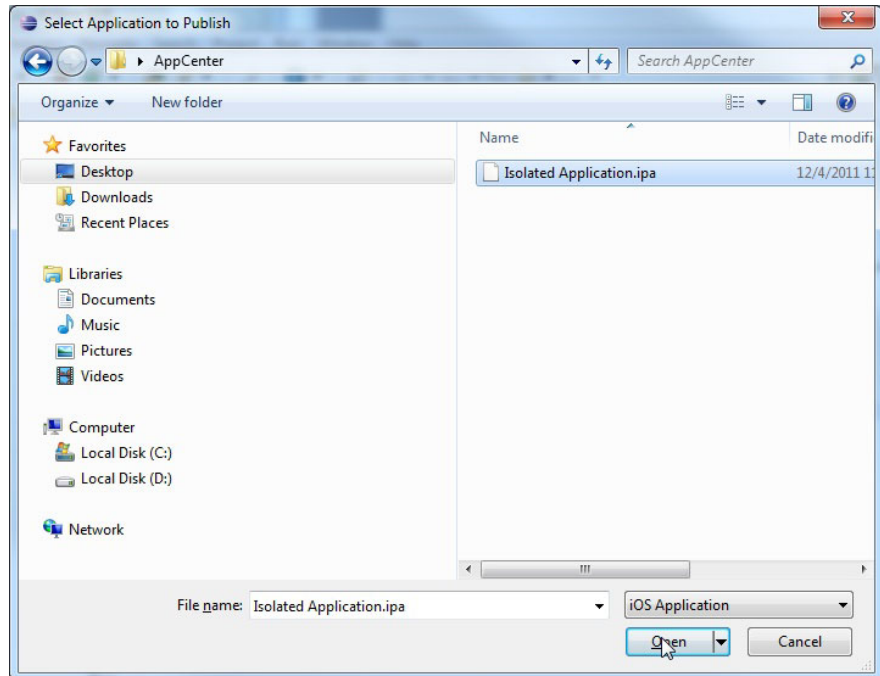


If the application already exists, publication will fail. You are given the option to overwrite the existing version of the application.



**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.



- 2) Navigate to the application (.apk, .ipa, or .zip) 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.

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 Application Center also provides an alternative URL for installing the client on a mobile device: `http://hostname:portnumber/appcenterconsole/inst.html`

The page of this URL works better with some older or some nonstandard mobile web browsers. If the page `installers.html` does not work on your mobile device, you can use `inst.html`.

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 805.

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 785.

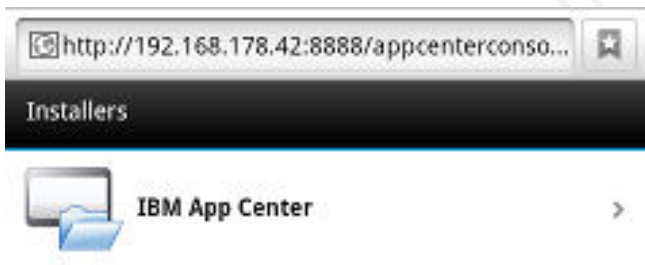


Figure 109. List of available mobile client applications to install

4. Select an item in the list to display the application details.  
Typically, these details include the application name and its version number.

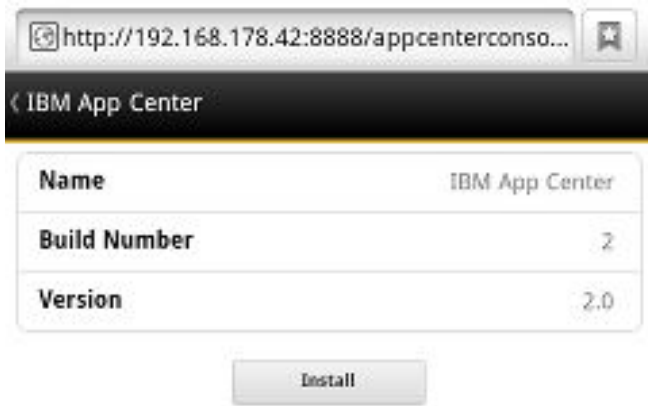


Figure 110. Application details

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.

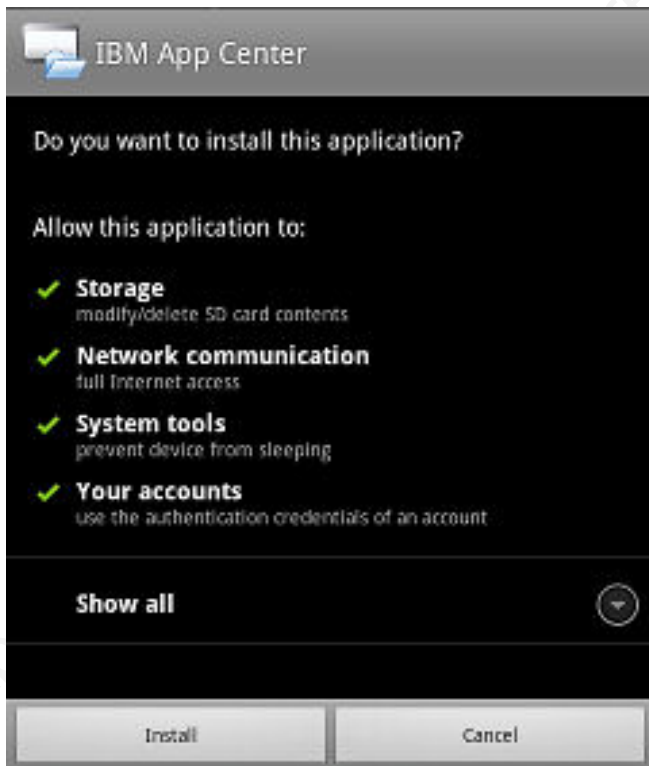


Figure 111. Installation of the mobile client on Android

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.

The Application Center also provides an alternative URL for installing the client on a mobile device: `http://hostname:portnumber/appcenterconsole/inst.html`

The page of this URL works better with some older or some nonstandard mobile web browsers. If the page `installers.html` does not work on your mobile device, you can use `inst.html`.

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**.



Figure 112. 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.

#### 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.

The Application Center also provides an alternative URL for installing the client on a mobile device: `http://hostname:portnumber/appcenterconsole/inst.html`

The page of this URL works better with some older or some nonstandard mobile web browsers. If the page `installers.html` does not work on your mobile device, you can use `inst.html`.

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.

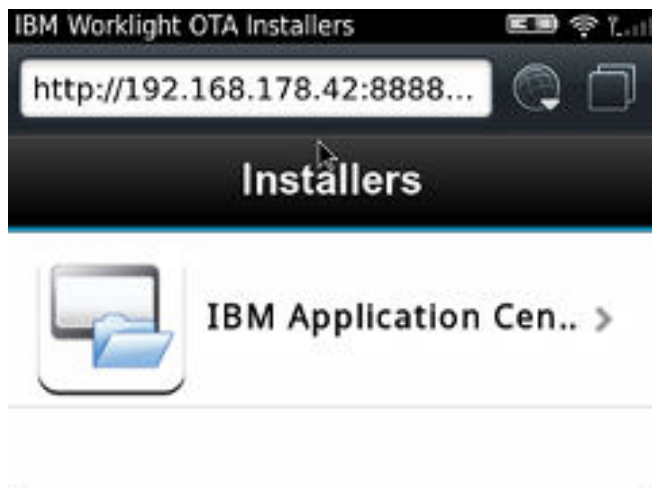


Figure 113. The installer in the BlackBerry browser

**Note:** BlackBerry OS 10 is not supported by the current version of the Application Center.

### The Login view

In the Login 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 **Login** view to enter your credentials to connect to the Application Center server to view the list of applications available for your device.

The **Login** view presents all the mandatory fields for the information required to connect to the server.

When the application is started the Login page is displayed. The login credentials are required to connect to the server.



3G 3:53

## App Center

**Login** Enter your ID

**Password** Enter your password

**Server** Enter IP or host name

**Port** Enter port (optional)

**Context** Enter context (optional)

**SSL**  OFF

**Log in**

Figure 114. Login view on Android or iOS phone

## IBM Worklight Application Center

Login: appcenteruser

Password: ....

Server: myappcenter.ibm.com:9080/applicationcenter

SSL:  ON

Log in

Figure 115. Login view on Android or iOS tablet

Log In / Settings

**IBM Application Center**  
This application requires you to enter login credentials and server URL!

User:  
Enter your user ID!

Password:  
Enter your password!  
 Show password  
 Complex input (e.g. Chinese)

Server:  
Enter IP or host name!

Port:  
Enter port (optional)!

Context:  
Enter context (optional)!

SSL

Log in

Figure 116. Login view on BlackBerry devices

### 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

The Application Center server address is composed of:

- Host name or IP address.
- Port, which is optional if the default port is used.
- Context, which is optional if the Application Center is installed at the root of the server.

On a phone, a field is available for each part of the address.

On a tablet, a single field that contains a preformatted example address is displayed. Use it as a model for entering the correct server address to avoid formatting errors. See “Preparations for using the mobile client” on page 772 for information on filling parts of the address in advance, or hardcode the address and hide the associated fields.

### 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.)

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.

### Complex input on BlackBerry devices

If you have non-Latin characters to enter in the text field, such as Chinese and Japanese user names, select **Complex input** on a BlackBerry device. Selecting **Complex input** switches to the BlackBerry complex input mode in all text fields of the application.

### Connecting to the server

To connect to the server:

1. Enter your user name and password.
2. Enter your Application Center server address.
3. If your configuration of the Application Center runs over the SSL protocol, select **SSL**.
4. Tap **Log in** to connect to the server.

If this login is successful, the user name and server address are saved to fill the fields on subsequent launches of the client.

### Views in the Application Center client

The client provides views that are adapted to the various tasks that you want to perform.

After a successful login, you can choose among these views.



Figure 117. Views in the client application (Android and iOS operating systems)

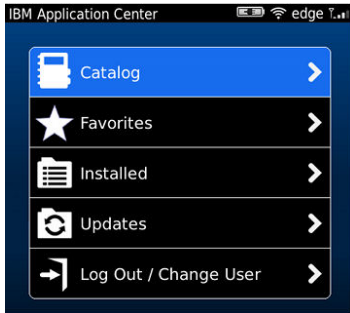


Figure 118. Views in the client application (BlackBerry devices)

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.

### Catalog

This view shows the applications that can be installed on a device.

### Favorites

This view shows the list of applications that you marked as favorites.

### Installed on BlackBerry version.

This view shows the applications installed on your mobile device. This view is not available on Android and iOS versions of the client.

### Updates

This view shows all applications that you marked as favorite apps and that have a later version available in the Application Center than the version, if any, installed on the device.

When you first start the mobile client, it opens the **Login** view for you to enter your user name, password, and the address of the Application Center server. This information is mandatory.

### Displays on different device types

The layout of the views is specific to the Android, iOS, or BlackBerry environment, even though the common functions that you can perform in the views are the same for all operating systems. Different device types might have quite different screen real estate. On the phone, a list is displayed. On a tablet, a grid of applications is used.

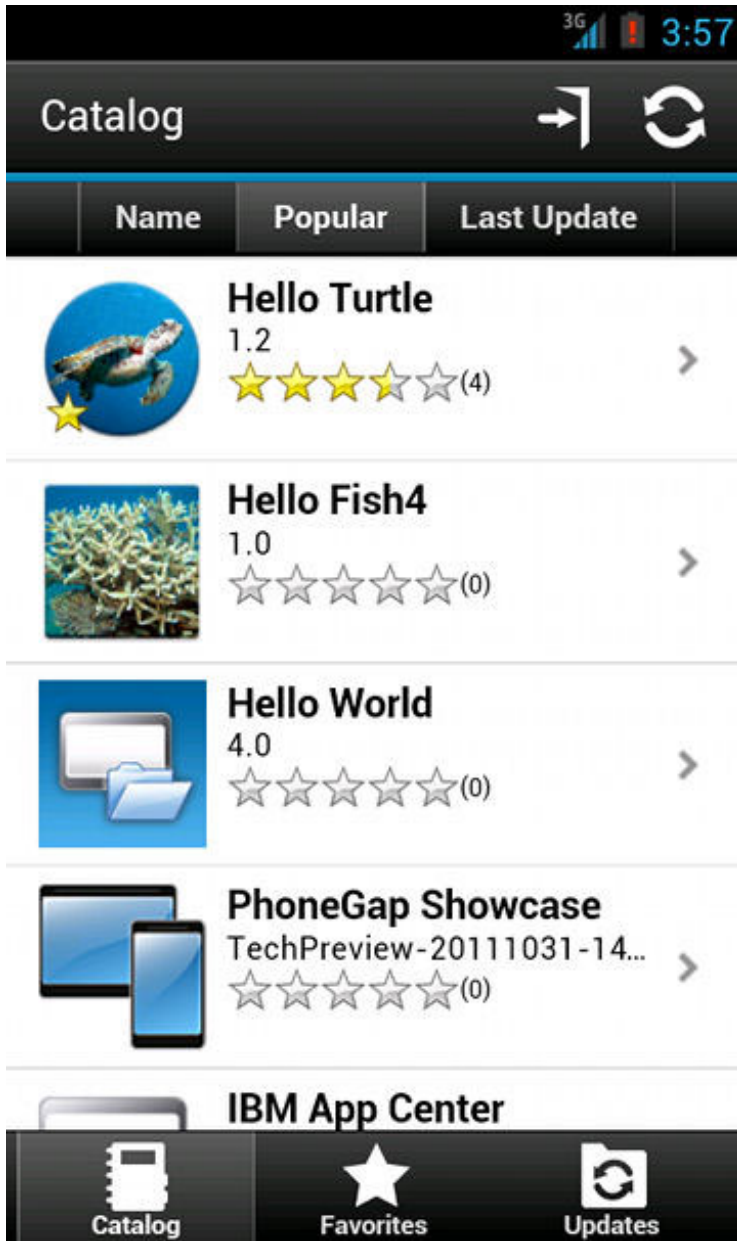


Figure 119. Catalog view on a phone

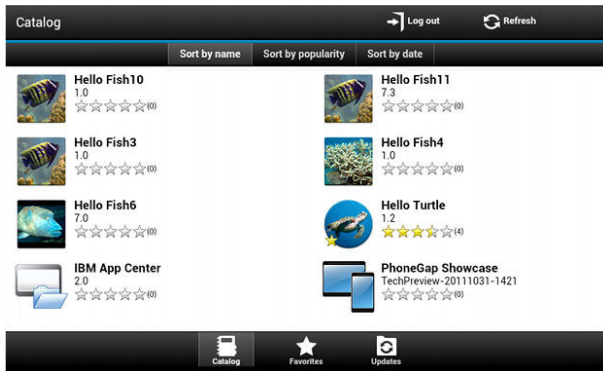



Figure 120. Catalog view on a tablet

## Features of the views

You can sort the lists by tapping one of the sort criteria at the top of the view. In the Android and iOS applications, criteria are displayed as tab panes. In the

BlackBerry client, sort criteria are available through the sort button. 


On BlackBerry devices, if the list of applications is too long, you can use the search field to find an application that contains the search string in its name.



Applications that are marked as favorites are indicated by a star superposed on the application icon.

The average rating of the latest version of an application is shown by using a number of stars and the number of ratings received. See “Preparations for using the mobile client” on page 772 for how to show the rating of all versions of the application instead of the latest version only.

Tapping an application in the list navigates to the Details view of the latest installed version of this application.

You can refresh a view by tapping the Refresh button. 

To return to the login page:

- In the Android and iOS applications, tap the logout button. 

- In the BlackBerry version of the client, tap the return button.  Then select Log out/Change User.

## The Details view

Tapping an application in the Catalog, Favorites, or Updates view navigates to the Details view where you can see details of the application properties. Details of the application version are displayed in this view.

- The name of the application.
- Commercial version: the published version of the application.
- Internal version: on Android, the internal version identification of the application; on iOS, the build number of the application; on BlackBerry, the version of the application and the same as the commercial version. See “Application properties” on page 785 for technical details concerning this property on all operating systems.
- Update date.
- Approximate size of the application file.
- Rating of the version and number of ratings received.
- Description of the application.

You can perform several actions in this view.

- Install, upgrade, downgrade, or uninstall an application version.
- Cancel the current operation in progress.
- Rate the application version if it is installed on the device.
- List the reviews of the this version or of all versions of the application.
- Show details of a previous version.
- Mark or unmark the application as a favorite app.
- Refresh the view with the latest changes from the Application Center server.

### **Installing an application on an Android device**

From the **Details** view, you can install an application on your Android device.

#### **About this task**

In the **Details** view, if a previous version of the application is not installed, you can install this application version on your Android device.



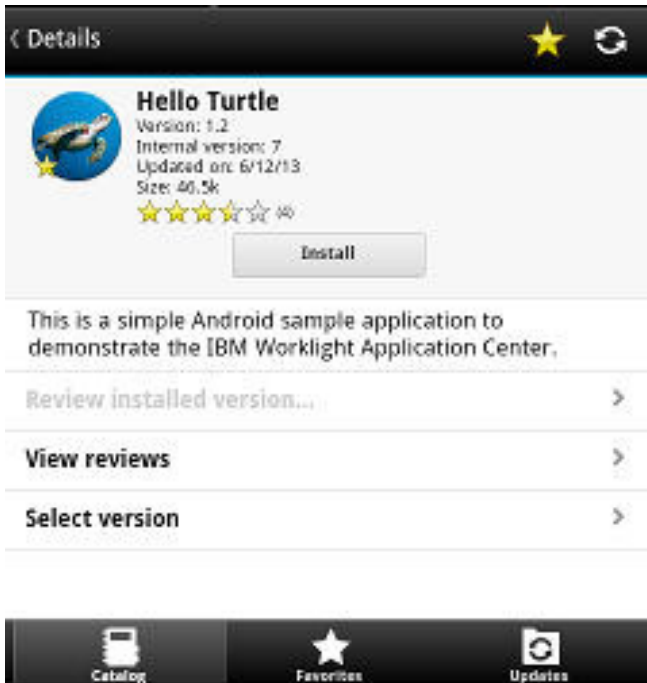


Figure 121. Details view of an app version shown on your Android device

### Procedure

1. In the **Details** view, tap **Install**.

The application is downloaded. You can tap **Cancel** in the **Details** view at any time during the download to cancel the download. (The **Cancel** button appears only during the installation steps.) If you let the download complete, you will see the rights that are granted to the application.



Figure 122. Application rights on your Android device

2. Tap **Install** to confirm installation of the application or **Cancel** to cancel installation..

Depending on the action taken, the application is installed or not.

If you selected **Cancel**, in the application rights confirmation panel, you can tap **Cancel** in the **Details** view at any time to notify the application that the installation has been canceled. The **Cancel** button appears in the **Details** view only during the installation steps.

### Installing an application on an iOS device

From the **Details** view, you can install or reinstall an application version on your iOS mobile device.

#### About this task



Figure 123. Details view of an app version shown on your iOS device

#### Procedure

1. In the **Details** view, tap **Install** or **Reinstall**. You are requested to confirm the download and installation of the application version.
2. Tap **Install** to confirm download and installation of the application version or **Cancel** to cancel the installation.

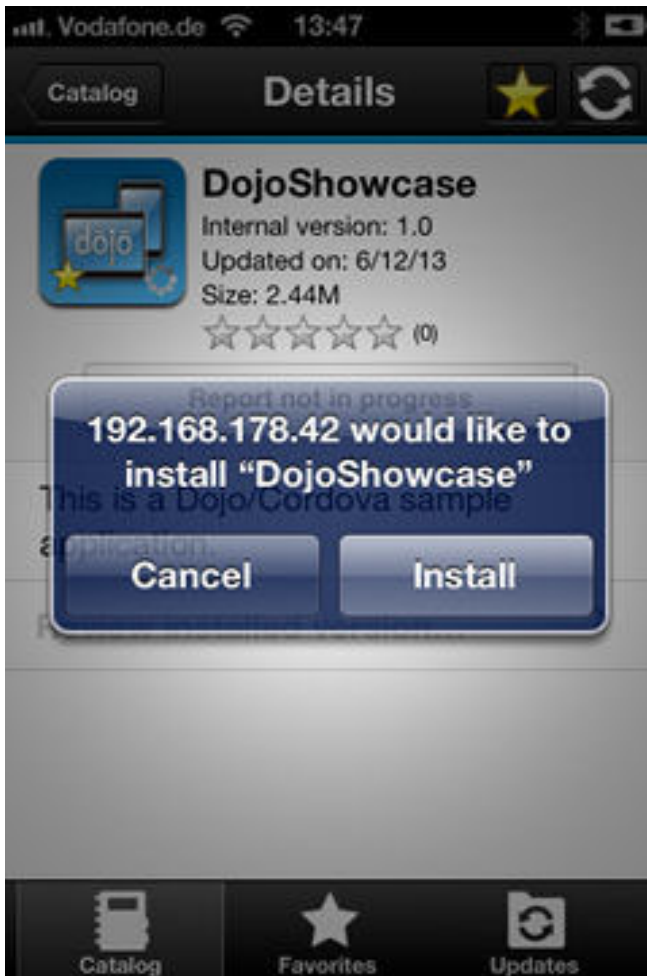


Figure 124. Canceling application installation on your iOS device

Depending on the action taken, the application is installed or not.

- Optional: If you tap **Cancel**, you must notify the application that the installation has been canceled by clicking the **Report Not in Progress** button.

### Installing an application on a BlackBerry device

From the **Details** view, you can install or reinstall an application version on your BlackBerry device.

#### Procedure

- In the **Details** view, tap **Install** or **Reinstall**. You are requested to confirm the download and installation of the application version.
- Optional: During the installation, a progress bar is displayed; tap or click the red cross next to the progress bar to cancel the installation while the application is being downloaded. When the download of the application is complete, the installation can no longer be canceled.



Figure 125. Downloading an application to a BlackBerry device

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.

### Removing an installed application

You can remove an application that is installed on your mobile device.

#### Procedure

1. Initiate the removal procedure that is valid for the operating system of your device.
  - Android: See the procedure in step 2.
  - iOS: Use the normal iOS procedure for removing an application.
  - BlackBerry: See the procedure in step 3.
2. **Android only:** Remove an application from an Android device.
  - a. In the **Details** view of any version of the application, tap **Uninstall**. The **Uninstall** button appears in the **Details** view only when a version of the application is installed. You are requested to confirm that the application version is to be uninstalled.
  - b. Tap **Uninstall** to uninstall the application version or **Cancel** to notify the application that the uninstallation command has been canceled.
3. **BlackBerry only:** Remove an application from a BlackBerry device.
  - a. In the **Details** view of any version of the application, tap **Uninstall**. The **Uninstall** button appears in the **Details** view only when this version of the application is installed. You are requested to confirm that the application version is to be uninstalled.
  - b. Tap **Uninstall** to uninstall the application version or **Cancel** to cancel the uninstallation command. 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.

### Showing details of a specific application version

Select a version of an application to show its details.

## About this task

You can show the details of the selected version of an application by following the appropriate procedure for an Android or iOS phone, a BlackBerry phone, or a tablet.

### Procedure

1. Show details of a specific application version on a mobile device by selecting the appropriate procedure to follow for your device.
  - An Android or iOS phone; see step 2.
  - A BlackBerry phone; see step 3.
  - A tablet; see step 4 on page 823.
2. **Android and iOS only:** Show details of a specific application version on an Android or iOS phone.
  - a. Tap **Select a version** to navigate to the version list view.

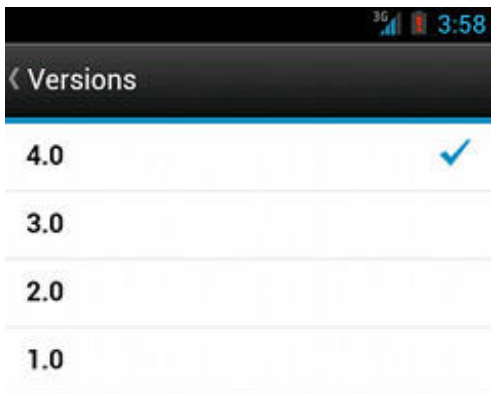


Figure 126. Specific version of an application selected in the list of versions on an Android or iOS phone

- b. Tap the required version of the application. The **Details** view is updated and shows the details of the selected application version.
3. **BlackBerry only:** Show details of a specific application version on a BlackBerry phone.
    - a. Slide to the **Versions** pane.



Figure 127. List of versions of an application on a BlackBerry phone

- b. Tap the required version of the application. The **Details** view is updated and shows the details of the selected application version.
  4. **Tablet devices only:** Show details of a specific application version on a tablet.
    - a. Tap **Select version**.
    - b. In the pop-up menu, select the required version of the application. The **Details** view is updated and shows the details of the selected application version.

### Updating an application

You can update an application that is installed on your device if a new version is available in the Application Center.

#### About this task

Follow this procedure to make the latest versions of favorite and recommended apps available on your device. Applications that are marked as favorites and that have an updated version are listed in the **Updates** view. The applications that are marked as recommended by the Application Center server administrator are also listed in the **Updates** view, even if they are not favorites.

If a more up-to-date version of an installed application is available on the server, it is listed under **Update or Recommended**.

#### Procedure

1. In the **Updates** view, navigate to the **Details** view.
2. In the **Details** view, select a newer version of the application or take the latest available version.
3. **Android and BlackBerry only:** On Android and BlackBerry devices, tap **Update**.
4. **iOS only:** On iOS devices, tap **Install latest**.
5. Follow the appropriate application installation procedure.
  - “Installing an application on an Android device” on page 817
  - “Installing an application on an iOS device” on page 819
  - “Installing an application on a BlackBerry device” on page 820

## Reverting an installed application

You can revert the version of an installed application if an earlier version exists on the server.

### Purpose

To replace the currently installed version of an application with an earlier version, from the **Catalog**, **Updates**, or **Favorites** view, navigate to the **Details** view. In the **Details** view, select an earlier version. See “Showing details of a specific application version” on page 821 for information about how to display details of a specific application version on a mobile device.

See “Preparations for using the mobile client” on page 772 for information about how to disable reverting to earlier versions of an application.

### On Android

If the installed version of the Android operating system is earlier than 4.2.2, tap **Revert**.

If the installed version of the Android operating system is 4.2.2 or later, you must uninstall the version currently installed before you can install the earlier version.

Then, follow the procedure documented in “Installing an application on an Android device” on page 817.

### On iOS

Use the normal procedure of the operating system to remove the application.

Tap **Install** to install the earlier version of the application. Follow the procedure documented in “Installing an application on an iOS device” on page 819.


### On BlackBerry

Tap or click **Revert**. Follow the procedure documented in “Installing an application on a BlackBerry device” on page 820.

## Marking or unmarking a favorite app

Mark your favorite apps or unmark an app to have it removed from the favorites list.

An application marked as a favorite on your device indicates that you are interested in this application. This application is then listed in the list of favorite apps to make locating it easier. This application is displayed on every device belonging to you that is compatible with the application. If a later version of the app is available in the Application Center, the application is listed in the **Updates** view.

To mark or unmark an application as a favorite app, tap the Favorites icon  in the header of the **Details** view.

An installed application is automatically marked as a favorite app.



## Submitting a review for an installed application

You can review an application version installed on your mobile device; the review must include a rating and a comment.

### About this task

You can only submit a review of a version of an application if that version is installed on your mobile device.

### Procedure

1. In the **Details** view, initiate your review:
  - On Android and iOS phones, tap **Review version X** to navigate to the review screen.
  - On BlackBerry phones, slide to the **Reviews** pane and select **Write Review**.
  - On Android and iOS tablets, tap **Review version X** to show the screen for entering your review.
2. Enter a nonzero star rating:
  - On mobile devices with touch screens, tap a star, from 1 to 5, to represent your approval rating of the version of the application.
  - On BlackBerry devices without touch screen, use the trackpad to slide and select the number of stars.

One star represents the lowest level of appreciation and five stars represent the highest level of appreciation.

3. Enter a comment about this version of the application.
4. Tap **Submit** to send your review to the Application Center.

### Viewing reviews

You can view reviews of a specific version of an application or of all versions of an application.

### Purpose

To view reviews of application versions; reviews are displayed in descending order from the most recent review. If the number of reviews fills more than one screen, tap **Load more** to show more reviews. On Android and iOS devices, the review details are visible in the list. On BlackBerry devices, select a review to view the review details.

### Viewing reviews of a specific version

The **Details** view always shows the details of a specific version. On a phone, the reviews are for that version.

In the **Details** view of an application version:

#### On an Android or iOS phone

Tap **View Reviews** to navigate to the **Reviews** view.

#### On a BlackBerry phone

Slide to the **Reviews** pane.

#### On a tablet


Tap **Reviews xx**, where *xx* is the displayed version of the application.

## Viewing reviews of all versions of an application

In the **Details** view of an application version:

### On an Android or iOS phone

Tap **View Reviews** to navigate to the **Reviews** view. Then tap the settings

icon , tap **All versions**, and confirm the selection.

### On a BlackBerry phone

Viewing reviews of all versions is only available when the details of the latest version are displayed. This action is not available when the details of another specific version are displayed. Slide to the **Reviews** pane, select the settings icon, select **All versions**, and confirm the selection.

### On a tablet

Tap **All Reviews**.

## 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).

**Note:** BlackBerry OS 10 is not supported by the current version of the Application Center.

### 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.

See <http://supportforums.blackberry.com/t5/Java-Development/Network-Transports/ta-p/482457> for more details.

Table 217. 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=172.0.0.1;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

Table 217. 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 review feature called from another application (advanced feature)

You can send reviews to the server from another IBM Worklight application.

The application version review feature described in “Submitting a review for an installed application” on page 825 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 review 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 review and tap **Submit** to send the review to the server.

## API parameters

The same API parameters are required on both Android and iOS operating systems.

Parameters:	
<b>pkg</b>	- The application package ID. Mandatory.
<b>version</b>	- The application package version; if unspecified, the version installed. Optional.
<b>stars</b>	- A value between 0 and 5 representing the number of stars of the rating. Optional.
<b>comment</b>	- 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:

<b>pkg</b>	is mandatory.
------------	---------------

<b>version</b>	is optional.
<b>stars</b>	is optional.
<b>comment</b>	is optional.

## Example of use of the API on iOS

`ibmapctrfeedback://com.TestFeedbackApp/2?stars=1&comment=Good`

---

## Federal standards support in IBM 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 V5.0.6 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 Worklight Server, and SSL communications with the 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. The cryptographic modules are also used for the cryptographic operations that are performed by the applications by using the Java™ Cryptography Extension (JCE). Since the 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 a 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.

**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 Chapter 3, "Tutorials and samples," on page 25.

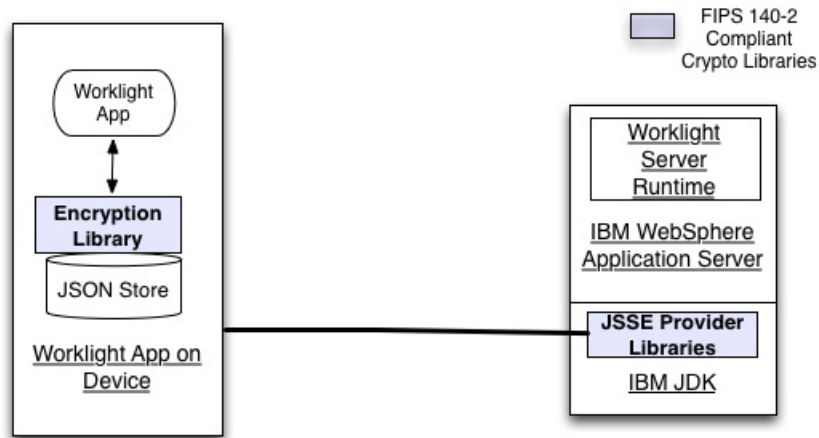
**Note:** There are some restrictions to be aware of:

- 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 Chapter 3, "Tutorials and samples," on page 25.

Figure 128. Example





For more information about JSONStore, see “JSONStore overview” on page 393.

## 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 environment to use the FIPS 140-2 validated modules. For more information, see Java Secure Socket Extension (JSSE) IBMJSSE2 Provider Reference Guide.

---

## Chapter 10. Monitoring, reports, and operational analytics

IBM Worklight includes a range of operational analytics and reporting mechanisms for collecting, viewing, and analyzing data from your IBM Worklight applications and servers, and for monitoring server health.

---

### 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

Worklight Server uses the standard `java.util.logging` package. By default, all IBM Worklight logging goes into the application server log files. You can control Worklight Server logging by using the standard tools available in each application server. If, for example, you want to activate trace logging in Liberty, add a trace element to the `server.xml` file. To activate trace logging in WebSphere Application Server, use the logging screen in the console and enable trace for IBM Worklight logs. IBM Worklight logs all begin with "com.worklight".

For more information about the logging models of each server platform, including the location of the log files, see the documentation for the relevant platform, as shown in the following table.

Table 218. Documentation for different server platforms

Server platform	Location of documentation
Apache Tomcat	<a href="http://tomcat.apache.org/tomcat-7.0-doc/logging.html#Using_java.util.logging_(default)">http://tomcat.apache.org/tomcat-7.0-doc/logging.html#Using_java.util.logging_(default)</a>
WebSphere Application Server Version 7.0	<a href="http://pic.dhe.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.base.doc/info/aes/ae/ttrb_trcover.html">http://pic.dhe.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.base.doc/info/aes/ae/ttrb_trcover.html</a>
WebSphere Application Server Version 8.0	<a href="http://pic.dhe.ibm.com/infocenter/wasinfo/v8r0/topic/com.ibm.websphere.base.doc/info/aes/ae/ttrb_trcover.html">http://pic.dhe.ibm.com/infocenter/wasinfo/v8r0/topic/com.ibm.websphere.base.doc/info/aes/ae/ttrb_trcover.html</a>
WebSphere Application Server Version 8.5 Full Profile	<a href="http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.multiplatform.doc/ae/ttrb_trcover.html">http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.multiplatform.doc/ae/ttrb_trcover.html</a>
WebSphere Application Server Version 8.5 Liberty Profile	<a href="http://publib.boulder.ibm.com/infocenter/radhelp/v8r5/topic/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/rwlp_logging.html">http://publib.boulder.ibm.com/infocenter/radhelp/v8r5/topic/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/rwlp_logging.html</a>

#### Log level mappings

Worklight Server uses `java.util.logging`. The logging levels map to the following levels:

- `WL.Logger.debug`: FINE
- `WL.Logger.info`: INFO
- `WL.Logger.warn`: WARNING
- `WL.Logger.error`: SEVERE

## 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 specific application 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 log information for auditing adapter calls, activate the audit logs by setting `audit="true"` in your `adapter.xml` file in the procedure definition.

## 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 for checking server health

Use IBM Worklight vitality queries to run a health check of your server, and determine the vitality status of your server.

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 219. Examples of queries that help determine server vitality

Query	Purpose
<code>http://&lt;server&gt;:&lt;port&gt;/&lt;publicWorkLightContext&gt;/ws/rest/vitality</code>	Checks the server as a whole.
<code>http://&lt;server&gt;:&lt;port&gt;/&lt;publicWorkLightContext&gt;/ws/rest/vitality?app=MyApp</code>	Checks the server and the MyApp application.
<code>http://&lt;server&gt;:&lt;port&gt;/&lt;publicWorkLightContext&gt;/ws/rest/vitality?app=MyApp&amp;adapter=MyAdapter</code>	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 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 220. 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

The returning XML contains more attributes, which are undocumented constants that you must not use.

## Configuring logging in the development server

Information about the default logging settings for the embedded development, and procedures for changing them.

When you are trying to diagnose problems in the Worklight Studio embedded test server (for example, when debugging a custom login module), it is important to be able to see log messages. The default settings for server logging are described in this section, along with the procedures for changing them if you must see finer levels of messages.

In previous releases of Worklight Studio, the embedded Jetty test server did not allow viewing the server logs. In Worklight Studio V6.0.0, the test server is replaced with an instance of the WebSphere Application Server Liberty Profile server and is referred to as the Worklight Development Server.

Logging levels for the Worklight Studio plugin and builder can be configured with a new file named `logging.properties`. This file is in the `.metadata` folder of your Eclipse workspace.

For example, if your Worklight Studio workspace is `/usr/workspace` (on UNIX) or `C:\workspace` (on Windows), the corresponding logging configuration file is `/usr/workspace/.metadata/logging.properties` or `C:\workspace\.metadata\logging.properties`.

This file contains the following default settings:

```
handlers = java.util.logging.FileHandler
.level = WARNING
com.worklight.level = INFO
```

### Changing the Worklight Console logging levels

To change the logging level for all packages in this instance of Eclipse, edit the `.level = line`. To change the logging level only for Worklight Studio, edit the `com.worklight.level = line`.

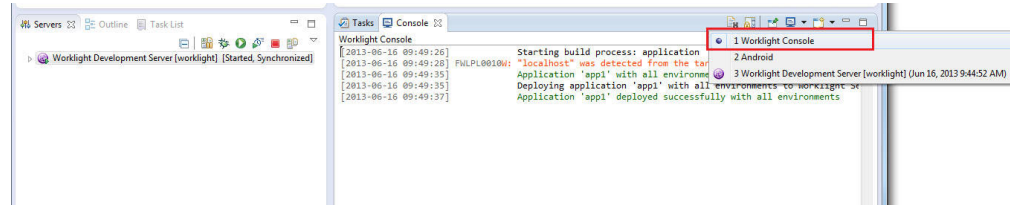
The available setting levels for `com.worklight.level = are:`

- SEVERE (highest value)
- WARNING
- INFO
- CONFIG
- FINE
- FINER
- FINEST (lowest value)

In addition, there is an ALL level that specifies logging of all messages, and an OFF level that turns off logging.

If you edit the `logging.properties` file to change the logging level, you must restart Worklight Studio before the change takes effect.

Whatever the logging level, the messages are displayed in Worklight Studio in its console view with the name **Worklight Console**, as shown in the following screen capture:



## Changing the Worklight Console Server console logging levels

Changing the logging properties for individual application server types is done with those servers' administration tools.

To provide two examples for WebSphere Application Server Liberty Profile, the `server.xml` file can be modified by appending the new logging element:

- To enable the INFO logging level for the server console, the following line is added to the `server.xml` file:
 

```
<logging consoleLogLevel="INFO"/>
```
- To enable trace log files, the following line is added to the `server.xml` file:
 

```
<logging traceSpecification="*=audit=enabled:com.worklight.*=info=enabled" />
```

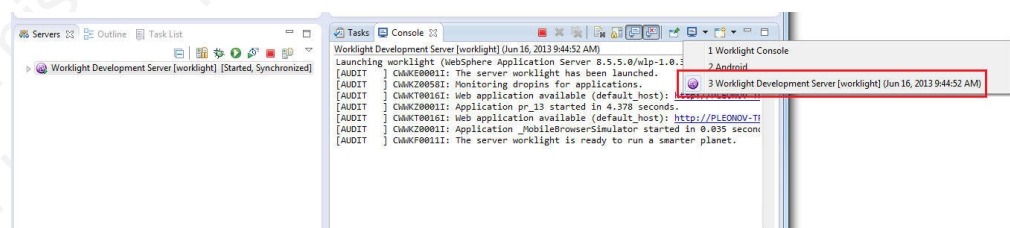
The available setting levels for `consoleLogLevel` are:

- INFO
- AUDIT
- WARNING
- ERROR
- OFF

Please note that this parameter does not support DEBUG level logging.

No server restart is necessary after you modify these settings.

Whatever the logging level, the messages are displayed in Worklight Studio in its console view with the name **Worklight Development Server**, as shown in the following screen capture:



This console view allows you to see messages from the Worklight Development Server, but with some known limitations:

- Localized log messages are shown incorrectly. For more information about this issue, see Liberty profile: Trace and logging.
- Setting the value of `consoleLogLevel` to WARNING, ERROR, or OFF causes the server not to start from Worklight Studio using the Eclipse serversview.

Trace log messages are not displayed in Worklight Studio, and are written to the `trace.log` file only. This logging trace is optional and supports fine-tuning such as packaging and more precise reporting levels, and is mainly used for debugging.

The `trace*.log` file is found under your Eclipse workspace in the folder `WorklightServerConfig\servers\worklight\logs\`.

For example, if your Worklight Studio workspace is `/usr/workspace` (on UNIX) or `C:\workspace` (on Windows), the log files are created under `/usr/workspace/WorklightServerConfig/servers/worklight/logs/` or `C:\workspace\WorklightServerConfig\servers\worklight\logs\`.

The available setting levels for `<logging traceSpecification="*=audit=enabled:com.worklight.*=info=enabled" />` are:

- Off - No events are logged.
- Fatal - Task cannot continue and component cannot function.
- Severe - Task cannot continue, but component can still function.
- Warning - Potential error or impending error.
- Audit - Significant event affecting server state or resources.
- Info - General information outlining overall task progress.
- Config - Configuration change or status.
- Detail - General information detailing subtask progress.
- Fine - Trace information - General trace.
- Finer - Trace information - Detailed trace + method entry / exit / return values.
- Finest - Trace information - A more detailed trace - Includes all the detail that is needed to debug problems.
- All - All events are logged. If you create custom levels, All includes your custom levels, and can provide a more detailed trace than Finest.

For more information about WebSphere Application Server Liberty Profile logging configuration, see Liberty profile: Trace and logging.

## 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.SimpleFormatter
```

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.

---

## Comparison of operational analytics and reports features

Compare the reports and operational analytics features to know why and how to best use each.

With the introduction of the IBM Worklight operational analytics feature, you can continue using the reports feature, use it along with the analytics feature, or just use the analytics feature. A comparison of the capabilities of these two features can help clarify the strengths of each, and help determine how you can best use them.

The data that is collected by the “Reports” on page 859 feature is a subset of the total data that is collected as part of the new “Operational analytics” on page 840 feature. You can use the reports feature and the operational analytics feature simultaneously, but enabling both features causes redundant storage of the data. Use the reports feature in cases where you want direct access to the database to run custom queries. An example of a scenario where direct database access is needed is the use of BIRT or a customized online analytical processing (OLAP) system that runs database queries against the Reports database.

*Table 221. Comparison of analytics and reports features*

	<b>Operational analytics feature</b>	<b>Reports feature</b>
<b>Primary usage</b>	Problem determination, device usage summary, geographic view of mobile activity	Device usage summary
<b>Typical user</b>	Administrator, operational support personnel, developer, analyst	Administrator, analyst
<b>Data used in analytics</b>	App crash from clients, Worklight server log, Worklight app to server interaction activities	Worklight app to server interaction activities

Table 221. Comparison of analytics and reports features (continued)

	Operational analytics feature	Reports feature
Data storage mechanism	Files on the analytics platform	Relational database
Analytics mechanism	Each log event is treated as a JSON document. The data in the document is indexed so that it can be searched by keyword in the document and presented in a canonical form that shows the app, version, some device data, location (if enabled), timestamp, adapter (if present in the document) and other data.	Each log event is treated as a row in the raw Reports database table and then aggregated for statistics into the app_activities database table, summarized to app, device operating system, and timestamp relationships.
Access mechanism	Worklight Console	BIRT or other reporting tools that can understand data cubes
Extendable	Extending the published reports is not supported.	Data can be extracted from the database tables by using any means that you desire, including but not limited to, BIRT.
Search across logs	Yes	No
Optional	Yes	Yes

In addition to an at-a-glance view of your mobile and web application analytics, the operational analytics includes the capability to perform raw search against server logs, client activities, captured client crash data. The operational analytics feature can also search any additional data that you explicitly provide through client and server-side API function calls that feed into the IBM WebSphere Analytics Platform.

## Operational analytics

The operational analytics feature enables searching across apps, services, devices, and other sources to collect data about usage or detect problems.

In addition to reports summarizing app activity, IBM Worklight includes a scalable operational analytics platform accessible in the IBM Worklight Console. The analytics feature enables enterprises to search across logs and events that are collected from devices, apps, and servers for patterns, problems, and platform usage statistics. You can enable analytics, reports, or both, depending on your needs. To understand the similarities and differences between the existing reports feature and the new operational analytics feature, see “Comparison of operational analytics and reports features” on page 839.

The data for operational analytics includes the following sources:

- Crash events of an app on iOS and Android devices.
- Interactions of any app-to-server activity (anything that is supported by the IBM Worklight client/server protocol including push notification).

- Server-side logs that are captured in traditional IBM Worklight log files.

The operational analytics feature is accessible from the IBM Worklight Console and includes these capabilities:

- Interactive web-based usage.

#### **Equivalent Reports feature**

An Analytics Dashboard view with all the same reports as available in the previous version of IBM Worklight with BIRT, with enhanced features. These features include interaction support to see the full device usage across the platform for the last 30, 60 or 90 days, and drill down to specific apps and app versions.

#### **Search view**

The ability to search on free text occurrences across logs for key words and the ability to contextually narrow or expand the search result.

#### **Server Log view**

A scrolling view of the server-side log information in a table form with ability to filter the view by keywords.

#### **Map view**

An aggregate representation of activity by location (latitude and longitude) on a world map, which can be zoomed or panned.

- Near-real time reporting across the various views. The data that is collected from the server or client can be searched for in near-real time. During the search, the IBM WebSphere Analytics platform processes the data and displays the results on the console.

In addition to an at-a-glance view of your mobile and web application analytics, the analytics feature includes the capability to perform raw search against server logs, client activities, captured client crash data, and any additional data you explicitly provide through client and server side API function calls that feed into the IBM WebSphere Analytics Platform.

#### **Related tasks:**

“Installing and configuring the IBM WebSphere Analytics Platform” on page 174  
To run the analytics features, you must install the IBM WebSphere Analytics Platform (IWAP).

## **IBM Worklight analytics components**

The IBM WebSphere Analytics Platform and several libraries make up the components of the IBM Worklight operational analytics feature.

IBM Worklight includes the IBM WebSphere Analytics Platform (IWAP), which is the engine that drives the IBM Worklight scalable operational analytics feature. IWAP is designed for challenges that are introduced by the big data of the mobile environment:

#### **Volume**

Mobile transactions continue to grow as more traffic comes from mobile channel.

#### **Velocity**

Mobile interactions arrive quickly from different areas, depending on user mobility patterns.

#### **Veracity**

With increased volume, there is an increase in noise.

### Variety

With increasing variation in mobile usage patterns, apps, physical device sizes, operating systems, and networks, collected data is increasingly varied as well.

IBM Worklight also includes IBM Tealeaf CX Mobile libraries for iOS and Android devices, and the JavaScript library for mobile web. These libraries are part of the client run time on these platforms and support in client-side collection of analytics data.

The diagram illustrates the high-level topology of the operational analytics feature components. The existing databases and the BIRT reports feature are also shown for context.

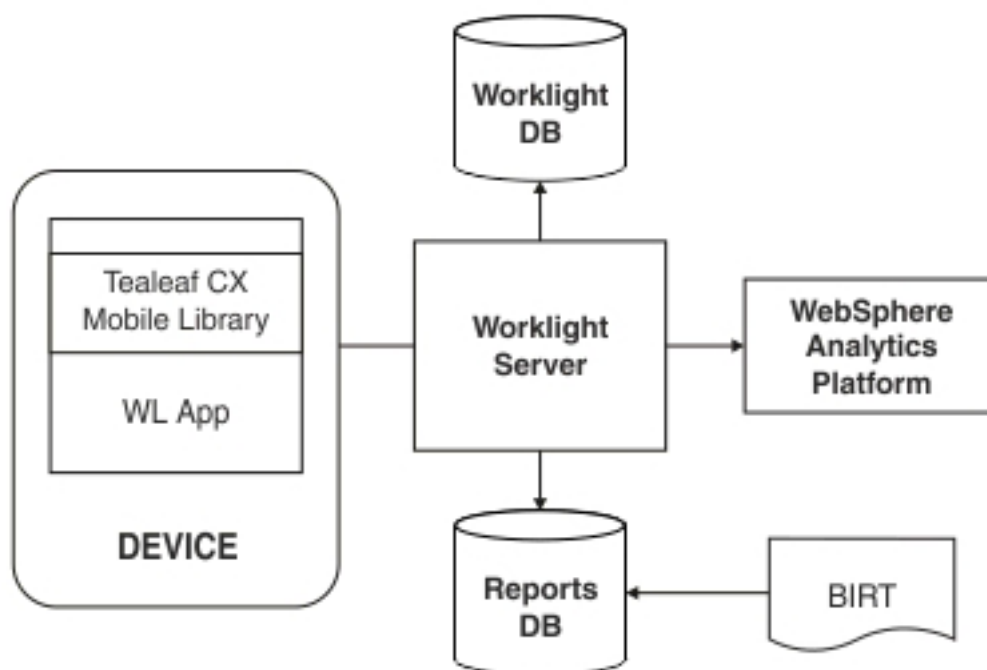


Figure 129. High-level overview of reporting and analytics infrastructure.

**Note:** Enabling reports, using BIRT to query the reports database, and enabling the IBM WebSphere Analytics Platform are each optional.

### Analytics event types

Event types for logged events are recorded and included in search results and reports.

Analytics data is collected in several different contexts in an IBM Worklight client/server infrastructure.

#### App Activity

All IBM Worklight client/server network communication activity.

#### Notification Activity:

Any push notification that is triggered from the backend.

**Server Log:**

From the IBM Worklight server, indicating server activity and log messages.

**Client Session:**

Client side application crashes and any custom analytics log events.

Event types are recorded as part of the log event context, then used in analysis for reporting. When you search for key terms in the Search View under the Analytics tab of IBM Worklight Console for example, event types are included in the search results.

## Analytics page

The Analytics page is the user interface from which you can search for analytics data and see results from your IBM Worklight system.

The Analytics page provides a range of analytics features contained in various views, each accessible from a tab. In addition to seeing a summary of your mobile and web application analytics, you can perform raw searches on the following sources:

- Server logs
- Client activities
- Captured client crash data
- Any additional data that you explicitly provide through client and server-side API function calls that feed into the IBM WebSphere Analytics Platform.

### Dashboard view

The Analytics Dashboard view presents a range of measures about the IBM Worklight system.

The Analytics Dashboard displays the following charts:

- Daily Hits
- Daily Visits
- Active Users
- Environment Usage
- Notifications Per Day
- Notifications Per Source
- New Users

You can read a description for each chart by clicking the icon in the Dashboard view next to the name of the chart.



Figure 130. The Dashboard view is in the Analytics page of the IBM Worklight Console

### Search view

The Search view in the Analytics page enables you to search for logged events from Worklight server.

The IBM Worklight Analytics page Search view also provides more information about the search term:

### Pie graph breakdown

The top search results broken down by source and event type.

### Top common terms

The top 5 most common terms found among the search hits. These terms can be used to find relationships and correlations among search results.

### Timeline of Search Hits

A bar chart displaying the search result hits per day.

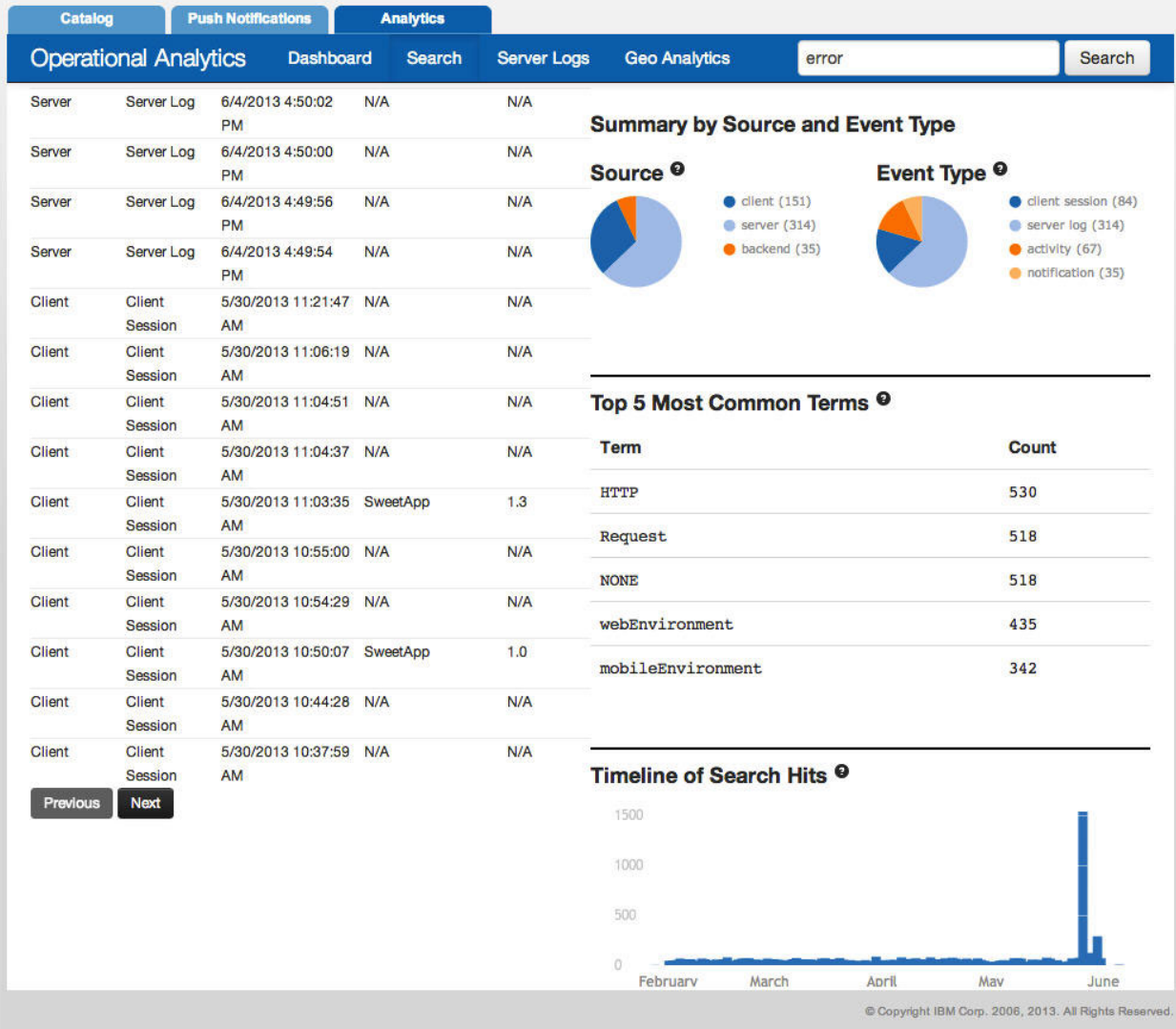


Figure 131. Search results

When you click a search result, the page displays detailed information about the individual hit.



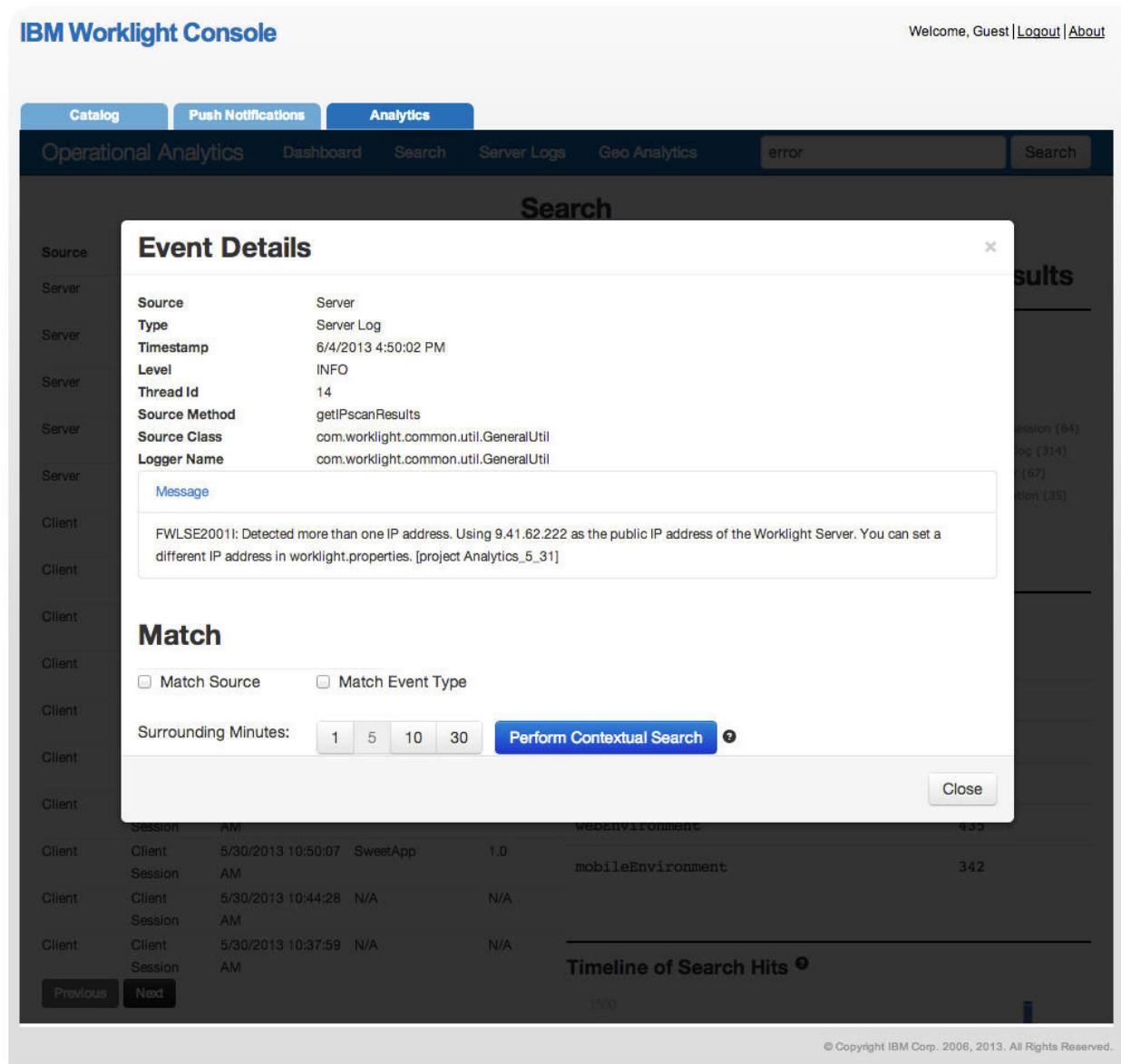


Figure 132. Search results details

You can also run a contextual search on a search hit, to run a new search that will return events that occurred in the same time frame as the original search result. The time frame can be selected by using the button group next to the contextual search button.

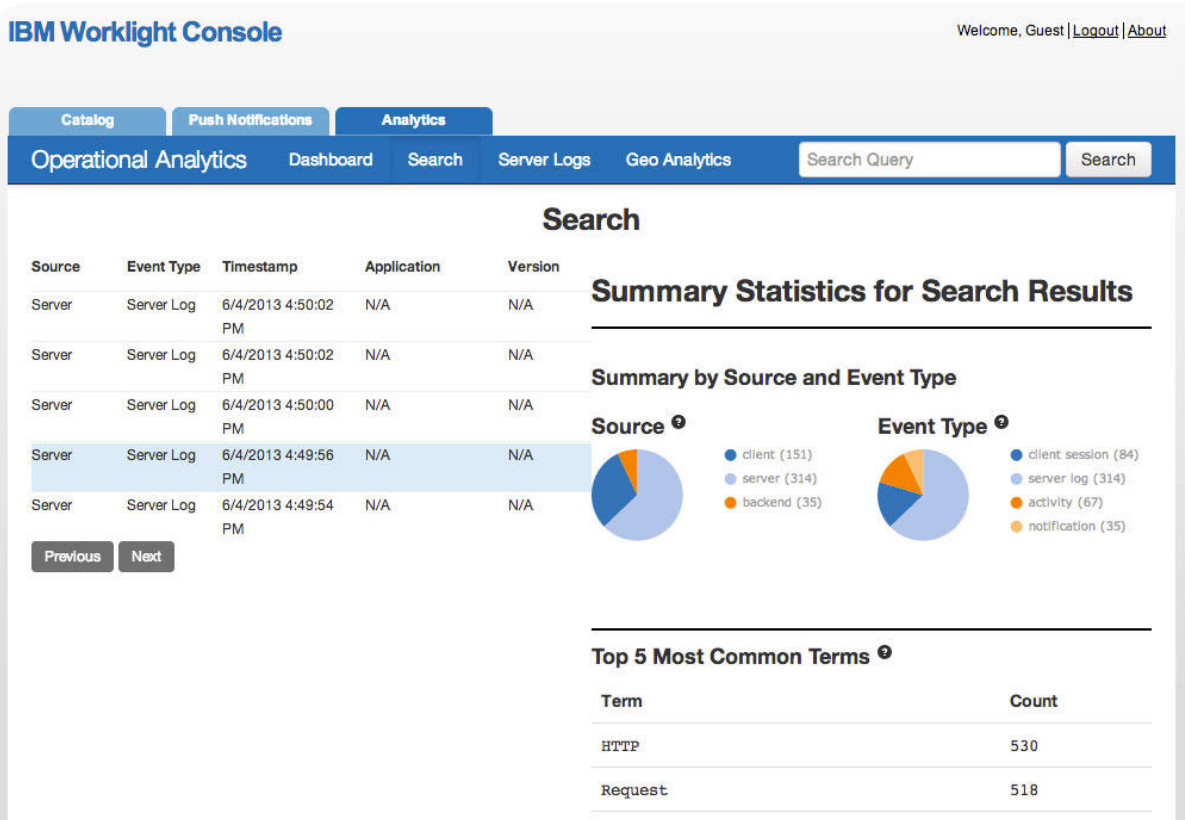


Figure 133. Contextual search with the original search hit highlighted

Contextual search is a powerful feature that can help to diagnose problems on an application, or a server issue. There are two common patterns that are supported in the current implementation of this Analytics feature, basic contextual search and a more advanced search, drilling down to details in the search results.

For a basic contextual search, terms you enter in the search box return matches from the log events that are collected by the IBM WebSphere Analytics Platform. Basic contextual search is useful to get an overview, to determine whether a term or set of terms appears in any of the log events captured. This pattern can be visualized as follows:

**Log data on your IBM  
WebSphere Analytics  
Platform**

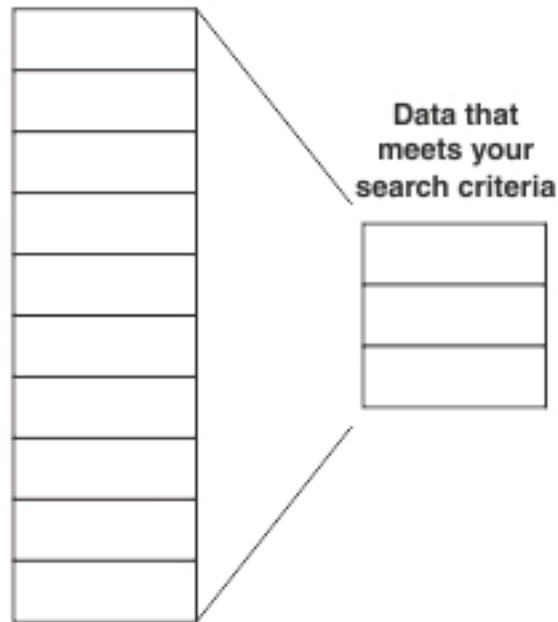


Figure 134. Contextual branching search

After a basic search you can select a log event, view the details, and run refined contextual searches centered on that log entry. Available search refinements include adding a time dimension for example, specifying a number of minutes surrounding the log event to find what other log events occurred in that time frame. You can also limit the context to either an event source (client, server, or backend) or an event type. This pattern is illustrated in the following diagram.

**Log data on your IBM  
WebSphere Analytics  
Platform**

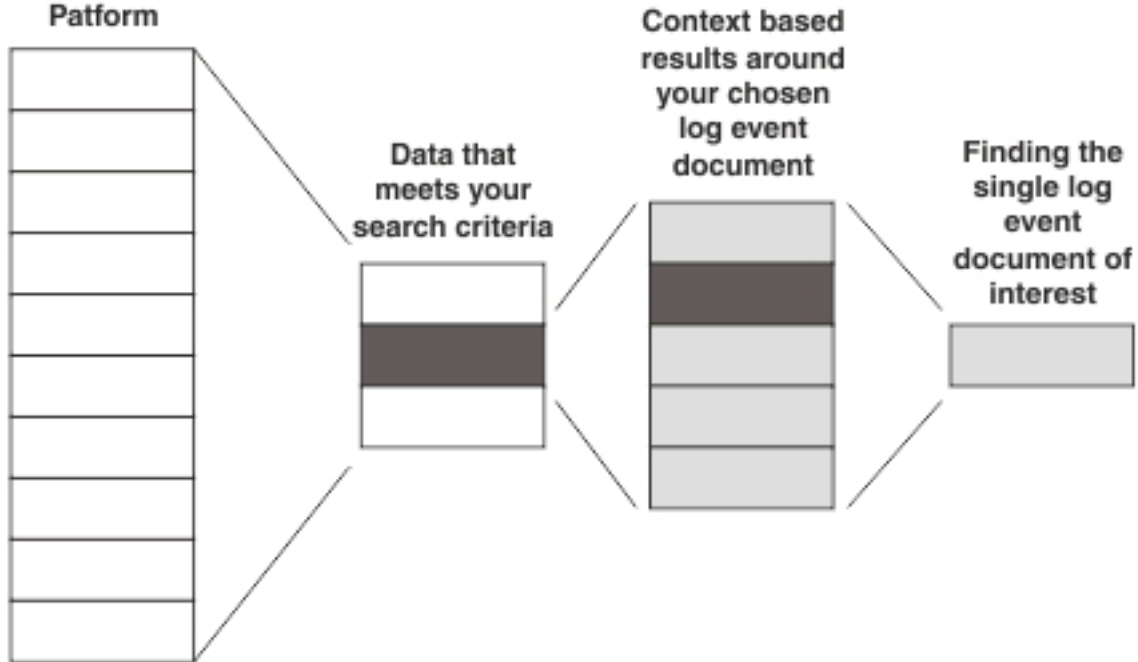


Figure 135. Context search drill-down

### Server Logs view

Worklight Server logs are visible in the Sever Logs view of the Analytics page in real time.

By default, Worklight Server forwards all server logs to the Analytics page Sever Logs view, and the logs can be viewed in real time. You can adjust the rate at which Worklight Server logs are retrieved and displayed. You can also filter the logs by keyword or by severity.

### Server Logs

Refresh Rate (seconds): 5 10 15 20 Pause

Timestamp	Thread Id	Severity	Source Class	Method	Message	View Stack
Monday, July 15, 2013 5:52 PM	7	WARNING	com.worklight.core.auth.impl.Auth	GCMMediator	AnalyticsServiceImplLoadProperties	<a href="#">View Stack</a>
Monday, July 15, 2013 5:53 PM	13	FINE	com.worklight.core.auth.impl.Auth	AuthenticationContext	AnalyticsServiceImplLoadProperties	<a href="#">View Stack</a>
Monday, July 15, 2013 5:54 PM	0	FINE	com.worklight.common.util.General	APNSMediator	AnalyticsServiceImplLoadProperties	<a href="#">View Stack</a>
Monday, July 15, 2013 5:55 PM	17	INFO	com.worklight.core.auth.impl.Auth	extractDbTypeFromConnection	AnalyticsServiceImplLoadProperties	<a href="#">View Stack</a>
Monday, July 15, 2013 5:56 PM	12	INFO	com.worklight.integration.notification	AuthenticationContext	found IP address:/9.41.54.146	<a href="#">View Stack</a>
Monday, July 15, 2013 5:56 PM	17	SEVERE	com.worklight.core.auth.impl.Auth	GCMMediator	found IP address:/9.41.54.146	<a href="#">View Stack</a>
Monday, July 15, 2013 5:57 PM	7	INFO	com.worklight.common.util.General	APNSMediator	found IP address:/9.41.54.146	<a href="#">View Stack</a>
Monday, July 15, 2013 5:57 PM	17	WARNING	com.worklight.common.util.General	extractDbTypeFromConnection	worklight: FWLSE3005: Application raw reports are disabled.	<a href="#">View Stack</a>
Monday, July 15, 2013 5:57 PM	9	WARNING	com.worklight.core.auth.impl.Auth	extractDbTypeFromConnection	AnalyticsServiceImplLoadProperties	<a href="#">View Stack</a>
Monday, July 15, 2013 5:58 PM	11	SEVERE	com.worklight.integration.notification	extractDbTypeFromConnection	AnalyticsServiceImplLoadProperties	<a href="#">View Stack</a>
Monday, July 15, 2013 5:58 PM	13	WARNING	com.worklight.integration.notification	AuthenticationContext	worklight: FWLSE3005: Application raw reports are disabled.	<a href="#">View Stack</a>
Monday, July 15, 2013 5:58 PM	19	WARNING	com.worklight.core.auth.impl.Auth	AuthenticationContext	AnalyticsServiceImplLoadProperties	<a href="#">View Stack</a>
Monday, July 15, 2013 5:59 PM	5	WARNING	com.worklight.core.auth.impl.Auth	getStringProperty	AnalyticsServiceImplLoadProperties	<a href="#">View Stack</a>

Figure 136. The Server Logs view.

### Geolocation view

To view geospatial and WiFi information in your client data to be analyzed, you must configure clients to include it.

You can use the IBM Worklight Location Services feature to gather geospatial and WiFi information from the analytics data collected. You can then view the results in the IBM Worklight Console Analytics page, in the Geolocation view. Geospatial information can include:

- Latitude
- Longitude
- Speed
- Direction

WiFi information can include:

- Available hotspots
- SSID

Follow the Location Services documentation to enable the automatic appending of geospatial and WiFi data to analytics events. When the location services feature is enabled, geospatial and WiFi data is recorded in the app activity and client session analytics events automatically. A map displays the latitude and longitude for each client that records analytics data.

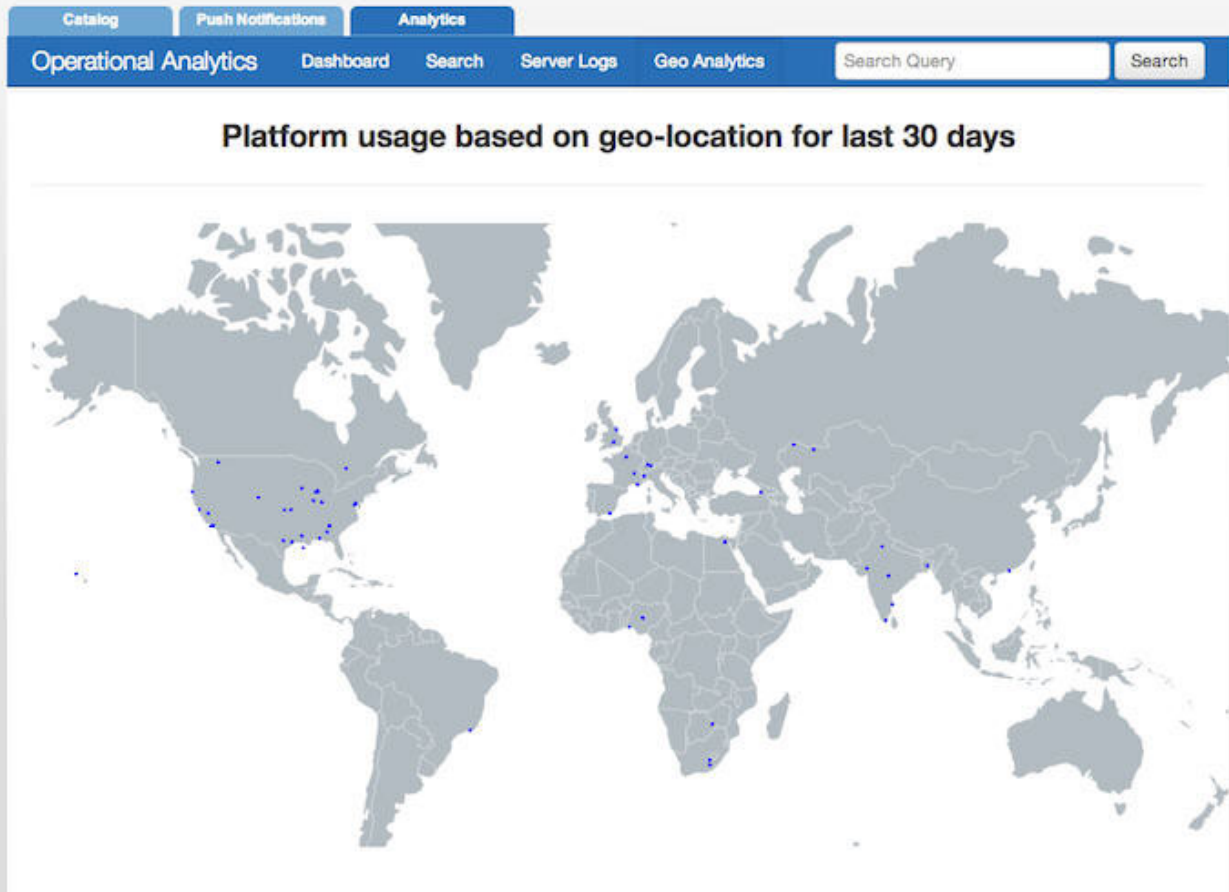


Figure 137. Geospatial information is displayed for clients that are configured to include it

## Capturing client analytic data

Learn about settings to send client application analytic data to Worklight Server.

By default, the following code snippet is populated into a new IBM Worklight application in the `initOptions.js` file:

```
var wlInitOptions = {  
  //Other key/value pairs not related to Analytics  
  analytics : {enabled: false}  
};
```

Notice the default value of `analytics.enabled` is `false`. To enable sending captured application crash data, and analytic data recorded by `WL.Analytics.log` function call, you must either set the option to `true` or call `WL.Analytics.enable` function.

In IBM Worklight, client application crashes are recorded and analytic data are sent to Worklight Server when the client analytic feature is enabled. If Worklight `sServer` is properly configured, this client application crash data is then forwarded

to the IBM WebSphere Analytics Platform. No additional client-side application code, or any client application configuration, is necessary to enable the client-side crash capture.

Captured crash data is stored persistently at the time of the crash and sent to the server at the next application start, after successful connection to Worklight Server.

**Note:** No data is sent to Worklight Server until the application is connected to the server. This can be achieved by setting `connectOnStartup : true` in the `wlInitOptions` object in `initOptions.js` (which is found in `[app folder]/common/js`), or is done automatically on the first successful call to an adapter in Worklight Server.

The full `WL.Analytics` API is described in “`WL.Analytics`” on page 491.

## Analytics data flow

Understanding the analytics data flow can help you adjust how much data is buffered or discarded.

Worklight Server can pass large amounts of data to the IBM WebSphere Analytics Platform. The amount of data collected depends on:

- The number of Worklight Server cluster members.
- The number of clients.
- How frequently clients communicate with the server, which depends on these factors:
  - The `connectOnStartup: true` setting.
  - The Location Services feature configuration.
  - The frequency of adapter calls.
- The usage of the client-side APIs to record more custom analytics data.
- The usage of the server-side API to record more activities.

IBM Worklight application clients send collected analytic data to Worklight Server. Depending on the origin of the data, it can be buffered on the clients prior to sending to best preserve battery life and network usage. Worklight Server forwards data to IBM WebSphere Analytics Platform by using HTTP POST. If this connection experiences socket timeouts or other failure, the data is discarded.

When the server is unusually busy, Worklight Server buffers data that is designated for IBM WebSphere Analytics Platform. You can tune IBM Worklight to adjust this data loss, taking advantage of the memory capacity of Worklight Server and speed of the network connection from Worklight Server to IBM WebSphere Analytics Platform. For more information see “Worklight Server throughput tuning” on page 854.



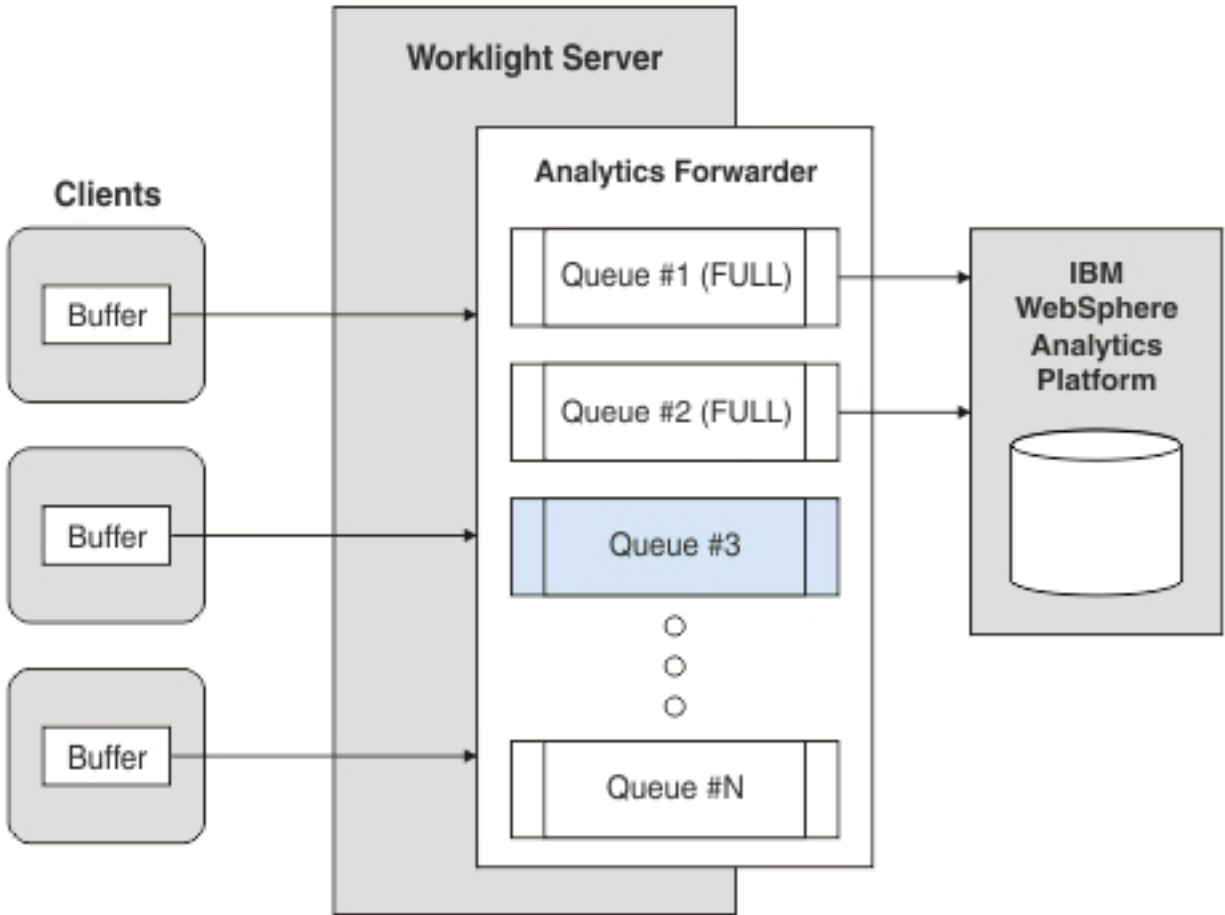


Figure 138. Architectural overview of analytic data throughput.

## Managing data throughput and accumulation

You can make configuration changes to adjust how data accumulates in IWAP.

The amount of data that is accumulated by the IBM WebSphere Analytics Platform depends on a number of factors such as:

- The log level that is configured on the server.
- How frequently client applications connect to Worklight Server
- How frequently client applications are calling the `WL.Analytics.log` function.
- How frequently client applications are calling the `WL.Client.logActivity` function.
- How the client application is used, for example how often it is brought into the foreground and how long it remains there.
- Whether the IBM Worklight Location Services feature is enabled, and how it is configured.
- The number of clients.
- Whether clients crash and how frequently they do so.
- How frequently server adapters are calling the `WL.Server.logActivity` function.

For example, consider a client application with the following characteristics:

- Has no additional customization or features enabled.
- Is used five times a day.
- Stays in the foreground for three minutes each time it is used.

In this scenario, the IBM WebSphere Analytics Platform accumulates approximately 35 KB of data per day that is related to this application. If in addition, the application makes 10 calls to the `WL.Analytics.log` function each time it enters the foreground, the IBM WebSphere Analytics Platform accumulates approximately 91 KB of data per day.

The search index is configured for 300 GB by default. This size is configurable by using the `updateTenant.py` script, as follows:

```
<INSTALL_LOCATION>/searchengine/python updateTenant.py -t worklight -p <password for tenant> -v <des
```

The search engine fails to respond if the disk on the server that is hosting the IBM WebSphere Analytics Platform is full. You can prune the data in the search index by using the `pruneData.py` script, which deletes older data when the search index exceeds the defined volume (300 GB by default, or the size that you specified by using `updateTenant.py`). You can run the `pruneData.py` script periodically by using the following command:

```
python <INSTALL_LOCATION>/searchengine/pruneData.py
```

You can also schedule the `pruneData.py` to run periodically by using `crontab` or other similar time-based service features, by creating a shell script that starts the command and schedules the script to run at a specified time.

### Worklight Server throughput tuning

You can tune the amount of data that is held or deleted from queues to balance the risk of data loss against the risk of overloading your server.

You can use the following two parameters in the JNDI configuration for your application WAR for throughput tuning:

- `wl.analytics.queues`
- `wl.analytics.queue.size`

`wl.analytics.queues` determines the maximum number of queues that the Worklight Server holds in memory. When the maximum number of queues is reached and all queues are full, Worklight Server drops the most recently received data.

`wl.analytics.queue.size` is the number of individual elements each queue can hold. Adjustment of these parameters affects:

- Memory that is consumed by the server.
- Frequency of POSTs to the IBM WebSphere Analytics Platform

The number of individual analytics events the server will hold at one time is `wl.analytics.queues * wl.analytics.queue.size`. Take this into consideration when defining these two variables. Setting them too low can cause large amounts of analytics data to be dropped if the server is unusually busy. Setting them too high can consume too much memory on Worklight Server.

### Turning off server log forwarding to IWAP

You can turn off server log forwarding if too much data is accumulating.

By default, Worklight Server forwards all server-side log data to the IBM WebSphere Analytics Platform to increase search and visibility of log data. This means that an administrator does not have to gain direct file system access to Worklight Server to inspect logs. Depending on the server activity level and Worklight Server cluster size, it can sometimes be better to disable this feature. You can disable the feature by setting the following flag in the JNDI configuration for your WAR: `wl.analytics.logs.forward=false`. The result is that Server Logs are not forwarded to the IBM WebSphere Analytics Platform.

## Enabling analytics for existing applications

You must modify applications that were created in earlier versions of IBM Worklight to enable analytics features.

Applications that you migrate from IBM Worklight V5.0.6 or earlier get most updated features automatically when the IBM Worklight V6.0.0 upgrader updates the applications. To enable analytics however, you must add the following lines of code manually to the `wlInitOptions` object found in `[appFolder]/common/js/initOptions.js`:

```
analytics : {
  enabled: true
  //url : //
}
```

Uncomment the `url` property and provide your server URL information if you send data to a server other than Worklight Server.

You must update the JNDI configuration for your WAR to set the other required properties, as explained in “Configuring Worklight Server for analytics” on page 176.

**Note:** Applications that use IBM Worklight 5.0.6 or earlier that contained Tealeaf client libraries must be manually upgraded into a new Worklight 6.0 project. For more information about how to migrate applications, see Migration of projects using Tealeaf libraries

## Logging additional data

You can use activity log settings to expand the type of analytics data that you collect.

There are two API calls on the client that you can use to accumulate more analytics data:

- `WL.Client.logActivity(String)`
- `WL.Analytics.log(Object, String)`

You can use the “`WL.Client.logActivity`” on page 516 function call to record any activity that you specify, and send the data to the IBM WebSphere Analytics Platform server (if configured), and the raw reports database (if configured).

“`WL.Analytics.log`” on page 492 is a richer way to get data into the IBM WebSphere Analytics Platform than “`WL.Client.logActivity`” on page 516. Using “`WL.Analytics.log`” on page 492 you can pass any JavaScript object as the first parameter, and optionally a descriptive string as the second parameter. This data is sent to the IBM WebSphere Analytics Platform (if configured), but not to the raw reports database.

**Note:** The JavaScript object that you provide as a parameter is indexed and therefore searchable on the IBM WebSphere Analytics Platform Console.

“WL.Analytics.log” on page 492 uses fewer of the battery and WiFi resources of your hybrid application, accumulates in a client-side queue, and is sent over the network to the server after a queue threshold is reached. “WL.Client.logActivity” on page 516 sends instantly when called. If you want to record large amounts of custom analytics data, use “WL.Analytics.log” on page 492

Table 222. Activity log settings

	<b>WL.Analytics.log</b>	<b>WL.Client.logActivity</b>
Destination	IWAP only	IWAP and BIRT
HTTP POST	One POST per ten invocations (default)	One POST per invocation
Data	Any JavaScript object and description	Description only

An example use of WL.Analytics.log is as an indirect callback for the WL.Logger API so that WL.Logger data is collected and sent to the IBM WebSphere Analytics Platform, where it is indexed and searchable. This process is a convenient way to remotely view client-side logs in close to real time. See the WL.Logger object API for information about setting the callback.

### Custom server app activity

There is one API call on the server, in an adapter, that you can use to record more analytics data: WL.Server.logActivity(String) It provides functionality similar to WL.Client.logActivity, but on the server side. You might want to use WL.Server.logActivity to record events or activities that occur within the scope of your IBM Worklight adapter execution lifecycle.

## Integrating with the IBM Tealeaf CX Mobile server

Tealeaf client libraries are pre-integrated in IBM Worklight applications, which means that IBM Worklight applications can be considered *Tealeaf-ready*.

IBM Tealeaf CX Mobile provides digital customer experience management and behavior analysis solutions for companies that do business online. This helps companies to better understand the purpose of customer online and mobile interactions, to be able to enhance the customer experience.

**Note:** IBM Tealeaf server is not included in the IBM Worklight product offering. It is a separately purchasable product in the IBM Mobile First portfolio of products

These are the main benefits of using IBM Tealeaf CX Mobile:

- Discover previously unknown site experience problems so you can improve success rates and increase online revenue.
- 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.

IBM Worklight includes IBM Tealeaf CX Mobile libraries on the iOS and Android devices and JavaScript library for mobile web. IBM Tealeaf CX is part of the client run time on these platforms and supports client-side collection of analytics data. IBM Worklight uses this library to collect app crash events.

If your enterprise would like to enhance analytics capabilities by using IBM Tealeaf CX Mobile, visit the IBM Tealeaf CX Mobile site for details.

IBM Tealeaf CX can be extended to send specific client events to IBM Worklight to support the operational analytics features. The diagram illustrates one scenario in which analytics information, including more than just crash events is collected from the client. This information is then processed by the IBM Tealeaf CX Mobile services, and only the crash events sent to IBM Worklight to support the operational analytics feature.

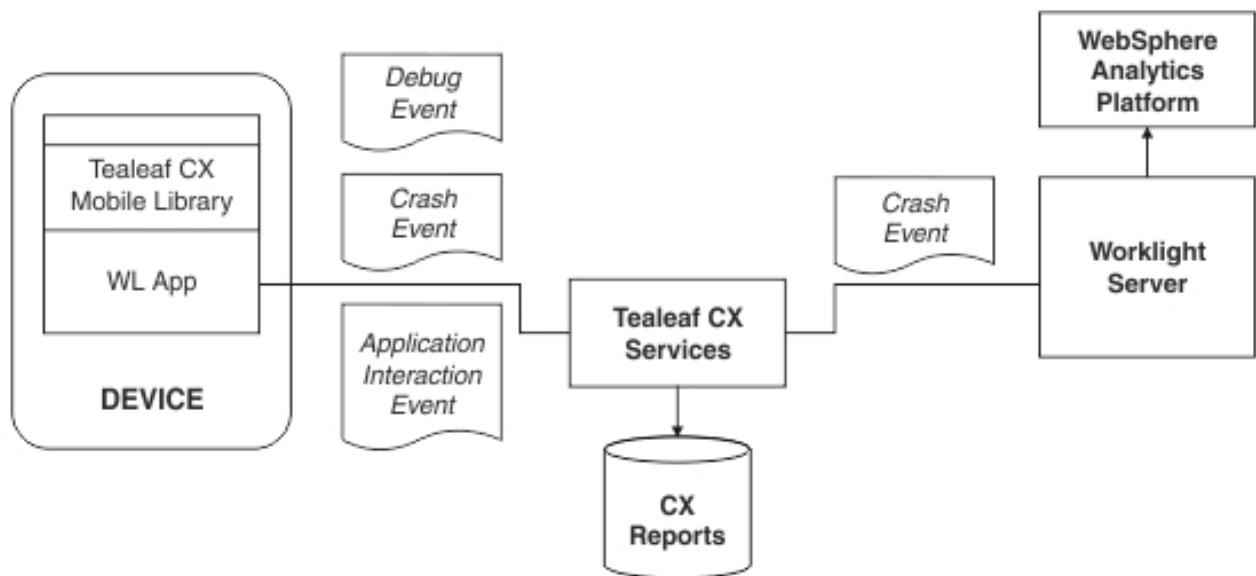


Figure 139. A possible configuration showing Tealeaf CX Mobile service collecting many client session events, and forwarding only crash events to Worklight Server.

### Accumulating data on IBM Tealeaf CX Mobile and IWAP

You can configure client applications to send data to both IBM Tealeaf CX Mobile and IWAP

To collect data on both IBM Tealeaf CX Mobile server and IBM WebSphere Analytics Platform, client applications must be configured to send “WL.Analytics.log” on page 492 (client session) data to the Tealeaf CX Mobile server URL. Calls to “WL.Client.logActivity” on page 516 and WL.Server.logActivity (app activity) are never forwarded to Tealeaf CX Mobile, and continue to accumulate data in the IBM WebSphere Analytics Platform and in the raw reports database, if configured in each.

To reset an IBM Worklight application to send analytics data to Tealeaf CX Mobile requires only one step, which is to call: “WL.Analytics.enable” on page 492({url: “http://yourTealeafServer.com”});

If the client application is already deployed and active among your users, you must call: "WL.Analytics.restart" on page 493({url: "http://yourTealeafServer.com"});

These calls result in your IBM Worklight application persisting the new setting until the user clears their application data, or another invocation to the same function is made with a different URL than the previous invocation. If you want the data sent to the Tealeaf CX server to also display in the Analytics tab in the Worklight Console, the Tealeaf CX Mobile server must be configured with an appropriate pre-processing script so that it can forward to Worklight Server. This is an advanced configuration.

### Accumulating data on IBM Tealeaf CX Mobile only

You can configure client applications to send data only to IBM Tealeaf CX Mobile and not to IWAP.

You can use the IBM Tealeaf CX Mobile client side libraries directly, instead of through the WL.Analytics API. Follow the Tealeaf CX Modile documentation to provide your own properties file (for Android), plist file (for IOS), or configuration object (for JavaScript), and access the respective Tealeaf CX Mobile libraries directly. Do not use the WL.Analytics API in this configuration.

**Note:** In this configuration, neither crash capture or WL.Analytics.log data is indexed or searchable on the IBM WebSphere Analytics Platform. The IBM Worklight application developer now has full control of the client Tealeaf CX Mobile integration.

## Troubleshooting analytics

Find solutions to problems with IBM Worklight analytics features.

Table 223. Analytics troubleshooting guidelines

Problem	Actions to take
WL.Analytics.log data is not searchable in the Analytics tab on the IBM Worklight Console.	<p>"WL.Analytics.log" on page 492 data is not sent until IBM Worklight authentication channels between the client and server are open.</p> <ul style="list-style-type: none"> <li>• Ensure that client apps have successfully authenticated with Worklight Server.</li> <li>• For WL applications that do not use authentication, the initialization sequence must be complete, which can be achieved by setting <code>connectOnStartup:true</code> in <code>initOptions.js</code>, making an adapter call.</li> <li>• Ensure that WL.Analytics is ready before calling "WL.Analytics.log" on page 492</li> <li>• If you are using Tealeaf directly and you modified the Tealeaf configuration, revert the configuration to the one generated by IBM Worklight.</li> </ul>
No analytics content appears in the <b>Analytics</b> tab on the IBM Worklight Console.	Ensure that the <code>wl.analytics.url</code> property is set in your JNDI configuration for this war. The correct configuration is confirmed by the presence of the Analytics tab on the IBM Worklight Console.



Table 223. Analytics troubleshooting guidelines (continued)

Problem	Actions to take
The Analytics tab is not present in the IBM Worklight Console.	Ensure that the <code>wl.analytics.url</code> property is set in your JNDI configuration for this war. The correct configuration is confirmed by the presence of the Analytics tab on the IBM Worklight Console.
The server logs tab in the analytics page shows too many or too few log records This tab shows Worklight Server logs in near real time.	Logs are sent based on the level of logging configured on Worklight Server. See “The WL.Logger object” on page 573
The server logs tab shows no server logs.	Server logs are not sent if the <code>wl.analytics.logs.forward</code> property is set to false in your JNDI configuration for this war. The default value for this property is true.
Analytics content is slow to appear in the <b>Analytics</b> tab in the IBM Worklight Console.	The analytics feature in client applications and in the Worklight Server buffer collect data and only send after a threshold is reached. See the “Worklight Server throughput tuning” on page 854 and “WL.Analytics” on page 491.
My application crashed, but no crash data appears at the server.	Crash data is recorded persistently on the device and sent during the next application start, after successfully connecting to Worklight Server. Ensure that the application has been restarted after the crash. Note the data may be buffered at Worklight Server, in which case see “Worklight Server throughput tuning” on page 854.

For more information, see “Troubleshooting Worklight Server” on page 177

## Reports

IBM Worklight provides an extensible mechanism for enterprises to use to integrate reporting tools with IBM Worklight.

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. To understand the similarities and differences between the existing reports feature and the new operational analytics feature, see “Comparison of operational analytics and reports features” on page 839

**Note:** .

IBM Worklight provides two reporting mechanisms:

### 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 861.



### Device usage reports.

IBM Worklight provides reports about 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 865.

**Note:** Device usage reports are functional only in IBM Worklight Customer Edition and IBM Worklight Enterprise Edition.

### 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 867.

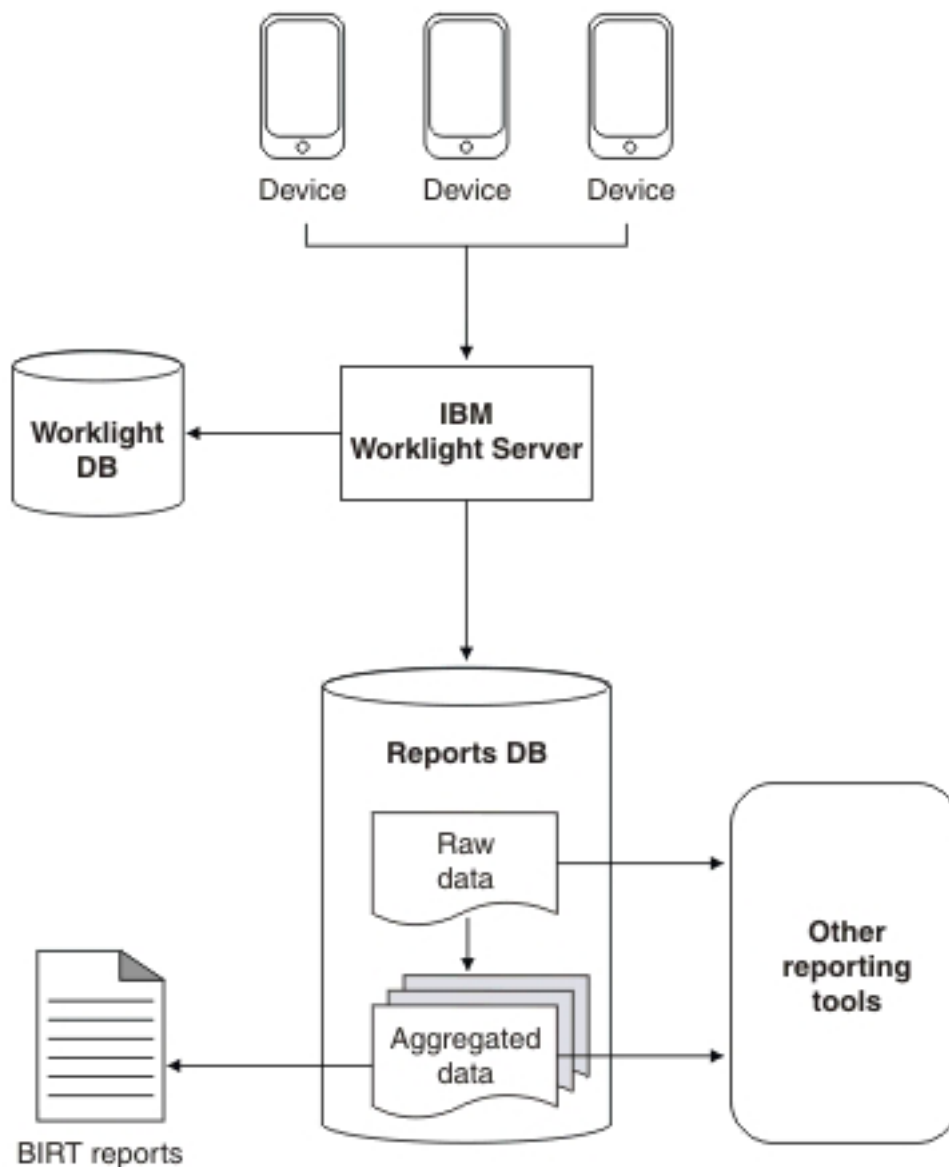


Figure 140. High-level overview of the reports architecture

The reports architecture diagram shows how the raw data feed comes from three devices into the Worklight Server and then into the IBM Worklight database, the Reports database, or both. From the Reports database, data then becomes aggregated data and is filtered out into the BIRT reports or to other reporting tools.

**Important:** When you work 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

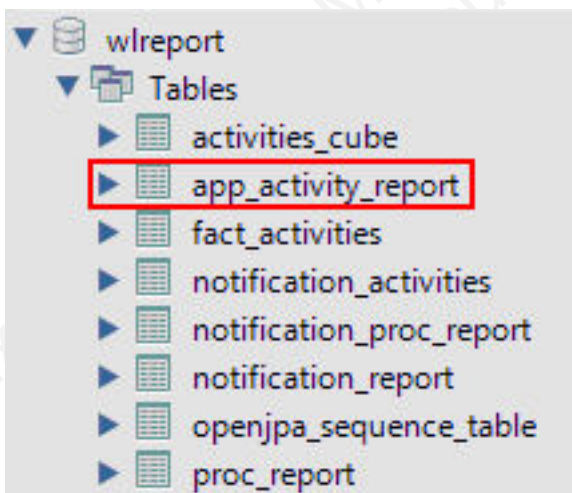
Raw data reports provide you with analytics information about your applications and adapter usage, such as activity type, device information, and application version. Use the following steps to enable the raw data reports feature:

### Procedure

1. Ensure that the IBM Worklight Server application server is not running.
2. Create a separate database or a new schema for reports. This action is not mandatory but is useful because the raw data table is rapidly populated. For information about creating databases in a development environment, see “IBM Worklight database setup for development mode” on page 716. For information about creating databases and schemas in a production environment, see “Creating and configuring the databases” on page 666.
3. When you work in a development environment, complete the following steps.
  - a. Edit the `worklight.properties` file. Uncomment the `reports.exportRawData` property and set its value to `true`.
  - b. 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.url=jdbc:mysql://localhost:3306/wlreport
wl.reports.db.username=worklight
wl.reports.db.password=worklight
```

- c. Ensure that the `wl.reports.db.url` property contains the URL of the database you are planning to use for raw data.
4. When you work in a production environment, connect to the reports database by using JNDI environment entries in addition to editing the `worklight.properties` file, as described in the previous step. See “Configuring an IBM Worklight project in production by using JNDI environment entries” on page 723.
5. Restart your application server.  
The `app_activity_report` table of the raw data database is populated with data as you use your applications and adapters.



The raw data `app_activity_report` table contains the following information:

Column	Description
ACTIVITY_TIMESTAMP	UTC time of entry
GADGET_NAME	IBM Worklight Application name

Column	Description
<b>GADGET_VERSION</b>	Application version
<b>ACTIVITY</b>	Activity type
<b>ENVIRONMENT</b>	Application environment name (iPhone, Android, etc.)
<b>SOURCE</b>	User identifier
<b>ADAPTER</b>	IBM Worklight adapter name
<b>PROC</b>	IBM Worklight adapter procedure name
<b>USERAGENT</b>	User agent from HTTP header of client device
<b>SESSION_ID</b>	A unique identifier for the user's session on the server
<b>IP_ADDRESS</b>	IP address of the client
<b>DEVICE_ID</b>	A unique device ID
<b>DEVICE_MODEL</b>	Manufacturer model, for example Galaxy I9000
<b>DEVICE_OS</b>	Device operating system version
<b>LONGITUDE</b>	The longitude of the device. Requires that ongoing acquisition is enabled for Geo.
<b>LATITUDE</b>	The latitude of the device. Requires that ongoing acquisition is enabled for Geo.
<b>POS_USER_TIME</b>	The local time on the device when the latest position information (longitude and latitude) were updated. Requires that ongoing acquisition is enabled for Geo.
<b>WIFI_APS</b>	The access points visible on the device. Requires that ongoing acquisition is enabled for WiFi.
<b>WIFI_CONNECTED_SSID</b>	The SSID (network identification) of the connected WiFi access point. Requires that ongoing acquisition is enabled for WiFi.
<b>WIFI_CONNECTED_MAC</b>	The MAC address of the connected WiFi access point. Requires that ongoing acquisition is enabled for WiFi.
<b>WIFI_USER_TIME</b>	The local time on the device when the latest WiFi information was updated. Requires that ongoing acquisition is enabled for WiFi.
<b>APP_CONTEXT</b>	The application context, as set by WL.Server.setApplicationContext.

The following activities can be included in reports:

Activity	Description
Init	Application initialization
Login	Successful authentication in using the application
Adoption New	Not supported in IBM Worklight V5.0
Adoption	Not supported in IBM Worklight V5.0
Query	Procedure call to an adapter

Activity	Description
Logout	User logout
Event	An event handler was called

In addition to predefined activity types, custom activities can be logged by using `WL.Client.logActivity("custom-string")` APIs.

When the activity is Event, the reporting information comes from the event device context instead of `WL.Server.getClientDeviceContext`. Also, when the activity is Event the **PROC** column gives the name of the event handler function that was called.

**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. For example, the following commands delete Oracle database rows that are more than 30 days old from the `activities_cube` and `app_activity_report` tables. For other databases such as MySQL, you need to modify the syntax appropriately.

**To delete rows from `activities_cube` that are more than 30 days old (assuming `ACTIVITY_DATE` is a `DATE` type field):**

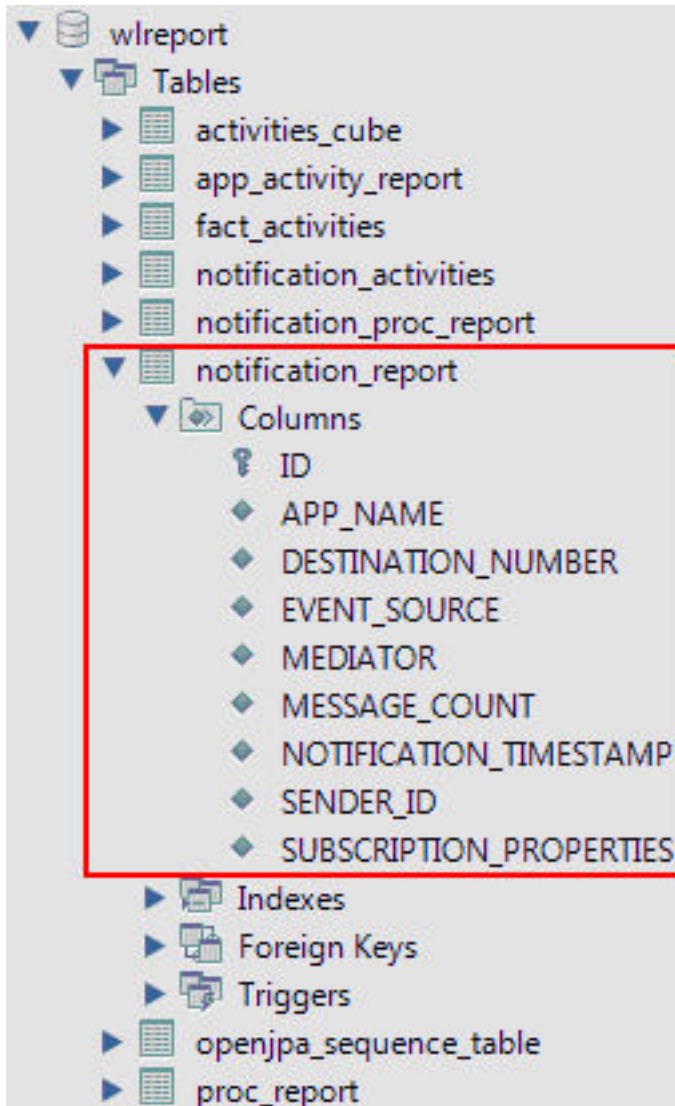
```
DELETE FROM ACTIVITIES_CUBE WHERE ACTIVITY_DATE <= TRUNC(SYSDATE) - 30
```

**To delete rows from `app_activity_report` that are more than 30 days old (assuming `ACTIVITY_TIMESTAMP` is a `TIMESTAMP` type field):**

```
DELETE FROM APP_ACTIVITY_REPORT WHERE ACTIVITY_TIMESTAMP <= TO_TIMESTAMP(TRUNC(SYSDATE) -
```

Purging data by deleting rows might fail on heavily-loaded systems. An alternative approach is to use database table partitions to facilitate the purging of accumulated data. For more information, see "Optimization and tuning of Worklight Server project databases" on page 86.

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.



## Device usage reports

For simpler and faster access to the reports data, Worklight Server runs an analytics data processor task at a default time interval of every 24 hours.

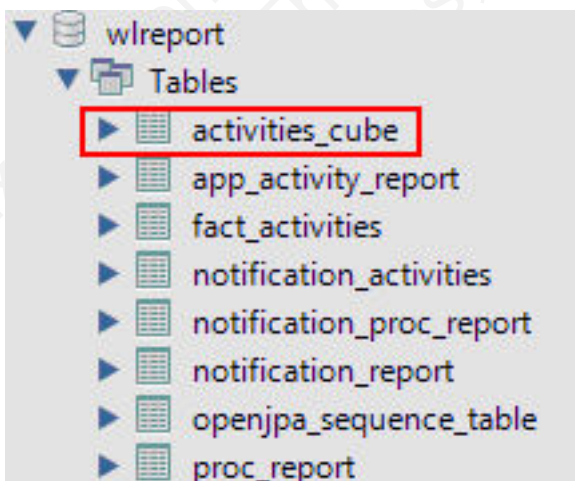
The analytics data processor task retrieves raw entries for the specified time interval from the `app_activity_report` table and processes them to populate the `fact_activities` table.

**Note:** The `fact_activities` table is only populated with usage data from hybrid and native applications from actual devices. Usage data from Worklight mobile web applications that are running on actual devices or from a browser, such as when you are using preview, is not populated into this table.



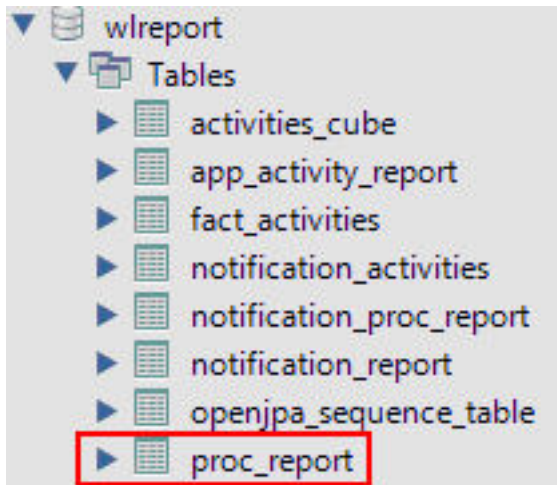


The fact\_activities table contains a total activity count (number of logged actions) per application, application version, device, and environment. The fact\_activities data is also processed and put into the activities\_cube table. This table has the same structure as the fact\_activities table and only contains records for the last 30 days.

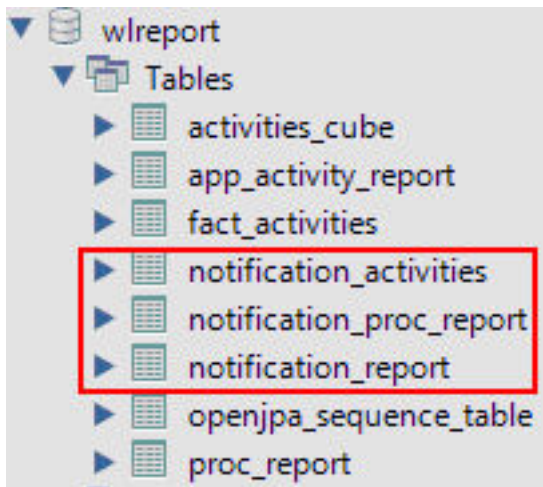




Each time the data processing is done, a time stamp is added to a `proc_report` table with the processing result (time stamp and number of processed entries).



In addition, `notification_report` table data is also processed to populate the `notification_activities` table with consolidated data. The table 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.



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 “Device usage reports” on page 865 and “Using raw data reports” on page 861. 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 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 224. 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

Table 224. 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	a22b0614-0d41-49c0-9ec5-370c804b98f8	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-1f0bd0a20d4	Apr 15, 2012 12:00 AM	WL-App-6
9	c7c502dc-fad1-411c-9f8c-50654b419df2	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-7745-41f5-9c67-9e9c003004f	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	2f218876-5f81-4ee5-90d3-6e28017d0f66	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-9e31-37d2b76c78ff	May 2, 2012 12:00 AM	WL-App-5
20	6dfffadab-48f5-4a24-9954-e78c441f917b	May 4, 2012 12:00 AM	WL-App-6
21	cdd92489-0fca-4449-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 141. 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 876
- 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. Extract 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 an option, and is 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 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 about exact settings.

```
set CATALINA_OPTS=-Xms512m -Xmx512m -XX
```

8. Restart your Tomcat.
9. Go to the Tomcat 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 session

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](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

Complete these steps to install Business Intelligence Reporting Tools 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 are using and copy it to the libs folder.
4. Download the latest release of BIRT run time 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

Set up 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.

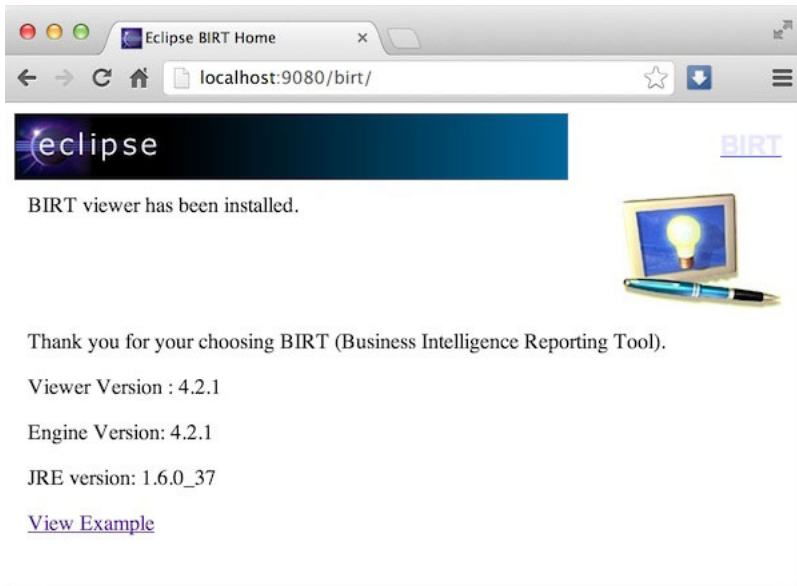
```
<server description="new server">
  <featureManager>
    <feature>jsp-2.2</feature>
  </featureManager>

  <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>
```



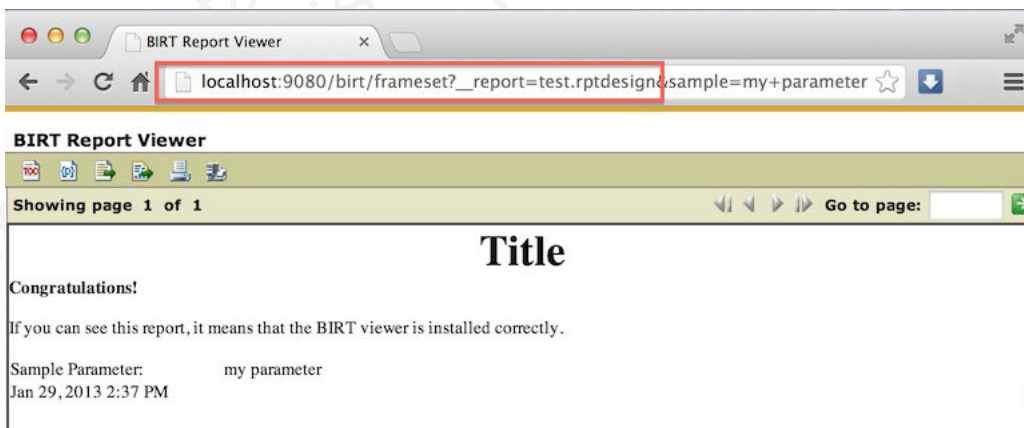
13. Start your Liberty instance.
14. Browse to `http://server:port/birt`. The BIRT Viewer landing page opens.



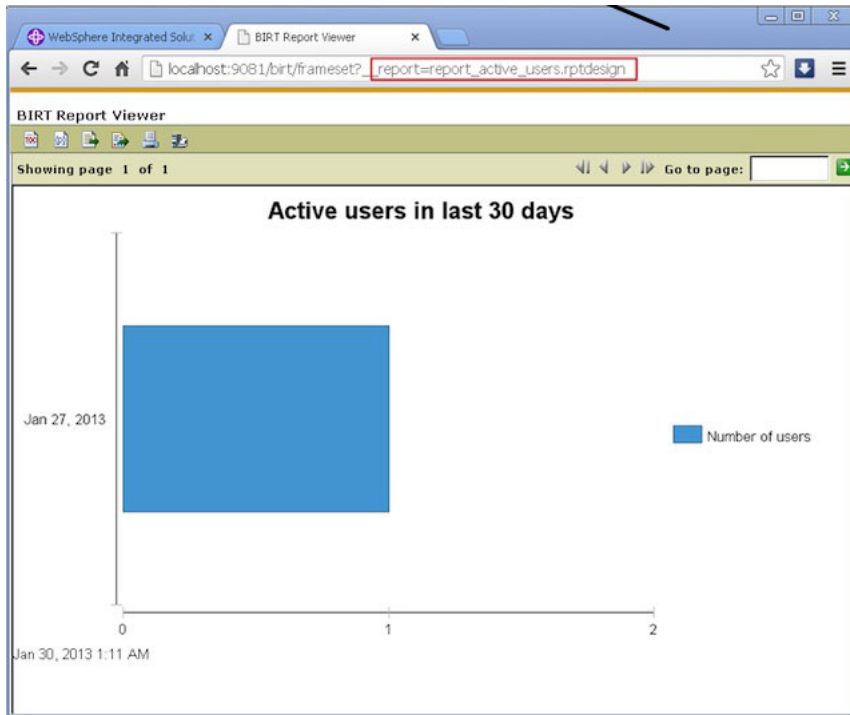
15. Click **View Example** link.
16. If you see the following error message, refresh your page.



17. The BIRT Viewer sample report appears.



Note `test.rptdesign` in the page URL. You can replace this text with the name of other **rptdesign** files, as shown here for example:



## Installing BIRT on WebSphere Application Server Full Profile

Complete these steps to install Business Intelligence Reporting Tools 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`



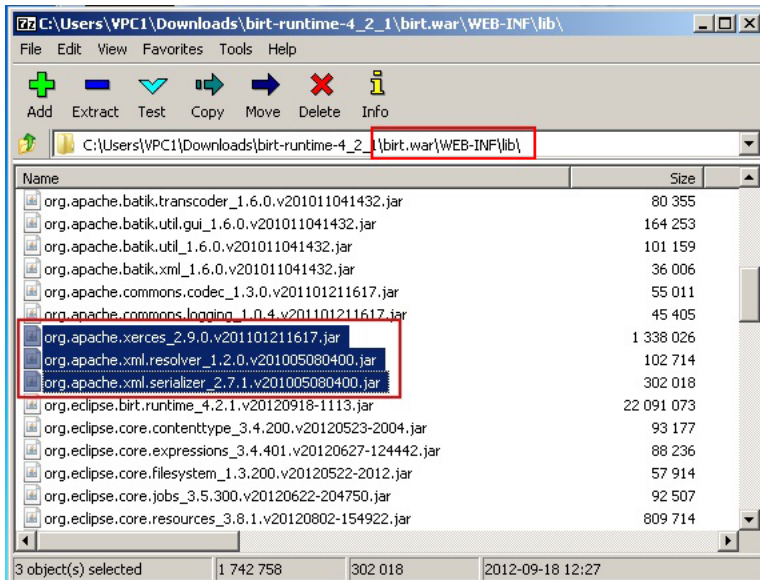


Figure 142. Deleting three files

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 the **Classes loaded with application class loader first** option.
13. Click **OK**.
14. Save the Master Configuration.

## Configuring BIRT reports for your application server by using Ant

You can update your BIRT reports with your web application server settings by using Ant.

### About this task

To use BIRT reports, you must update them with your web application server settings and install them in your server web applications folder. The easiest way to do this is to specify a `<reports>` element in the Ant script that invokes the `<configureapplicationserver>` Ant task.

## Procedure

1. Ensure that the `<configureapplicationserver>` invocation has the inner element `<reports todir="web applications directory"/>`. See “Ant `configureapplicationserver` task reference” on page 698 for more details.
2. Invoke the Ant script, which copies the report templates from the `WorklightServer/report-templates/` directory to the web applications directory, adjusting the `<data-sources>` element as needed.
3. Verify that the BIRT Viewer application is installed and running on your application server.
4. To view or edit a BIRT Report, go to the path `http://your-server/birt/frameset?__report=[report name].rptdesign.`, in which `[report name].rptdesign` 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`

## Manually 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 Worklight Server installation folder created by the IBM Installation Manager.
2. Locate the `\report-templates\` folder, which contains a set of `.rptdesign` files.
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].rptdesign 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

When BIRT is installed in Eclipse, it 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.

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 Eclipse installation instructions in the installation package. Eclipse and the BIRT components, including the Report Designer, are installed along with Eclipse.

## 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 IBM Worklight, follow these steps:

### Procedure

1. Click **Help > Install new software**
2. In the **Work with...** dropdown, select <http://download.eclipse.com/release/juno>
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 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
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

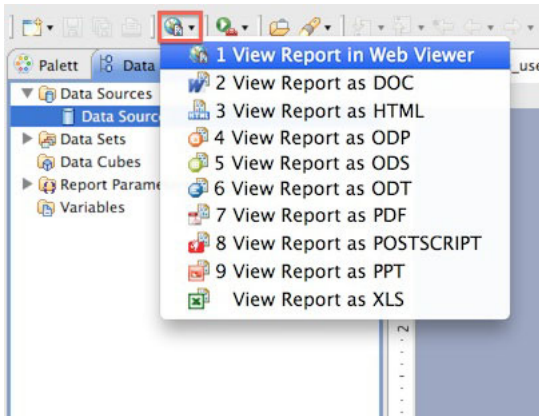
With BIRT installed in Eclipse, you can view reports through the Eclipse interface.

### About this task

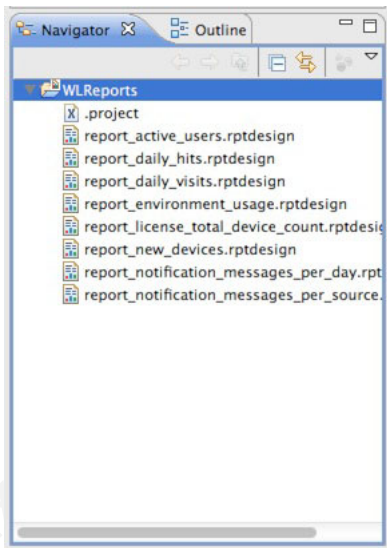
To view BIRT reports in Eclipse, follow these steps:

### Procedure

1. Click the black arrow next to **View Report**.



2. Select the output format for your report
3. View the report.



## Notification reports database schema

IBM Worklight uses a database schema to store the notification reports data derived from the raw data.


<b>NOTIFICATION_ACTIVITIES</b>	
SID varchar(255)	
NOTIFICATION_DATE date	
EVENT_SOURCE varchar(255)	
MEDIATOR varchar(255)	
TOTAL_NOTIFICATIONS int(11)	

Figure 143. 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.

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



---

## Chapter 11. Integrating 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).

---

### Introduction to IBM Worklight integration options

Developers and administrators can use integration options that are available as part of the larger IBM Worklight offering, as introduced here.

IBM Worklight provides extensible connectivity options to external resources by using the adapter technology available in IBM Worklight. IBM Worklight also provides a flexible authentication framework to support existing security requirements through the authenticator or login modules.

Figure 144 on page 882 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

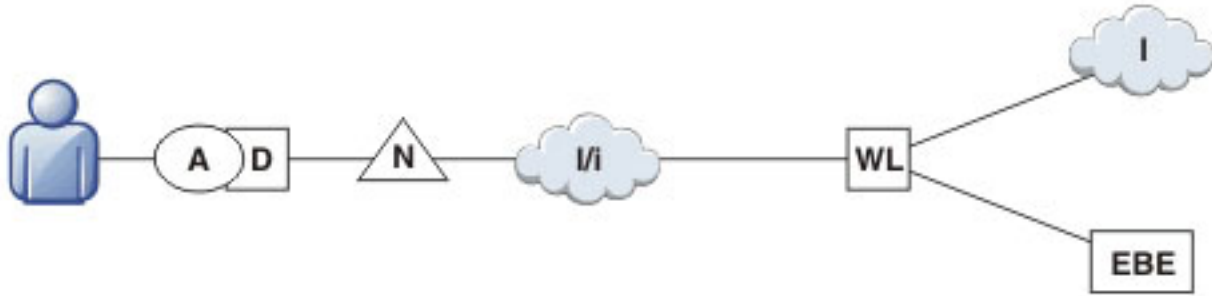


Figure 144. Overall Topology

Figure 145 shows where these products fit within the typical IBM Worklight topology diagram shown in Figure 144.

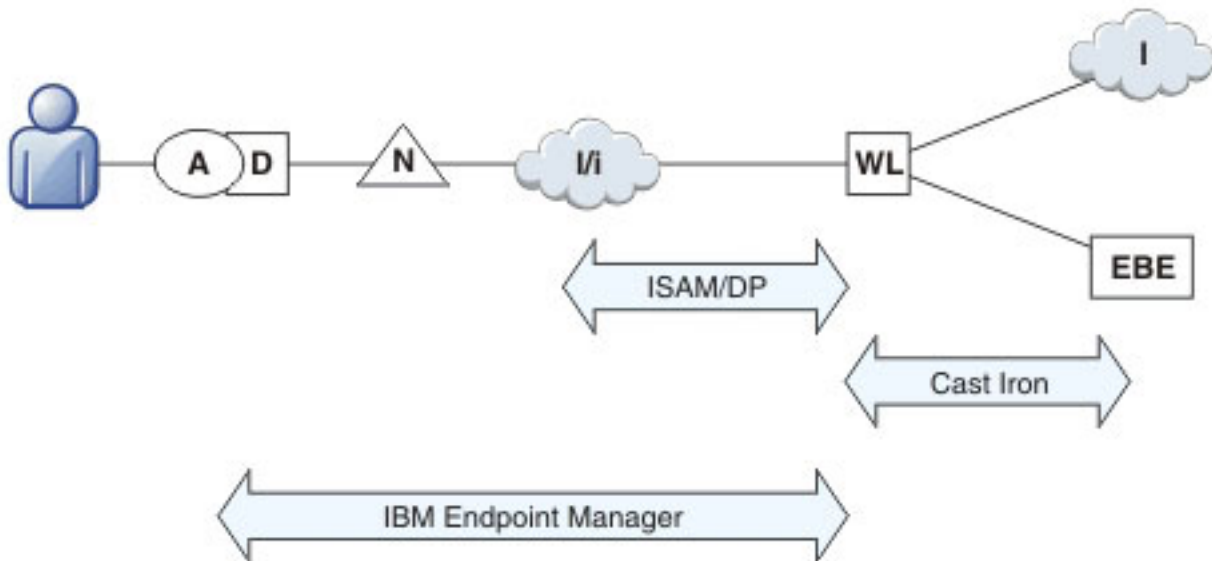


Figure 145. Integration Points

## 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 IBM Worklight:

- 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 146 shows how the topology in Figure 1 in *Introduction to IBM Worklight integration options* 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.

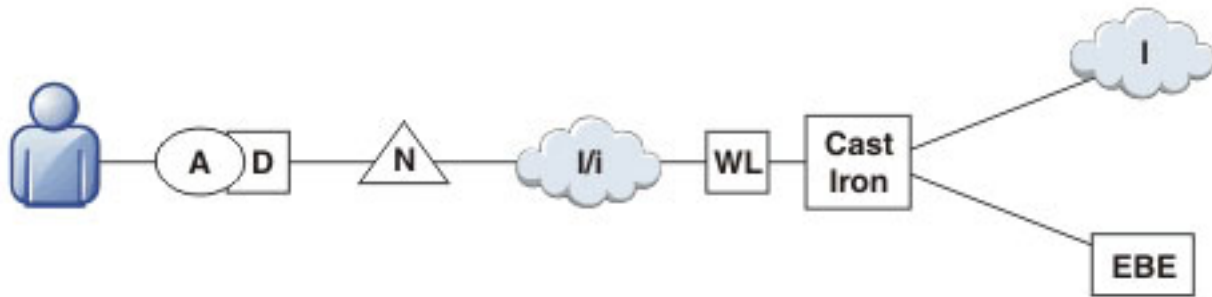


Figure 146. Integration with Cast Iron

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 Chapter 3, "Tutorials and samples," on page 25.

## 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 147, and follow the gateway pattern.

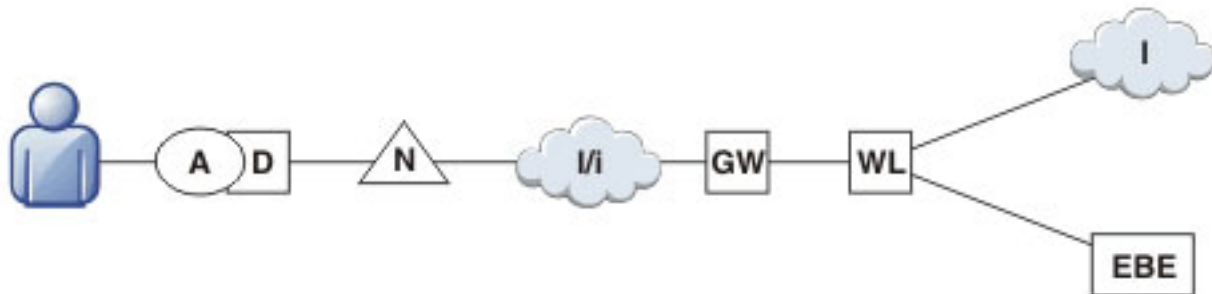


Figure 147. Integration with reverse proxy

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 148.

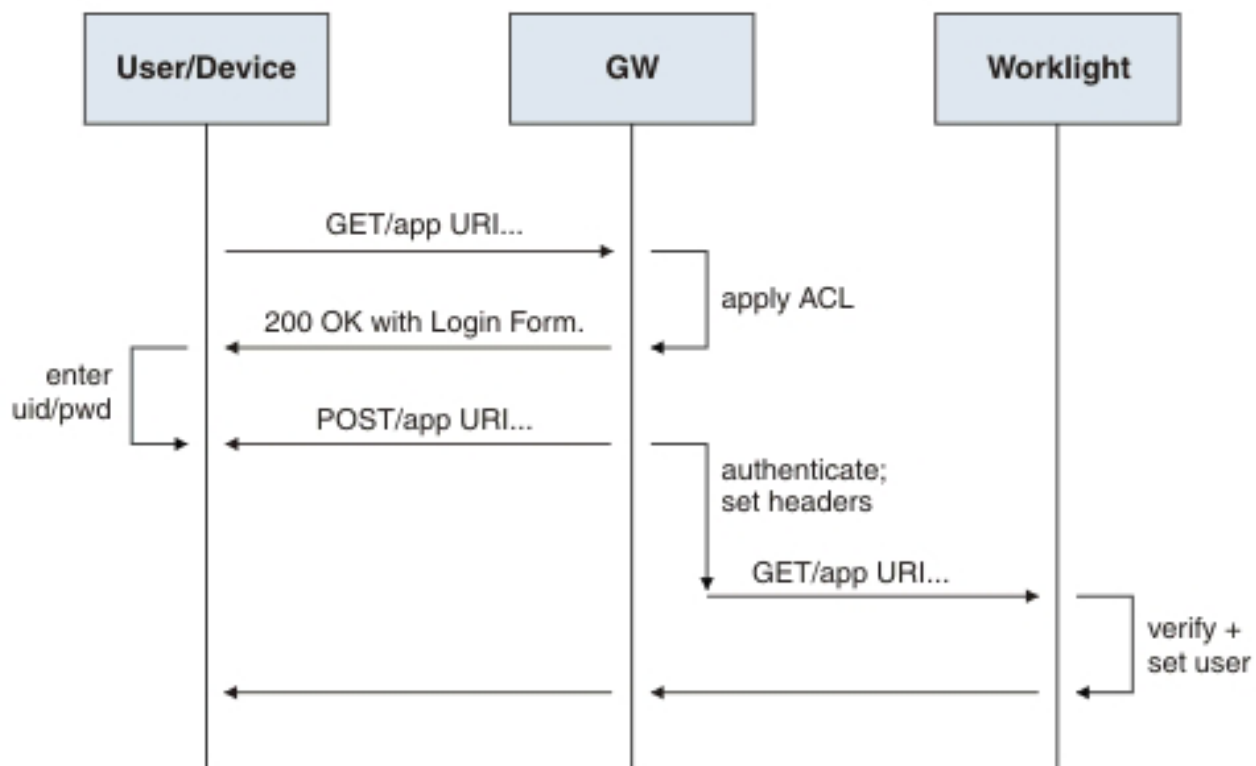


Figure 148. Authentication flow

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.

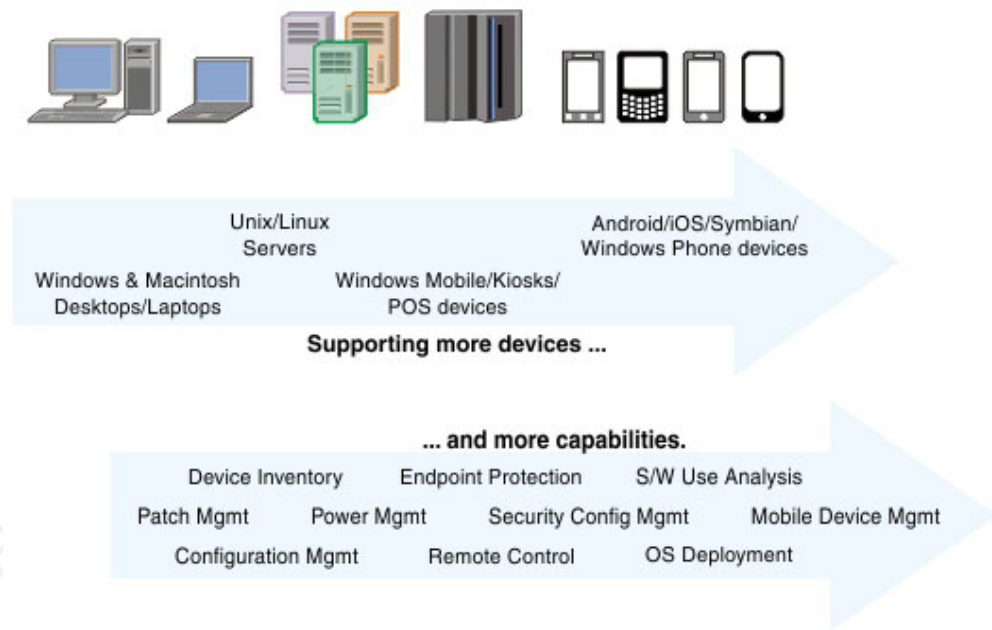
- On successful authentication, the gateway forwards an LTPA token (in the form of an HTTP cookie) to IBM Worklight
- IBM Worklight on WebSphere Application Server is configured to use `WebSphereFormBasedAuthenticator` and `WebSphereLoginModule`.

---

## IBM Endpoint Manager for Mobile Devices overview

An overview of the features and architecture of IBM Endpoint Manager for Mobile Devices.

IBM Worklight allows the integration of security features that are provided by IBM Endpoint Manager. The purpose of IBM Endpoint Manager is to deliver a unified systems and security management solution for all enterprise devices.



The most interesting capabilities that are provided by IBM Endpoint Manager for Mobile Devices from a security standpoint are delivered in the following areas:

- Enterprise Access Management – Configuration of email, VPN, and WiFi.
- Policy and Security Management – Password policies, device encryption, jailbreak, and root detection.
- Management Actions – Selective wipe, full wipe, deny email access, remote lock, user notification, clear passcode.
- Application Management – Application inventory, enterprise app store, whitelisting, blacklisting, Apple Volume Purchase Program (VPP).
- Container Solution – Enterproud Divide provides a secure container for BYOD (Bring Your Own Device) devices that is secure and manageable. Divide is an app that provides a workspace that mimics device capabilities while being isolated from the rest of the device. This allows information within Divide to be secured and managed separately from the rest of the device.

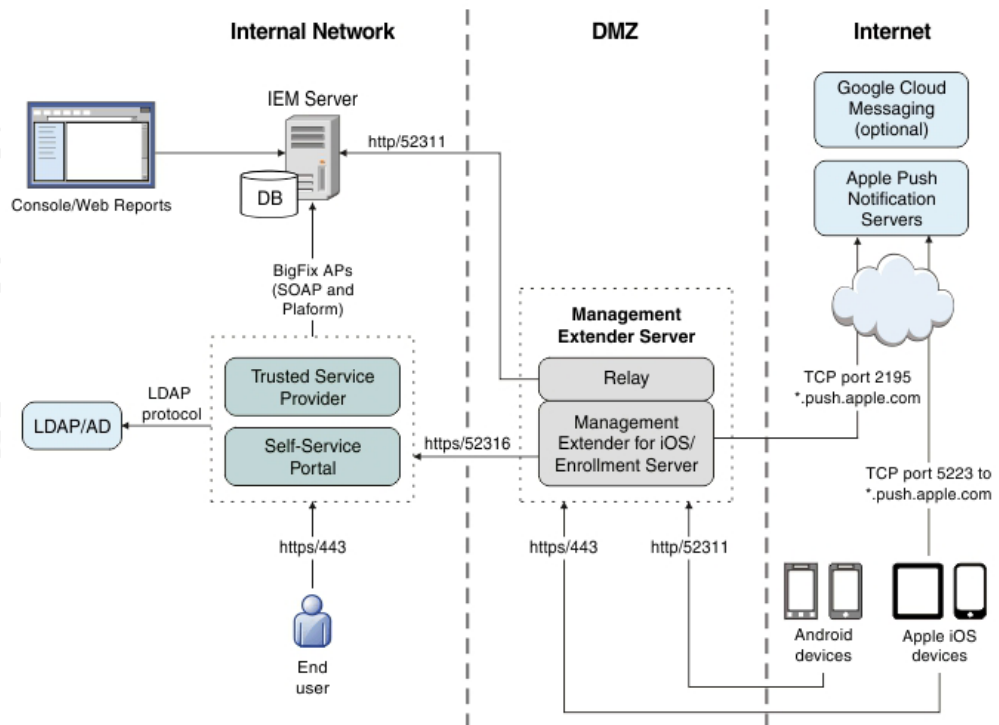
- PIM (Personal Information Manager) – NitroDesk TouchDown is a PIM product that allows enterprise data such as email to be secured separately in a BYOD Android environment.
- Support for SAFE (Samsung's proprietary APIs that allow more security than standard Android).

## IBM Endpoint Manager for Mobile Devices architecture

IBM Endpoint Manager for Mobile Devices uses two approaches to manage those devices:

- An agent-based, Mobile Device Management (MDM) API-based approach that is supported on Android and iOS Devices through the IBM Mobile Client. This approach provides the full set of capabilities through either a native Agent on the Android platform or the usage of Apple's MDM APIs and the Push Notification Server infrastructure.
- An email-based management through Exchange (Active Sync) and Lotus® Traveler (IBM Sync). In this approach, Android, iOS, Windows Phone, and Symbian are supported, but the functionality is limited and includes the ability to wipe a device, deny email access and set password policies. You cannot see individual device details, perform application management, configure WiFi or VPN connections or provide advance restrictions as in the agent-based, MDM API-based approach.
- Container management is available through Enterproid Divide, as discussed in the previous section.
- PIM is available through NitroDesk TouchDown.

The following diagram shows an architectural overview of a production-level, agent-based, MDM API-based implementation with IBM Endpoint Manager for Mobile Devices.



## Managing end points with IBM Endpoint Manager

An overview of the use of IBM Endpoint Manager for Mobile Devices 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 for Mobile Devices, as shown in Figure 149.

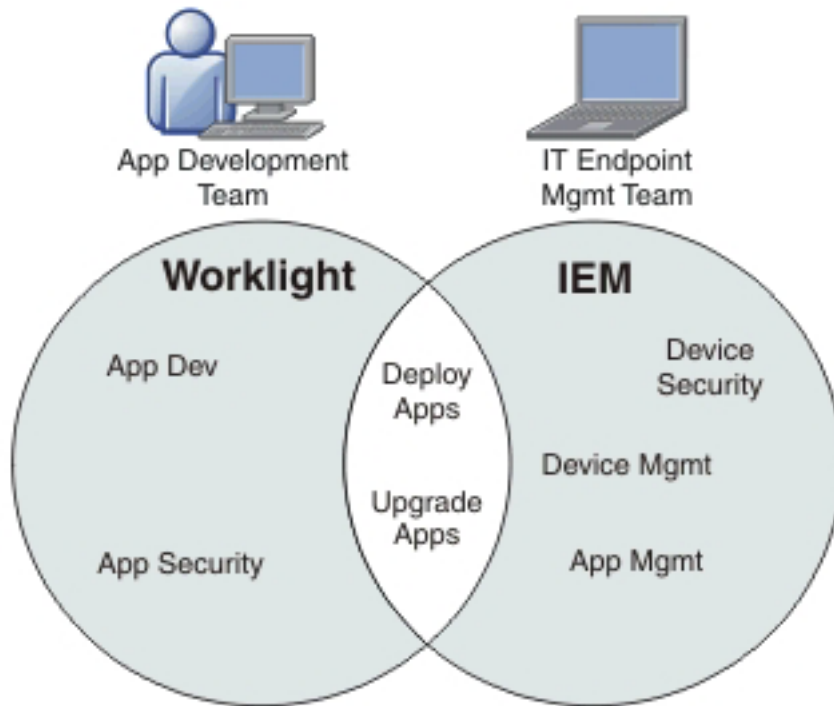


Figure 149. IBM Worklight and IBM Endpoint Manager management capabilities

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.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>

---

## Chapter 12. 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.

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.
  - 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®.

## 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.

---

## Chapter 13. 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 IBM Mobile Test Workbench for Worklight” on page 41
- “Troubleshooting Worklight Server” on page 177
- “Troubleshooting an installation blocked by DB2 connection errors” on page 178
- “Troubleshooting failure to create the DB2 database” on page 177
- “Troubleshooting IBM HTTP Server startup” on page 172
- “Troubleshooting to find the cause of installation failure” on page 177
- “Troubleshooting a Cast Iron adapter – connectivity issues” on page 363
- “Troubleshooting JSONStore and data synchronization” on page 410
- “Troubleshooting analytics” on page 858

**Important:** For more information about known limitations or issues in the product, see “Known limitations” on page 9 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.



---

## Chapter 14. Glossary

This glossary includes terms and definitions for IBM Worklight.

The following cross-references are used in this glossary:

- "See" refers you from a term to a preferred synonym, or from an acronym or abbreviation to the defined full form.
- "See also" refers you to a related or contrasting term.

To view glossaries for other IBM products, go to [www.ibm.com/software/globalization/terminology](http://www.ibm.com/software/globalization/terminology) (opens in new window).

"A" "B" on page 896 "C" on page 896 "D" on page 897 "E" on page 897 "F" on page 897 "G" on page 898 "H" on page 898 "I" on page 898 "K" on page 898 "L" on page 898 "M" on page 899 "N" on page 899 "P" on page 899 "R" on page 900 "S" on page 900 "T" on page 901 "V" on page 901 "W" on page 901

---

### A

#### **acquisition policy**

A policy that controls how data is collected from a sensor of a mobile device. The policy is defined by application code.

#### **adapter**

The server-side code of an IBM Worklight application. Adapters connect to enterprise applications, deliver data to and from mobile applications, and perform any necessary server-side logic on sent data.

**alias** An assumed or actual association between two data entities, or between a data entity and a pointer.

#### **Android**

A mobile operating system created by Google, most of which is released under the Apache 2.0 and GPLv2 open source licenses. See also mobile device.

**API** See application programming interface.

**app** A mobile device application.

#### **Application Center**

An IBM Worklight component that can be used to share applications and facilitate collaboration between team members in a single repository of mobile applications.

#### **Application Center installer**

An application that lists the catalog of available applications in the Application Center. The Application Center installer must be present on a device in order to install applications from your private application repository.

#### **application descriptor file**

A metadata file that defines various aspects of an application.

**application programming interface (API)**

An interface that allows an application program that is written in a high-level language to use specific data or functions of the operating system or another program.

**authentication**

A security service that provides proof that a user of a computer system is genuinely who that person claims to be. Common mechanisms for implementing this service are passwords and digital signatures.

**authenticator**

1. A server-side component that issues a sequence of challenges on the server side and responds on the client side. See also challenge handler.
2. In the Kerberos protocol, a string of data that is generated by the client and sent with a ticket that is used by the server to certify the identity of the client.

---

**B**

**binary** Pertaining to something that is compiled, or is executable.

**BlackBerry OS**

A closed source, proprietary mobile operating system created by Research in Motion. See also mobile device.

**block** A collection of several properties (such as adapter, procedure, or parameter).

**build definition**

An object that defines a build, such as a weekly project-wide integration build.

---

**C****callback function**

Executable code that allows a lower-level software layer to call a function defined in a higher-level layer.

**catalog**

A collection of apps.

**certificate**

In computer security, a digital document that binds a public key to the identity of the certificate owner, thereby enabling the certificate owner to be authenticated. A certificate is issued by a certificate authority and is digitally signed by that authority.

**challenge**

A request for certain information to a system. The information, which is sent back to the server in response to this request, is necessary for client authentication.

**challenge handler**

A client-side component that issues a sequence of challenges on the server side and responds on the client side. See also authenticator.

**client** A software program or computer that requests services from a server.

**client-side authentication component**

A component that collects client information, then uses login modules to verify this information.

**clone** An identical copy of the latest approved version of a component, with a new unique component ID.

**cluster**  
A group of servers that share a database instance.

**component**  
A reusable object or program that performs a specific function and works with other components and applications.

**credential**  
A set of information that grants a user or process certain access rights.

---

## D

**data source**  
The means by which an application accesses data from a database.

**deployment**  
The process of installing and configuring a software application and all its components.

**device** See mobile device.

**device context**  
Data that is used to identify the location of a device. This data can include geographical coordinates, WiFi access points, and timestamp details. See also trigger.

**documentify**  
A JSONStore command used to create a document.

---

## E

**emulator**  
An application that can be used to run an application meant for a platform other than the current platform. For example, BlackBerry OS includes an emulator to run Android applications.

**encryption**  
In computer security, the process of transforming data into an unintelligible form in such a way that the original data either cannot be obtained or can be obtained only by using a decryption process.

**entity** A user, group, or resource that is defined to a security service,

**environment**  
A specific instance of a configuration of hardware and software.

**event** An occurrence of significance to a task or system. Events can include completion or failure of an operation, a user action, or the change in state of a process.

**event source**  
An object that supports an asynchronous notification server within a single Java virtual machine. Using an event source, the event listener object can be registered and used to implement any interface.

---

## F

**facet** An XML entity that restricts XML data types.

**fire** In object-oriented programming, to cause a state transition.

**fragment**

A file that contains HTML tags that can be appended to a parent element.

---

**G**

**gateway**

A device or program used to connect networks or systems with different network architectures.

**geofence**

A circle or a polygon that defines a geographical area.

**geolocating**

The process of pinpointing a location based on the assessment of various types of signals. In mobile computing, often WLAN access points and cell towers are used to approximate a location. See also location services.

---

**H**

**hybrid application**

An application that is primarily written in Web-oriented languages (HTML5, CSS, and JS), but is wrapped in a native shell so that the app behaves like, and provides the user with all the capabilities of, a native app.

---

**I**

**inner application**

An application that contains the HTML, CSS, and JavaScript parts that run within a shell component. Inner applications must be packaged within a shell component to create a full hybrid application.

---

**K**

**key**

1. A cryptographic mathematical value that is used to digitally sign, verify, encrypt, or decrypt a message.
  2. One or more characters within an item of data that are used to uniquely identify a record and establish its order with respect to other records.
- 

**L**

**library**

1. A collection of model elements, including business items, processes, tasks, resources, and organizations.
2. A system object that serves as a directory to other objects. A library groups related objects, and allows users to find objects by name.

**load balancing**

A computer networking method for distributing workloads across multiple computers or a computer cluster, network links, central processing units,

disk drives, or other resources. Successful load balancing optimizes resource use, maximizes throughput, minimizes response time, and avoids overload.

**local store**

An area on a device where applications can locally store and retrieve data without the need for a network connection.

**location services**

A feature in IBM Worklight that can be used to create differentiated services that are based on a user location. Location services involve collecting geolocation and WiFi data and transmitting this data to a server, where it can be used for executing business logic and analytics. Changes in the location data result in triggers being activated, which cause application logic to execute. See also geolocating.

---

**M**

**mobile**

See mobile device.

**mobile client**

See Application Center installer.

**mobile device (mobile)**

A telephone, tablet, or personal digital assistant that operates on a radio network. See also Android, BlackBerry OS.

---

**N**

**native app**

An app that is compiled into binary code for use on the mobile operating system on the device.

**node** A logical group of managed servers.

**notification**

An occurrence within a process that can trigger an action. Notifications can be used to model conditions of interest to be transmitted from a sender to a (typically unknown) set of interested parties (the receivers).

---

**P**

**page navigation**

A browser feature that enables users to navigate backwards and forwards in a browser.

**poll** To repeatedly request data from a server.

**project**

The development environment for various components, such as applications, adapters, configuration files, custom Java code, and libraries.

**project WAR file**

A web archive (WAR) file that is deployed on an application server. This file contains the default server-specific configurations such as security profiles, server properties, and more.

**provision**

To provide, deploy, and track a service, component, application, or resource.

**proxy** An application gateway from one network to another for a specific network application such as Telnet or FTP, for example, where a firewall proxy Telnet server performs authentication of the user and then lets the traffic flow through the proxy as if it were not there. Function is performed in the firewall and not in the client workstation, causing more load in the firewall.

**push** To send information from a server to a client. When a server pushes content, it is the server that initiates the transaction, not a request from the client.

**push notification**

An alert indicating a change or update that appears on a mobile app icon.

---

**R**

**realm** A collection of resource managers that honor a common set of user credentials and authorizations.

**reverse proxy**

An IP-forwarding topology where the proxy is on behalf of the back-end HTTP server. It is an application proxy for servers using HTTP.

**root** The directory that contains all other directories in a system.

---

**S****server-side authentication component**

See authenticator.

**service**

A program that performs a primary function within a server or related software.

**session**

A logical or virtual connection between two stations, software programs, or devices on a network that allows the two elements to communicate and exchange data for the duration of the session.

**shell**

A component that provides custom native capabilities and security features for applications.

**sign**

To attach a unique electronic signature, derived from the sender's user ID, to a document or field when a document is mailed. Signing mail ensures that if an unauthorized user creates a new copy of a user's ID, the unauthorized user cannot forge signatures with it. In addition, the signature verifies that no one has tampered with the data while the message was in transit.

**simulator**

An environment for staging code that is written for a different platform. Simulators are used to develop and test code in the same IDE, but then deploy that code to its specific platform. For example, one can develop code for a BlackBerry device on a computer, then test it using a simulator on that computer.

**skin** An element of a graphical user interface that can be changed to alter the appearance of the interface without affecting its functionality.

**slide** To move a slider interface item horizontally on a touchscreen. Typically, apps use slide gestures to lock and unlock phones, or toggle options.

**subelement**

In UN/EDIFACT EDI standards, an EDI data element that is part of an EDI composite data element. For example, an EDI data element and its qualifier are subelements of an EDI composite data element.

**subscription**

A record that contains the information that a subscriber passes to a local broker or server to describe the publications that it wants to receive.

**syntax** The rules for the construction of a command or statement.

**system message**

An automated message on a mobile device that provides operational status or alerts, for example if connections are successful or not.

---

**T**

**tap** To briefly touch a touchscreen. Typically, apps use tap gestures to select items (similar to a left mouse button click).

**template**

A group of elements that share common properties. These properties can be defined only once, at the template level, and are inherited by all elements that use the template.

**trigger**

A mechanism that detects an occurrence, and can cause additional processing in response. Triggers can be activated when changes occur in the device context. See also device context.

---

**V**

**view** A pane that is outside of the editor area that can be used to look at or work with the resources in the workbench.

---

**W**

**web application**

An application that is accessible by a web browser and that provides some function beyond static display of information, for instance by allowing the user to query a database. Common components of a web application include HTML pages, JSP pages, and servlets.

**web application server**

The runtime environment for dynamic web applications. A Java EE web application server implements the services of the Java EE standard.

**web resource**

Any one of the resources that are created during the development of a web application for example web projects, HTML pages, JavaServer Pages (JSP) files, servlets, custom tag libraries, and archive files.



**widget**

A portable, reusable application or piece of dynamic content that can be placed into a web page, receive input, and communicate with an application or with another widget.

**Worklight adapter**

See adapter.

**Worklight Console**

A web-based interface that is used to control and manage applications that are deployed in Worklight Server, and to collect and analyze user statistics.

**Worklight Server**

An IBM Worklight component that acts as a runtime container for mobile applications that are developed using Worklight Studio. The Worklight Server 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 Studio**

An IBM Worklight component that is an integrated development environment (IDE) that can be used to develop and test mobile applications.

**wrapper**

A section of code that contains code that could otherwise not be interpreted by the compiler. The wrapper acts as an interface between the compiler and the wrapped code.

---

## Chapter 15. 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.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation

Dept F6, Bldg 1

294 Route 100

Somers NY 10589-3216

USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

#### **COPYRIGHT LICENSE:**

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. \_enter the year or years\_. All rights reserved.

## **Privacy Policy Considerations**

IBM Software products, including software as a service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies that collect session information (generated by the application server). These cookies contain no personally identifiable information and are required for session management. Additionally, persistent cookies may be randomly generated to recognize and manage anonymous users. These cookies also contain no personally identifiable information and are required.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent. For more information about the use of various technologies, including cookies, for these purposes, see IBM’s Privacy Policy at <http://www.ibm.com/privacy> and IBM’s Online Privacy Statement at <http://www.ibm.com/privacy/details> the section entitled “Cookies, Web Beacons and Other Technologies” and the “IBM Software Products and Software-as-a-Service Privacy Statement” at <http://www.ibm.com/software/info/product-privacy>.

## **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](http://www.ibm.com)<sup>®</sup>, AIX, Cast Iron, Cognos, DataPower, DB2, developerWorks, Lotus, 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](http://www.ibm.com/legal/copytrade.shtml).

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.

---

## Chapter 16. 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 offline user documentation  
This document is provided "as is".  
In case of issues, refer to the online user documentation.



---

## Chapter 17. 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](http://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](http://www.ibm.com/legal/copytrade.shtml).

© Copyright IBM Corporation 2006, 2014.

This information center is Built on Eclipse. ([www.eclipse.org](http://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

## Special characters

- <adapter>
  - element of adapter XML file 364
- <authentication>
  - element of HTTP adapter 370
- <connectionPolicy>
  - element of adapter XML file 366
  - element of the Cast Iron adapter 374
  - element of the HTTP adapter 368
  - element of the JMS adapter 375
  - element of the SQL adapter 372
- <connectivity>
  - element of adapter XML file 366
- <jmsConnection>
  - element of the JMS adapter 376
- <loadConstraints>
  - element of adapter XML file 366
- <namingConnection>
  - element of the JMS adapter 375
- <procedure>
  - element of adapter XML file 367
- <proxy>
  - element of HTTP adapter 371
- 861

## A

- Access Control List
  - Application Center 125, 126
- access for users and groups
  - Application Center 125, 126, 127, 128
- accessibility 469
- ACL
  - Application Center 125, 126
  - ACL management for Application Center with LDAP
    - WebSphere Application Server V8 128
- acquisition policy
  - setting 474
- adapter authenticator 429
- adapter configuration files
  - exporting 736
- adapter framework 360
- adapter invocation 380
- adapter procedures
  - HTTP
    - implementing 382
    - implementing 381
- adapter validation 383
- adapter XML file 364
  - <adapter> element 364
  - <connectionPolicy> element 366
  - <connectivity> element 366
  - <loadConstraints> element 366
  - <procedure> element 367
- adapterAuthenticator 429
- adapters
  - administering in console 735
  - anatomy 360

- adapters (*continued*)
  - benefits 360
  - building
    - Ant task 730, 732
  - Cast Iron
    - See Cast Iron adapters
  - composition 360
  - configuring 377
  - creating 377
  - deleting 737
  - deploying
    - Ant task 730, 732
    - from the console 737
    - from the Studio 392
  - deploying between environments 663
  - HTTP
    - See HTTP adapters
  - JMS
    - See JMS adapters
  - modifying 737
  - overview 360
  - replacing 737
  - SQL
    - See SQL adapters
- adding
  - desktop environment 262
  - mobile environment 262
  - to an
    - IBM Worklight application 262
    - web environment 262
- adding custom splash images 249
- additional data 855
- administering
  - application 755
  - apps and adapters
    - in Worklight Console 735
- Adobe AIR applications
  - signing 329
- Adobe AIR tools
  - installing 37
- AIR
  - See Adobe AIR
- AIR applications
  - signing 329
- analytics 3, 174, 176, 839, 840, 841, 842, 843, 844, 850, 851, 852, 855, 857, 858
  - what's new 15
- Android tools
  - installing 38
- animating transitions
  - from and to Java page 321
  - from Objective-C page to web view 319
  - from web view to Objective-C page 319
- Ant tasks
  - application servers 698, 706, 708
  - building adapters 730, 732
  - building applications 730
  - building projects 665

- Ant tasks (*continued*)
  - configuring application servers 666, 694, 698, 706, 708
    - WebSphere Application Server Network Deployment 708
  - configuring databases 666, 669, 674
  - configuring multiple Worklight projects 708
  - deploying adapters 730, 732
  - deploying applications 730
  - deploying projects 730
  - reference 674, 698, 706, 708
  - sample configuration files 706, 708
- Apache 72, 686
- Apache Tomcat 869
- Apache Tomcat server
  - manual configuration 78, 693
- API 456
  - JavaScript 224
- APIs 236
- application 241
  - administering 755
- application authenticity
  - overview 441
- application cache 342, 346, 347
  - management 346
  - managing 342, 347
- Application Center 3, 224
  - access for users and groups 125, 126, 127, 128
  - LDAP and WebSphere Application Server V7 125
  - LDAP and WebSphere Application Server V8 126, 127
- Application Center Access Control List Virtual Member Manager 126
- application descriptor 249, 255
  - <security> element 255
- application features 342, 343
  - including and excluding 342, 343
- application folder 247
- application icons 249
- application main file 249
- application publishing 3
- application resources 249
- application server 224
  - configuring
    - Ant task 666, 694, 698, 706, 708
    - reference 698, 706, 708
- Application server 871
- Application Server 191, 192
- application skins
  - applying 288
  - deleting 288
  - developing 288
- application strings 456
- applications
  - anatomy 247
  - building
    - Ant task 730
  - composition 247

- applications (*continued*)
  - creating 239, 240
  - deploying
    - Ant task 730
  - developing 239
  - hybrid 239
  - native 239
  - overview 247
  - web 239
- applying skins 288
- apps
  - administering in console 735
  - deleting 736
  - deploying 736
  - deploying between environments 663
  - submitting 736
- authenticated push for Windows Phone 8 415
- authentication
  - configuring
    - Administration Console 426
    - Application Center 426
    - usage reports 426
- authentication configuration
  - attributes of login modules 431
  - authentication realms 423
  - authenticators 424
  - configuring
    - authenticators 426
    - realms 426
  - database login module 432
  - header login module 434
  - LDAP login module 434
  - login modules 424
    - attributes 431
    - database 432
    - header 434
    - LDAP 434
    - non-validating 432
    - single identity 433
    - WASLTPAModule 434
  - non-validating login module 432
  - single identity login module 433
  - WASLTPAModule login module 434
- authentication configuration file 424
- authentication realms 423
- authenticationConfig.xml 424
- authenticators 424
  - basic 427
  - configuring 426
  - form-based 428
  - header 429
  - LTPA 430
  - persistent cookie 429
- auto-complete 383
- auto-provisioning 436

## B

- back-end connections 3
- back-end services
  - invoking 389
- basic authenticator 427
- BasicAuthenticator 427, 428
- benefits of adapters 360
- BIRT 869, 871

- browser configuration
  - Linux 293
- browsers
  - Rich Page Editor 293
- buffer zones 475
- build settings 342, 351, 354, 356
  - what's new 17
- build-settings.xml 351, 354, 356
- building adapters
  - Ant task 730, 732
- building applications
  - Ant task 730
- building project
  - Ant task 665

## C

- CA certificates
  - selecting 439
- Cache Manifest 342, 346, 347
  - editing 347
  - managing 342, 346, 347
- capturing data 851
- Cascading Style Sheet files
  - concatenation of 351, 356
  - minification of 351, 354
- Cast Iron adapter
  - <connectionPolicy> element 374
  - root element 374
  - troubleshooting 360
- Cast Iron adapters 360
- changing
  - Dojo version 287
  - port number of application server 40
- changing the context root 268
- changing the target server 268
- client libraries 856
- client to server integration 236
- cloud
  - deployment with IBM PureApplication System 740
- clusters 856, 857
- collection.toString 547
- common
  - user-interface controls 278
- components 841
- concatenation 342, 351, 356
  - what's new 17
- concatenation engine 356
- confidence levels 475
- configuration 203, 666
  - databases 203, 666
  - security 88
- configuring 72, 686
  - adapters 377
  - Apache 72, 686
  - authentication
    - web widgets 329
  - authenticators 426
  - Derby 72, 686
  - device provisioning 439
  - realms 426
  - web widget authentication 329
  - Worklight Server
    - MySQL 116

- configuring LDAP for Application Center
  - WebSphere Application Center
    - V7 125
    - WebSphere Application Center
      - V8 127
- connecting
  - to Worklight Server 317
- connectOnStartup 317
- console 850
  - administering apps and adapters 735
- context root 698
  - changing 266, 268
- contextual search 844
- controlling the application cache
  - what's new 17
- Cordova 224, 456
- count 533
  - JSONStore 533
- create 241
- createIdentity 224
- creating
  - adapters 377
  - applications 239, 240
  - Dojo-enabled Worklight projects 286
  - projects 240
  - QNX 327
- cross-platform compatibility layer 236
- CSS files
  - concatenation of 351, 356
  - minification of 351, 354
- custom code 224
- custom server 855
- custom splash images 249
- customSecurityTest 421

## D

- dashboard 843
- data accumulation 853
- data flow 852
- data synchronization
  - troubleshooting 410
- data throughput 853
- database login module 432
- databases
  - configuration 678
  - configuring
    - Ant task 666, 669, 674, 694, 698, 706, 708
    - reference 674
  - creation 678
  - upgrading 203
- datasource custom property 416
- deleting
  - adapters 737
  - apps 736
- deleting skins 288
- deploying
  - adapters 663
    - from the console 737
    - from the Studio 392
  - application
    - to the cloud by using IBM PureApplication System 740
  - apps 663, 736
  - projects 203

- deploying (*continued*)
  - Worklight Server
    - to the cloud by using IBM PureApplication System 740
- deploying adapters
  - Ant task 730, 732
- deploying applications
  - Ant task 730
- deploying projects
  - Ant task 730
- deployment 666
  - project 666
- Derby 72, 686
- descriptors
  - application 255
- Desktop Browser apps 346
- destroy 534
  - JSONStore 534
- detect information 456
- develop 241
- Developer Edition
  - IBM Worklight 8
- developing
  - application skins 288
  - applications 239
  - guidelines
    - desktop and web environments 328
- development 3, 246, 278
  - Android 336
  - hybrid app 246
    - user interface 278
  - iOS 333
  - Java Platform, Micro Edition 340
  - native application 331, 333, 336, 340
  - user interface
    - hybrid app 278
- development environment 191, 192, 224
- development guidelines
  - desktop and web environments 328
- device provisioning 436
  - configuring 439
  - implementing 439
- device run time 236
- device type
  - what's new 22
- device-specific 456
- devices
  - tracking location 481
- disabling an app 758
- display
  - switching
    - between web view and native page 318
- distribution structure 63
  - Worklight Server 63
- documentify 535
  - JSONStore 535
- Dojo 184, 224, 246, 445, 456
  - code 228
  - code migration 228
  - iOS fixes 228
  - migration 228
  - toolkit 228
- Dojo library project 184
  - removing 285
  - setup 281

- Dojo Mobile 279
- Dojo tooling 184, 246

## E

- Eclipse 35
  - supported versions 9
- editors
  - Rich Page Editor 290
- embedded server
  - logging 836
- embedded WebSphere Application Server
  - Liberty Profile
    - logging 836
- encrypted data store 236
- enhance 535
  - JSONStore 535
- environment
  - what's new 22
- environments 239
  - production 663
  - QA 663
  - test 663
- error codes 401
- event types 842
- exporting
  - adapter configuration files 736
- extracting
  - public signing keys 323

## F

- Facebook apps
  - migrating 188
- failure 401
- feature comparison 839
- feature table 9
- feature-platform matrix 9
- features
  - Worklight Studio 383
- federal 830
- Federal Desktop Core Configuration 830
- Federal Information Processing Standards 830
- folder
  - application 247
- form-based authenticator 428
- FormBasedAuthenticator 428
- framework 456
  - adapter 360
    - Dojo 445
  - globalization 445

## G

- geofence
  - buffer zones 475
  - confidence levels 475
  - creating 475
- geolocation 850
- geospatial 850
- getErrorMessage 539
  - JSONStore 539
- getPushRequired 539
  - JSONStore 539
- getting started 25

- getting started (*continued*)
  - samples 25
  - tutorials 25
- globalization 445, 456, 466
  - limitations 9
- glossary 895

## H

- header authenticator 429
- header login module 434
- HeaderAuthenticator 429
- HeaderLoginModule 434
- HTTP adapter
  - <authentication> element 370
  - <connectionPolicy> element 368
  - <proxy> element 371
  - root element 368
- HTTP adapters 360
- hybrid app 246, 278
  - development 246
    - user interface 278
  - user interface
    - development 278
- hybrid application 241
- hybrid applications 445
  - accessibility 469
- hybrid development 1
- hybrid mixed development 1

## I

- IBM DB2 416
- IBM Worklight 3, 241, 456
  - application authenticity
    - overview 441
  - integrating IBM Endpoint Manager 885
  - security 417
    - application authenticity 441
    - IBM Endpoint Manager 885
    - security overview 417
- IBM Worklight client-side API 236
- IBM Worklight Consumer Edition 8
- IBM Worklight Enterprise Edition 8
- IBM Worklight server-side API 237
- icons
  - specifying
    - Android apps 322
    - iPhone apps 322
- implementing
  - adapter procedures 381
    - HTTP 382
  - device provisioning 439
- including and excluding features
  - what's new 17
- initialization options 246
- initOptions.js 246
- inner application 241
- installation 43
  - limitations 9
    - Worklight Server 43
- Installation 35
- Installation Manager 35

- installing
  - custom IBM Worklight database workload standards 741
  - IBM Mobile Application Platform Pattern Type 741
  - IBM Worklight support for PureApplication System 741
  - tools 37
- Installing
  - Adobe AIR tools 37
  - Android tools 38
  - iOS tools 37
  - Mobile Test Workbench for Worklight 40
  - test
    - workbench 40
  - WebWorks 38
  - Windows 8 tools 39
  - Windows Phone 7.5 tools 39
  - Windows Phone 8 tools 39
  - Worklight Studio 35
    - into an Eclipse IDE 36
    - with Rational Team Concert V4.0 36
  - Xcode 37
- installing IBM Worklight PureApplication System Extension for Worklight Studio 742
- invoking
  - back-end services 389
- iOS
  - API 620
  - Objective-C 620
- iOS tools
  - installing 37
- isPushRequired 542
  - JSONStore 542
- iwap 853
- IWAP 174, 841, 843, 857, 858

**J**

- Java ME 224
- JavaScript 445, 456
  - Rhino container 381
- JavaScript API 278
- JavaScript files
  - concatenation of 351, 356
  - minification of 351, 354
- JavaScript frameworks 445
  - accessibility 469
- JavaScript toolkits 279
- JavaScript UI framework 279
- JMS adapter
  - <connectionPolicy> element 375
  - <jmsConnection> element 376
  - <namingConnection> element 375
  - root element 374
- JMS adapters 360
- JNDI 723
- jQuery 445, 456
- jQuery Mobile 279
- JS files
  - concatenation of 351, 356
  - minification of 351, 354
- JSON translation capability 237
- JSONStore 236, 401, 830

- JSONStore (*continued*)
  - count 533
  - destroy 534
  - documentify 535
  - enhance 535
  - getErrorMessage 539
  - getPushRequired 539
  - isPushRequired 542
  - load 543
  - push 543
  - pushRequiredCount 545
  - toString 547

**K**

- keys
  - extracting 323
- known limitations 9, 893

**L**

- language 456
- LDAP
  - Application Center on WebSphere Application Server V7 125
  - Application Center on WebSphere Application Server V8 126, 127, 128
- LDAP login module 434
- LdapLoginModule 434
- Liberty 871
- Liberty Profile 710
- library
  - adapter 237
- Lightweight Directory Access Protocol
  - Application Center on WebSphere Application Server V7 125
  - Application Center on WebSphere Application Server V8 126, 127, 128
- limitations 9
- Linux
  - browser configuration 294
  - XULRunner browser configuration 294
- load 543
  - JSONStore 543
- locale 456
- location services 850
  - Android support 472
  - application in background 483
  - differentiating between indoor areas 477
  - geofence 475
  - indoor areas 477
  - iOS support 472
  - overview 470
  - securing server resources 480
  - setting acquisition policy 474
  - tracking devices 481
  - triggers 472
- logging 833, 836
- login form 262
- login forms
  - web widgets 329
- login modules 424
  - attributes 431
  - database 432

- login modules (*continued*)
  - header 434
  - LDAP 434
  - non-validating 432
  - single identity 433
  - WASLTPAModule 434
- login screen
  - screen widgets
    - setting size 329
- logs
  - location 833
  - monitoring 833
- LTPA authenticator 430

**M**

- main file, of application 249
- migrating 181, 184
- Migrating
  - Facebook apps 188
- migration 181, 191, 192, 224
- minification 342, 351, 354
  - what's new 17
- minification engine 354
- miscellaneous 23
- mobile applications
  - building 285
  - patterns 301
  - running 285
- mobile browser simulator 233
  - testing 307
- mobile devices
  - creating web pages 299
- mobile navigation
  - view 303
- mobile patterns 301
- Mobile SDKs
  - installing 37
  - tools 37
- Mobile Web apps 346
- mobile web pages
  - Mobile Navigation view 303
- mobileSecurityTest 421
- modifying
  - adapters 737
- monitoring 3, 833
- MPNS response codes 415
- multi-language 456
- MySQL 116
  - stale connections 116

**N**

- native and web development technologies 233
- native API 241
- native applications
  - accessibility 469
- native development 1
- native pages
  - overview 318
- non-validating login module 432
- NonValidatingLoginModule 432



## O

- offline 3
- operating systems
  - supported 9
- optimizing Worklight applications 342
- Oracle databases 75, 77, 78, 690, 692, 693
  - Apache Tomcat server 78, 693
  - manual configuration 75, 77, 78, 690, 692, 693
  - manual setup 75, 690
  - WebSphere Application Server 77, 692
- overview
  - adapters 360
  - location services 470

## P

- persistent cookie authenticator 429
- PersistentCookieAuthenticator 429
- plug-in
  - globalization 450, 453
  - Mobile
    - jQuery 450
    - Sencha Touch 453
- port number
  - of application server 40
- preferences
  - Rich Page Editor 295
- procedures
  - invoking 388
  - running 388
  - testing 388
- production environment 191, 192, 224
- projects 239
  - anatomy 246
  - building
    - Ant task 665
  - composition 246
  - creating 240
  - deploying
    - Ant task 730
  - Dojo library 285
  - overview 246
- Properties view
  - displaying tag information 302
- provisioning 436
  - devices 439
- public signing keys
  - extracting 323
- push 543
  - JSONStore 543
- push notification 410, 411, 416
  - architecture 410, 411
  - proxy settings 410
- push notifications 3, 224, 445, 466
- push notification 416
- pushRequiredCount 545
  - JSONStore 545

## Q

- queues 852
- quick fix 383

## R

- raw data 861
  - RDBMSLoginModule 432
  - realms
    - authentication 423
    - configuring 426
  - receiving data
    - Java page 320
    - Objective-C page 318
  - reducing application size 342, 343, 346, 347, 354, 356
  - reducing the application size
    - what's new 17
  - remote disable 758
    - default behavior 758
    - modifying the default behavior 758
  - remoteDisable 758
  - replacing
    - adapters 737
  - Report viewer 869
  - reports 839, 861
  - resources
    - accessibility 469
  - returning control
    - from Java page 321
    - from Objective-C page 319
  - Rhino container 381
  - Rich Page Editor 290
    - browser configuration 293
    - browser requirements 292
    - creating web pages 299
    - editing HTML files 297
    - limitations 9
    - opening web pages 296
    - setting preferences 295
    - views
      - design view 297
      - source view 297
      - split view 297
    - web pages
      - adding elements 302
  - root element
    - Cast Iron adapter 374
    - HTTP adapter 368
    - JMS adapter 374
    - SQL adapter 371
  - run time skinning 236
  - running
    - back-end services 389
  - runtime skinning 233
- ## S
- samples 25
  - screen widgets
    - setting size of login screen 329
  - scripts 249
  - SDK
    - See Worklight
  - search 840, 844
  - security 3
    - BlackBerry 10
      - creating QNX environment 327
    - configuration 88
    - IBM Endpoint Manager 885
    - overview 417

- security framework 237
    - overview 417
  - security tests 421
  - securityTest 421
  - Sencha Touch 279, 445, 456
  - server resources
    - securing 480
  - server-side application code 237
  - setting size
    - login screen
      - screen widgets 329
    - settings page 289
  - shell 233
  - shell component 241
  - signing
    - AIR applications 329
    - Windows 8 apps 330
  - single identity login module 433
  - SingleIdentityLoginModule 433
  - skins 239
    - applying 288
    - deleting 288
    - developing 288
  - SOAP
    - invoking services 382
  - software development kits
    - supported 9
  - source control 244
  - specifying
    - icons
      - Android apps 322
      - iPhone apps 322
    - taskbar
      - AIR 328
  - splash images 249
  - splash screens 249
  - SQL adapter
    - <connectionPolicy> element 372
    - root element 371
  - SQL adapters 360
  - SSL 224
  - starting
    - Worklight Studio
      - Developer Edition 37
  - status codes 410
  - Studio
    - features 383
    - style sheets 249
  - submitting
    - apps 736
  - supported browsers
    - Rich Page Editor 292
  - switching display
    - between web view and native page 318
  - synchronization
    - See data synchronization
  - system messages 456
- ## T
- tags
    - displaying information 302
  - taskbar
    - AIR
      - specifying 328
  - Tealeaf 855, 857, 858



- Tealeaf CX 856
- Technote 893
- test server
  - logging 836
- testing 3
  - mobile applications 294, 305, 307, 308, 309
  - mobile browser simulator 294, 305, 307, 308, 309
- thumbnail images 249
- tools
  - installing 37
- toString 547
  - JSONStore 547
- translation 456
- triggers 472
- troubleshooting 893
  - Cast Iron adapter 360
  - data synchronization 410
- tutorials 25

## U

- unified
  - push notifications 237
- United States Government Configuration Baseline 830
- update 3
- upgrading 191, 192, 224
  - databases 203
  - Worklight Studio 189
- Upgrading
  - Worklight Studio 190
- url
  - worklight server 289
- user interface 278, 279
  - development
    - hybrid app 278
  - hybrid app
    - development 278

## V

- validation
  - adapters 383
- version 181
- version control 244
- Virtual Member Manager
  - Application Center Access Control List 126
- Visual Studio 39
- Visual Studio 2010 39
- Visual Studio 2012 39
- VMM
  - Virtual Member Manager 126

## W

- WASLTPAModule login module 434

- web and native pages
  - interaction 318
- web browsers
  - supported 9
- web development 1
- web pages
  - adding elements
    - Rich Page Editor 302
  - creating in Rich Page Editor 299
  - opening in Rich Page Editor 296
- web services 445
  - Cordova 465
  - globalization 465
- web widget authentication
  - configuring 329
- web widgets
  - login forms 329
- webSecurityTest 421
- WebSphere 174, 710, 871
- WebSphere Application Server 416
  - manual configuration 77, 692
- WebSphere Application Server Network Deployment
  - configuring application servers
    - Ant tasks 708
- WebSphere Application Server V7
  - configuring LDAP for Application Center 125
- WebSphere Application Server V8
  - configuring LDAP for Application Center 127
  - managing ACL for Application Center with LDAP 128
- WebSphereFormBasedAuthenticator 430
- WebSphereLoginModule 434
- WEBWORKS\_HOME
  - installing 38
- what's new 23
  - analytics 15
  - API 20
  - build settings 17
  - concatenation 17
  - device type 22
  - Dojo 20
  - environment 22
  - external library 20
  - including and excluding features 17
  - minification 17
  - reducing the application size 17
- widgets
  - embedding in web pages 330
  - login forms 329
- WiFi 850
- Windows 39
- Windows 8 apps
  - signing 330
- Windows 8 tools
  - installing 39
- Windows Phone 7.5 tools
  - installing 39
- Windows Phone 8 224

- Windows Phone 8 (*continued*)
  - authenticated push 415
  - push notifications 415
- Windows Phone 8 tools
  - installing 39
- WL.Client.connect 317
- WL.JSONStore.destroy 534
- WL.JSONStore.documentify 535
- WL.JSONStore.get(collectionName).count 533
- WL.JSONStore.get(collectionName).getPushRequired 539
- WL.JSONStore.get(collectionName).isPushRequired(doc) 542
- WL.JSONStore.get(collectionName).load 543
- WL.JSONStore.get(collectionName).push 543
- WL.JSONStore.get(collectionName).pushRequiredCount 545
- WL.JSONStore.getErrorMessage 539
- WLClient 620
- WLPush 620
- working with multiple Worklight Servers 268
- Worklight applications
  - accessibility 469
- Worklight build process 354, 356
- Worklight Console 266
  - Access Disabled 758
  - Active 758
  - administering apps and adapters 735
- Worklight Development Server 266
- Worklight projects
  - changing 287
  - creating 286
  - Dojo version 287
  - Dojo-enabled 286
- Worklight Server 176, 191, 192, 224
  - changing target server 268
  - URL to the console 266
- worklight studio 233
- Worklight Studio
  - Developer Edition
    - starting 37
  - features 383
  - new console 266
  - new server 268
  - upgrading 189
- WorkLightLoginModule
  - Interface 224
- www 246

## X

- Xcode
  - installing 37
- XULRunner browser
  - browser configuration 294

## Z

- Zune Software 39